# MatrixSplat: Matrix-Parallel and Feature-Scalable 3D Gaussian Splatting Rendering on GPUs

**Yangming Zhang**[1] **Siyi Wu**[1] **Jian Wang**[2] **Bingzhe Li**[3]
**Wei Niu**[4] **Sheng Di**[5] **Ming Li**[1] **Miao Yin**[1]
[1]The University of Texas at Arlington [2]Snap Inc. [3]The University of Texas at Dallas
[4]University of Georgia [5]Argonne National Laboratory
{yxz0925, sxw8121}@mavs.uta.edu jwang4@snapchat.com bingzhe.li@utdallas.edu
wniu@uga.edu sdi1@anl.gov {ming.li, miao.yin}@uta.edu

## Abstract

3D Gaussian Splatting (3DGS) has emerged as a mainstream representation for real-time novel view synthesis and 3D reconstruction. However, existing 3DGS rendering pipelines are poorly aligned with modern GPU architectures: depth-sorted alpha compositing introduces strong per-pixel sequential dependencies, limiting parallelism and resulting in scalar Fused Multiply-Add (FMA)-dominated execution that underutilizes high-throughput matrix units such as Tensor Cores. This mismatch becomes more severe for multimodal and semantically enriched 3DGS, where high-dimensional features must be accumulated from millions of Gaussians, leading to rapidly increasing computation and memory costs. To address these challenges, we propose MatrixSplat, a matrix-parallel rendering framework for feature-scalable 3DGS on modern GPUs. We first identify the sequential dependency of alpha compositing as a key obstacle to efficient high-dimensional feature rendering, and adopt an order-independent weighted accumulation formulation that transforms feature rendering into matrix-oriented computation. Our framework features three key designs: *i)* a tile-based splatting pipeline that maps feature accumulation to warp-level matrix-multiply-accumulate execution on Tensor Cores; *ii)* a warp-granularity Gaussian packing strategy that mitigates visibility irregularity and improves feature reuse; and *iii)* a lazy channel-batched feature-fetching pipeline that reduces shared-memory pressure, maintains high warp occupancy, and overlaps memory access with matrix execution for high-dimensional features. Experiments on Mip-NeRF 360, LERF, and 3D-OVS demonstrate both high performance and strong scalability for feature-rich 3DGS rendering. Particularly, our method achieves up to 3.8× speedup over 3DGS and 2× over TC-GS while maintaining rendering quality and semantic accuracy, and further reduces render-kernel time by up to 36.7× for high-dimensional features. Integrated into existing fast renderers, it provides an additional 1.69–2.84× speedup.

## 1 Introduction

Modern GPU architectures are increasingly optimized for high-throughput matrix computation, yet many emerging graphics and vision workloads fail to fully exploit these capabilities. 3D Gaussian Splatting (3DGS) [19] has recently emerged as a mainstream paradigm for novel view synthesis and 3D reconstruction [3, 14, 17]. It represents a scene as a set of 3D Gaussian primitives, each parameterized by spatial position, scale, opacity, rotation, and spherical harmonic (SH) coefficients, enabling differentiable rendering with high visual quality and real-time performance [1]. However, the standard 3DGS pipeline relies on depth-sorted alpha compositing, where Gaussian contributions are accumulated in a strict front-to-back order. This design introduces strong per-pixel sequential

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

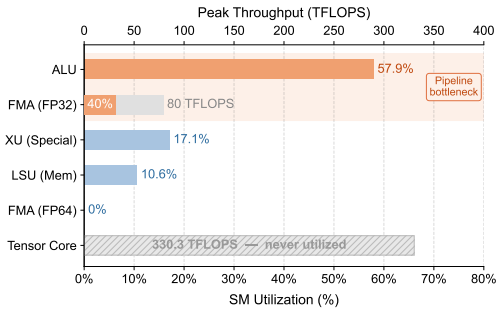


Figure 1: **SM utilization of 3DGS rendering.** Rendering is dominated by ALU and FMA (FP32), leaving Tensor Core never utilized.

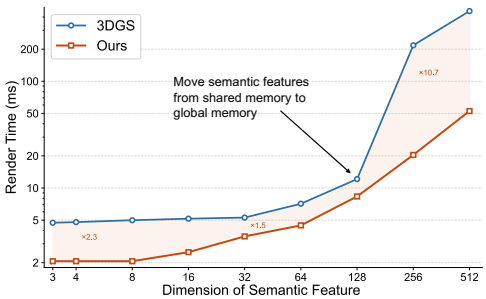


Figure 2: **Semantic feature scalability.** Rendering performance degrades dramatically with increasing feature dimensionality.

dependencies, limiting parallelism and causing frequent instruction stalls, with execution dominated by scalar Fused Multiply-Add (FMA) operations [29, 11, 8]. Consequently, the rendering pipeline is poorly aligned with modern matrix-computation-oriented GPU architectures, leaving high-throughput matrix units largely underutilized. As shown in Fig. 1, profiling of Streaming Multiprocessor (SM) utilization reveals that rendering is dominated by Arithmetic Logic Unit (ALU) and FP32 FMA operations, while Tensor Cores remain entirely unused despite offering up to 4× higher peak throughput. This mismatch exposes a ***fundamental gap*** between the dependency structure of current 3DGS rendering algorithms and the computing patterns of modern GPUs.

This challenge becomes significantly more severe as 3DGS evolves toward multimodal and semantically enriched representations. Recent advances extend 3DGS beyond photorealistic reconstruction by integrating high-dimensional features, enabling support for emerging applications such as world models in robotics, embodied AI, and autonomous driving [32, 41]. For example, prior works [39, 27, 18] embed high-dimensional language features into Gaussian primitives, enabling semantically aligned 3D understanding without relying on expensive and often low-quality 3D annotations. However, existing 3DGS rendering pipelines are not designed for such ***feature-scalable*** workloads, and the algorithm-hardware mismatch is therefore further magnified. As feature dimensionality increases, the accumulation workload over millions of Gaussians grows rapidly, resulting in substantially higher memory traffic and computational cost [16]. As shown in Fig. 2, the original 3DGS can no longer stage the full feature vector in shared memory when the dimension is greater than 128, so features are fetched from global memory. Rendering performance degrades dramatically with increasing feature dimensionality [27, 49], especially once features exceed the shared-memory capacity.

Prior works [7, 10, 33, 23, 48, 35] have attempted to address this mismatch from both algorithmic and hardware perspectives, but they accelerate different stages of the pipeline. Pipeline-level accelerators such as FlashGS [11], Speedy-Splat [14], and AdR-Gaussian [43] reduce the work *around* blending through tighter culling, scheduling, and sorting, yet still accumulate each pixel with a serial scalar loop. Sorting-free methods [15, 21, 31] replace sorted alpha compositing with weighted-sum or stochastic rendering to remove depth sorting, but execute the resulting accumulation with the same scalar FMAs. Tensor-Core approaches such as TC-GS [29] and GEMM-GS [26] map the alpha computation onto matrix units, while the blending loop itself remains order-dependent. Therefore, a gap remains: no existing method uses order independence to express the feature accumulation itself as matrix multiplication, even though this is the part whose cost grows with feature dimensionality [13].

To bridge this algorithm-hardware gap, we propose MatrixSplat, a matrix-parallel framework for feature-scalable 3DGS rendering on modern GPUs. Our key insight is that the sequential dependency introduced by alpha compositing is what prevents feature accumulation from being executed as matrix operations. By revisiting the rendering pipeline, we derive an order-independent weighted accumulation formulation that removes this dependency and transforms scalable feature rendering into matrix-oriented computation. Building on this formulation, we design a scalable tile-based splatting pipeline that maps feature accumulation to warp-level packed matrix-multiply-accumulate (MMA) execution. To further support high-dimensional semantic features, we introduce a lazy channel-batched feature-fetching strategy that preserves data reuse, reduces shared-memory pressure, and maintains high execution efficiency as feature dimensionality grows. Together, these designs enable efficient and scalable 3DGS rendering by leveraging high-throughput matrix units on modern

GPUs. Because our method changes how blending is executed rather than how much work reaches it, it is complementary to pipeline-level accelerators and can be combined with them.

Our contributions are summarized as follows:

- We analyze the 3DGS rendering pipeline and identify the sequential dependency of alpha compositing as a key bottleneck for feature-embedded Gaussian rendering on modern GPUs. Inspired by prior order-independent rendering ideas, we tailor weighted accumulation to semantic feature rendering and reorganize the resulting dependency-free computation into a hardware-aligned, matrix-oriented execution pattern.
- We propose a warp-level, matrix-oriented design that enables efficient mapping of feature-scalable rendering to Tensor Core MMA execution. By packing valid Gaussians at warp level, our method enables efficient matrix-computation-friendly feature accumulation while reducing visibility irregularity. This design improves the reuse of Gaussian features.
- We develop a scalable, lazy, channel-batched feature-fetching pipeline. This lazy loading reduces shared-memory pressure as feature dimensionality grows, preserves higher active warps per SM, and hides latency by overlapping memory loading and MMA execution, thereby supporting efficient rendering of high-dimensional features.

We evaluate our method on Mip-NeRF 360, LERF, and 3D-OVS. Experiments show that our method achieves up to 3.8× end-to-end speedup over 3DGS and 2× over TC-GS while maintaining visual quality and semantic accuracy, and further reduces render-kernel time by up to 36.7× for high-dimensional semantic features. It is also 2.2–3.5× faster than Sort-Free rendering and provides an additional 1.69–2.84× speedup when integrated into existing fast renderers.

## 2 Related Works

**Feature-rich 3D Gaussian Splatting.** Gaussian Grouping [45], LangSplat [39], and Semantic Gaussians [12] attach identity or CLIP-aligned features to Gaussians for segmentation, editing, and open-vocabulary queries. Because rendering cost grows with feature dimensionality, these systems usually compress features before rendering: Feature-3DGS [49] renders a low-dimensional field followed by a decoder, LangSplat compresses 512-D CLIP features into a 3-D latent space, and LangSplatV2 [28] replaces decoding with sparse coefficient splatting. ClunoGS [44] further compacts language Gaussians by jointly sparsifying and quantizing them in a unified optimization. Since feature accumulation remains a dominant cost [46], we accelerate it directly so that features can be rendered at their native dimension.

Table 1: **Positioning of related methods.** TC: Tensor Cores.

| Method | Stage | Order | Dim. |
|---|---|---|---|
| FlashGS, Speedy-Splat, AdR-Gaussian, HiGS | Culling, scheduling | Sorted | RGB |
| TC-GS, GEMM-GS | Alpha (TC) | Sorted | RGB |
| ORANGE | Blending (NPU) | Sorted | RGB |
| Sort-Free, StochasticSplats | Sorting | None | RGB |
| Duplex-GS | Sorting | Proxy | RGB |
| Feature-3DGS, LangSplat(V2) | Decoding | Sorted | Compr. |
| **MatrixSplat** | **Accumulation (TC)** | **None** | **Native** |

**Fast 3DGS rendering.** Pipeline-level accelerators reduce the work around blending while keeping in-order compositing (Tab. 1). StopThePop [40] adds hierarchical re-sorting and culling for view-consistent rendering, FlashGS [11] combines precise Gaussian-tile intersection with kernel-level scheduling, Speedy-Splat [14] tightens Gaussian localization and prunes primitives, AdR-Gaussian [43] culls with adaptive radii and balances load across threads, and HiGS [36] decouples binning and rasterization tile sizes. Compression and pruning methods [10, 23, 33, 48] reduce the number of Gaussians, and dedicated accelerators [35, 46] design hardware for 3DGS rendering. Matrix-unit methods such as TC-GS [29], GEMM-GS [26], and ORANGE [25] map alpha computation or blending onto Tensor Cores or NPU matrix units, but their blending loop remains order-dependent. We instead matrixize the accumulation itself, so our method is complementary to these approaches and can be combined with them (Sec. 5.3).

**Order-independent rendering.** Weighted blended OIT [9, 34] replaces ordered compositing with commutative weighted accumulation, which avoids sorting and the popping artifacts caused by unstable depth order. In 3DGS, Sort-Free Gaussian Splatting [15] learns depth-dependent weights through linear-correction weighted-sum rendering, StochasticSplats [21] replaces sorting with stochastic

rasterization, and Duplex-GS [31] combines proxy-level ordering with weighted blending to improve quality. These methods use order independence to avoid sorting, but still accumulate each pixel with scalar FMAs. We instead use it to change the compute primitive, turning the per-pixel accumulation into dense matrix products on Tensor Cores.

## 3 Background

**Order-Independent Transparency.** Rendering semi-transparent scenes requires modeling partial coverage, which is commonly achieved with the *over* operator [38]. Like Eq. 4, it is inherently order-dependent, because the transmittance of each primitive depends on all preceding primitives, introducing a loop-carried dependency during blending.

To mitigate the cost of sorting, many approximate compositing methods have been proposed. A notable advancement is weighted blended order-independent transparency (OIT) [34]. This approach replaces conventional front-to-back alpha blending with a commutative weighted-sum accumulation:

$$\boldsymbol{C} = \frac{\sum_{i\in\mathcal{N}} \boldsymbol{c}_i \cdot w\left(\alpha_i, d_i\right)}{\sum_{i\in\mathcal{N}} w\left(\alpha_i, d_i\right)} \left(1 - \prod_{i\in\mathcal{N}} \left(1-\alpha_i\right)\right) + \boldsymbol{c}_0 \prod_{i\in\mathcal{N}} \left(1-\alpha_i\right), \tag{1}$$

where $\boldsymbol{c}_0$ is the background color, and $\boldsymbol{c}_i$ and $\alpha_i$ represent the color and opacity of the $i$-th primitive. The function $w\left(\alpha_i, d_i\right)$ denotes a weight that decreases with distance. This approach eliminates the sorting stage and avoids visual "popping" artifacts caused by unstable depth ordering.

Based on this idea, Linear Correction Weighted Sum Rendering (LC-WSR) [6, 15] further refines occlusion modeling by assigning depth-dependent weights through a learnable linear function:

$$w(\alpha_i, d_i) = \alpha_i \max\left(0, 1 - \frac{d_i}{\sigma}\right) v_i, \quad i = 1, 2, \cdots, N. \tag{2}$$

where $\sigma$, $v_i$ are learnable parameters and $\alpha_i$ is the projected opacity of the $i$-th Gaussian at the pixel. LC-WSR assigns larger weights to near Gaussians while suppressing distant ones to zero.

**3D Gaussian Splatting.** Standard 3D Gaussian Splatting (3DGS) [19] represents a scene as a set of optimized anisotropic 3D Gaussians. Each Gaussian $g_i$ is defined by its center position $\boldsymbol{\mu}_i \in \mathbb{R}^3$, covariance matrix $\boldsymbol{\Sigma}_i$, opacity $o_i$, and spherical harmonics (SH) coefficients for view-dependent color. Given an arbitrary 3D position $\boldsymbol{x}$, the opacity contribution of the $i$-th Gaussian is defined as

$$\alpha_i(\boldsymbol{x}) = o_i \exp\left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_i)^\top \boldsymbol{\Sigma}_i^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_i)\right), \quad i = 1, 2, \cdots, N. \tag{3}$$

During rendering, each Gaussian is projected onto the image plane with 2D covariance $\boldsymbol{\Sigma}_i' = \boldsymbol{J}_i \boldsymbol{T}_i \boldsymbol{\Sigma}_i \boldsymbol{T}_i^\top \boldsymbol{J}_i^\top$, where $\boldsymbol{T}_i$ is the view transformation and $\boldsymbol{J}_i$ the Jacobian of the local affine approximation of the projection. For each pixel, the contributing Gaussians are sorted in depth order and composited using alpha blending. The final pixel color $\boldsymbol{C}$ is computed as

$$\boldsymbol{C} = \sum_{i\in\mathcal{N}} \boldsymbol{c}_i \alpha_i \prod_{j=1}^{i-1} (1-\alpha_j), \tag{4}$$

where $\boldsymbol{c}_i$ is the view-dependent color decoded from the SH coefficients, $\alpha_i$ is the projected opacity of the $i$-th Gaussian at the pixel, and $\mathcal{N}$ denotes the depth-ordered set of Gaussians contributing to pixel.

**Language 3D Gaussian Splatting.** LangSplat [39] distills 2D region-level CLIP embeddings into 3D Gaussian features, using SAM [22] to obtain hierarchical segmentation maps at the subpart, part, and whole levels, denoted $s$, $p$, and $w$. Then, LangSplat [39] learns a scene-wise autoencoder, where an encoder $E$ compresses the CLIP feature into a low-dimensional latent feature $H_t^l(v) = E\left(L_t^l(v)\right)$, and a decoder reconstructs the original CLIP embedding. Each Gaussian carries hierarchical latent features, rendered with the standard front-to-back alpha compositing:

$$\boldsymbol{F}^l(v) = \sum_{i\in\mathcal{N}} \boldsymbol{f}_i^l \alpha_i \prod_{j=1}^{i-1} \left(1-\alpha_j\right), l \in \{s, p, w\}, \tag{5}$$

where $\boldsymbol{F}^l(v)$ denotes the rendered latent semantic feature at pixel $v$.

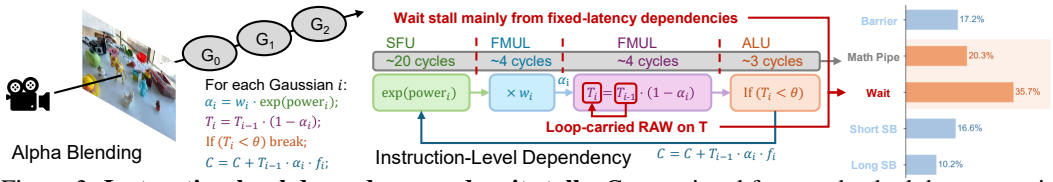


Figure 3: **Instruction-level dependency and wait stalls.** Conventional front-to-back alpha compositing introduces a long serial instruction sequence chain, leading to a strong wait stall.

# 4 Methodology

## 4.1 Revisiting the Bottlenecks in Feature-Embedded 3DGS Rendering on GPUs

To analyze the bottlenecks of feature-embedded Gaussian rendering, we first profile the original LangSplat pipeline. As shown in Tab. 2, rendering is the dominant stage, accounting for 77% of the total pipeline latency, and the majority of time is spent in the RenderCUDA kernel. This indicates that the RenderCUDA kernel is the primary system bottleneck.

Table 2: Pipeline latency breakdown.

| Stage | Kernel | % |
|---|---|---|
| Rendering | Preprocessing | 7.2 |
| | DuplicateWithKeys | 4.1 |
| | Sorting | 14.3 |
| | RenderCUDA | 51.3 |
| Decoding | — | 19.2 |
| Post-Processing | — | 3.8 |

We further analyze the stall behavior of the RenderCUDA kernel in Fig. 3. For each Gaussian, the current transmittance update must read back the accumulated result from the previous Gaussians ($T_{j+1} = T_j \cdot (1 - \alpha_j)$), forming a loop-carried read-after-write (RAW) dependency that enforces strict sequential execution. The stall breakdown results on the right of Fig. 3 are consistent with this dependency-limited behavior. The wait (fixed-latency data dependency on arithmetic operations) is the dominant stall type, primarily the computation instruction, with approximately 32 cycles of fixed latency. While the short scoreboard (short SB) accounts for only 10.2%. Critically, for the profiled workloads, these results suggest that the kernel is dependency-limited to a significant degree, rather than purely memory-bound. This loop-carried transmittance dependency prevents any cross-iteration instruction overlap within a warp, limiting the modern GPU's ability to hide instruction latency.

Inspired by those observations, we adopt order-independent blending to a parallel-friendly pattern and further reorganize high-dimensional feature rendering into a hardware-aligned pipeline.

## 4.2 Order-Independent Embedded Feature Blending

To remove the sequential dependency of alpha compositing (where each Gaussian's contribution depends on the accumulated transmittance of preceding Gaussians), we reformulate semantic Gaussian rendering as an order-independent weighted accumulation over Gaussian embeddings.

As shown in Eq. 4 and 5, the rendering process for RGB color and semantic features of Gaussian $g$ is essentially the same. Depth can also be included as an additional rendering attribute. Therefore, we concatenate these attributes into a unified embedded feature $\boldsymbol{e}_i$,

$$\boldsymbol{e}_i^\top = \text{concat}\left(\boldsymbol{c}_i,\ \boldsymbol{f}_i^l,\ [d_i]\right), \quad \boldsymbol{e}_i \in \mathbb{R}^{3+C+[1]}, \tag{6}$$

where $[d_i]$ denotes an optional depth and $C$ is the dimension of $\boldsymbol{f}_i^l$ mentioned in Eq. 5, so the embedding dimension is $D = 3 + C\,[+1]$. We assign each valid Gaussian $g$ a pixel-dependent contribution weight $w_{p,g}$ and compute the normalized per-pixel embedding $\boldsymbol{E}_p$ as

$$\boldsymbol{E}_p = \frac{\sum_{i\in\mathcal{N}} w_{p,i}\boldsymbol{e}_i + w_{\text{bg}}\boldsymbol{e}_{\text{bg}}}{\sum_{i\in\mathcal{N}} w_{p,i} + w_{\text{bg}}}, \tag{7}$$

where $w_{\text{bg}}$ and $\boldsymbol{e}_{\text{bg}}$ are the weight and embedding feature of the background. Because the weighted accumulation is commutative, the contribution of each Gaussian can be evaluated independently. We then adopt Eq. 2 as the weight function $w_{p,i}$, which decreases with distance. Following LC-WSR, $\sigma$ is a learnable scene-scale parameter, $v_i$ is a view-dependent scalar decoded from additional SH coefficients, and $w_{\text{bg}}$ is a learnable background weight; all are optimized jointly with the Gaussian parameters. We provide a detailed mathematical derivation of the blending formulation and its gradients in Appendix A, showing that the pipeline remains end-to-end differentiable.

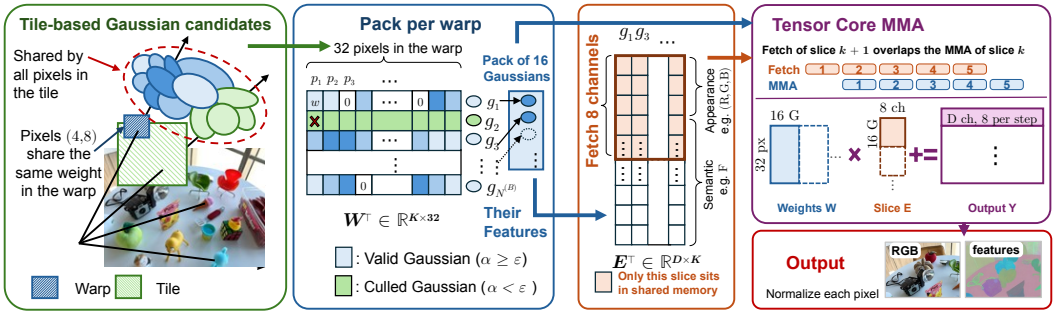


Figure 4: **Overview of our tile-based warp-level MMA rendering pipeline.** Each warp processes 32 pixels and packs 16 valid Gaussians into a weight matrix $\boldsymbol{W}$. Their features are fetched in 8-channel slices, so only the current slice resides in shared memory, and Tensor Cores multiply $\boldsymbol{W}$ with each slice while the next slice is being fetched.

This reformulation expresses rendering as a sum of independently weighted embedding features. In the next section, we introduce how to reorganize the heavily FMA computation into batched matrix multiply-accumulate (MMA) operations executed on Tensor Cores.

### 4.3 Packed Tile-Based Warp-Level MMA Execution

The original 3DGS employs a tile-based rasterizer to achieve fast rendering. This method partitions the screen into $16 \times 16$ tiles where each tile is assigned to one thread block. All warps within the block collaboratively load the Gaussians overlapping the tile into shared memory. During tile-based rasterization, all warps in the same block share the same candidate Gaussian list,

$$\mathcal{G}^{(B)} = \{g_1, g_2, \cdots, g_{N^{(B)}}\}, \tag{8}$$

where $B$ denotes the block index and $N^{(B)}$ is the number of candidate Gaussians for the current tile.

To efficiently map semantic blending onto Tensor Core execution, the computation granularity should be aligned with the warp-level MMA primitive. Accordingly, we design a tile-based, warp-level accumulation pipeline. Specifically, we partition each $16 \times 16$ tile into multiple $4 \times 8$ pixel groups, each processed by a warp. For warp $u$ in block $B$, let $\mathcal{P}^{(u)} = \{p_1, p_2, \cdots, p_{32}\}$ denote the set of pixels handled by this warp, as shown on the left of Fig. 4.

Directly issuing MMA operations over all tile-level candidate Gaussians is inefficient because many Gaussians have negligible opacity for a specific pixel. During rasterization, Gaussian-pixel pairs whose projected opacity $\alpha$ is smaller than $\varepsilon$ are culled. For each candidate Gaussian $g$, we first compute a pixel-level validity mask and use it to obtain the masked rendering weight $\bar{w}_{p,g}^{(u)}$:

$$m_{p,g}^{(u)} = \mathbf{1}\left(\alpha_{p,g} \geq \varepsilon\right), \qquad \bar{w}_{p,g}^{(u)} = m_{p,g}^{(u)} w_{p,g}, \quad p \in \mathcal{P}^{(u)}. \tag{9}$$

If $\sum_{p \in \mathcal{P}^{(u)}} m_{p,g}^{(u)} = 0$, the Gaussian $g$ does not contribute to any of the 32 pixels in warp $u$ and is skipped for this warp-level MMA execution. Otherwise, it is packed into the current flush batch. This design is effective in practice because the zero-voting only performs within a single warp and the pixels with the same warp are more spatially adjacent and usually have similar valid Gaussians.

Once collecting $K_{\text{mma}} = 16$ valid Gaussians for the current warp, we form a packed weight matrix and a packed embedding features matrix:

$$\boldsymbol{W}^{(u,t)} = \left[\bar{w}_{p_i,g_j}^{(u)}\right]_{i=1,j=1}^{32,16} \in \mathbb{R}^{32\times 16}, \qquad \boldsymbol{E}^{(u,t)} = \left[\boldsymbol{e}_{g_1}^{\top}, \boldsymbol{e}_{g_2}^{\top}, \cdots, \boldsymbol{e}_{g_{16}}^{\top}\right]^{\top} \in \mathbb{R}^{16\times D}. \tag{10}$$

Here, $t$ denotes the MMA flush-batch index, and $K_{\text{mma}} = 16$ is the reduction dimension of the `mma.m16n8k16` primitive used in our implementation, whose operands are stored in FP16 (Appendix B). The packed batch is then accumulated through Tensor Core:

$$\boldsymbol{Y}^{(u)} = \sum_{t=0}^{\lceil N^{(u)}/K_{\text{mma}} \rceil - 1} \boldsymbol{W}^{(u,t)} \boldsymbol{E}^{(u,t)}, \tag{11}$$

where $N^{(u)}$ denotes the number of Gaussians packed for warp $u$ after validity skip. Thus $\boldsymbol{W}^{(u)} \in \mathbb{R}^{32\times K}$ (drawn transposed in Fig. 4) holds any number $K = N^{(u)}$ of packed Gaussians, and each MMA flush consumes 16 of them. The output $\boldsymbol{Y}^{(u)} \in \mathbb{R}^{32\times D}$ holds the accumulated embeddings of the warp's pixels before the per-pixel normalization of Eq. 7.

While the scheduler issues MMA operations for the current packed batch, the warp can continue buffering valid Gaussians from shared memory to registers, overlapping data preparation with MMA execution latency so that Tensor Cores stay busy across flush batches.

### 4.4 Scalable MMA Rendering for High-Dimensional Embedded Features

In the traditional render implementation, the whole feature vector of each overlapping Gaussian is preloaded into shared memory before blending. As the embedded feature dimension $D$ increases, the shared memory used by each block grows proportionally. This sharply reduces the number of resident warps on each SM and limits the ability of the scheduler to hide latency.

To address this issue, we adopt a lazy batched channel feature fetching strategy. We divide the feature dimension into a channel batch of 8, which is aligned with the MMA output-channel tile. With $C_{\mathrm{mma}} = 8$, each MMA step computes an 8-channel slice:

$$\boldsymbol{Y}^{(u)}_{:,\,8q:8q+7} = \sum_{t=0}^{\lceil N^{(u)}/K_{\mathrm{mma}}\rceil-1} \boldsymbol{W}^{(u,t)}\boldsymbol{E}^{(u,t)}_{:,\,8q:8q+7}, \tag{12}$$

where $q = 0, \cdots, \lceil D/8\rceil - 1$ indexes the channel tile. At each flush MMA operation, the kernel loads only the required 8-dimensional feature slice for the current Gaussians, and the next slice is loaded while Tensor Cores process the current one (Fig. 4). This design reduces shared-memory pressure while keeping the same matrix-oriented structure as low-dimensional rendering.

## 5 Performance Evaluation

### 5.1 Experimental Setups

**Datasets & Metrics.** We evaluate our method on multiple datasets to assess both visual reconstruction quality and semantic feature performance. For image reconstruction, we use the Mip-NeRF 360 dataset [2], with segmentation labels annotated by GAGS [37], which provides complex visual scenes with associated semantic information. We report reconstruction fidelity using Peak Signal-to-Noise Ratio (PSNR), Structural Similarity (SSIM), and Learned Perceptual Image Patch Similarity (LPIPS). To evaluate semantic feature quality, we follow LangSplat [39] and conduct experiments on the LERF [20] and 3D-OVS [30] benchmarks, which are specifically designed for language-driven 3D understanding tasks. On these datasets, we report mean Intersection over Union (mIoU) and localization accuracy for 3D semantic segmentation and object localization.

**Baselines.** We compare with 3DGS [19] using the LangSplat feature pipeline [39] and with TC-GS [29]. For order-independent methods, Sort-Free Gaussian Splatting [15] has no official implementation, so we use a publicly available third-party implementation, and we retrain Duplex-GS [31] with its official code. For fast renderers, we use the official code of FlashGS [11], GEMM-GS [26], Speedy-Splat [14], and AdR-Gaussian [43], and integrate our method by replacing only their blending kernel while keeping their culling, scheduling, and sorting unchanged.

**Implementation Details.** To ensure fair comparison, we conduct all main experiments on an NVIDIA RTX 4090 GPU using the same settings as the original 3DGS [19]. For FPS evaluation, we perform several warm-up iterations prior to measurement and report results based on repeated timed runs with GPU synchronization before and after each render. Following TC-GS [29], we adopt a mixed-precision strategy that combines FP16 MMA operations with standard FP32 FMA operations. Main results use an end-to-end timing protocol, and comparisons with acceleration methods use the kernel-isolated protocol of FlashGS [11] (Appendix C). Results on RTX 6000 Ada and H100 GPUs, which show consistent speedups across architectures, are reported in Appendix F.

Table 3: **Quantitative comparison on Mip-NeRF 360 (OVS) and LERF.** We report image reconstruction quality, semantic performance, and end-to-end rendering speed. Following LangSplat [39], mIoU and localization accuracy (LocAcc) are evaluated based on rendered language features. FPS follows the end-to-end protocol (Appendix C) on an RTX 4090. We use 3DGS equipped with the TC-GS [29] module as 3DGS(+TC-GS). Best results for each metric are **bold**; LPIPS is lower-is-better, and all other metrics are higher-is-better.

| | Mip-NeRF 360 | | | | | | LERF | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Method | PSNR | SSIM | LPIPS | mIoU | LocAcc | FPS | PSNR | SSIM | LPIPS | mIoU | LocAcc | FPS |
| 3DGS [19, 39] | 30.12 | 0.910 | 0.181 | 0.536 | 0.698 | 156.0 | 23.91 | **0.856** | 0.219 | 0.419 | 0.632 | 237.9 |
| 3DGS(+TC-GS) [29] | 30.14 | **0.911** | 0.183 | 0.536 | 0.698 | 295.1 | **23.97** | **0.856** | **0.218** | 0.419 | 0.632 | 310.3 |
| MatrixSplat | **30.35** | **0.911** | **0.178** | **0.541** | **0.730** | **599.9** | 23.80 | 0.851 | 0.225 | **0.445** | **0.679** | **702.6** |

## 5.2 Quantitative Results

Tab. 3 and Tab. 4 summarize the main quantitative results on Mip-NeRF 360 [2], LERF [20], and 3D-OVS [30]. Following LangSplat [39], mIoU and localization accuracy (LocAcc) are evaluated based on rendered language features. TC-GS's [29] render quality remains very close to 3DGS because it does not change the rendering pipeline. Our method consistently achieves notable rendering speed improvements across all datasets while maintaining competitive reconstruction quality and semantic accuracy compared with both the vanilla 3DGS [19] and TC-GS [29].

Table 4: **Quantitative comparison on 3D-OVS.** Same setting as Tab. 3.

| | 3D-OVS | | | | | |
|---|---|---|---|---|---|---|
| Method | PSNR | SSIM | LPIPS | mIoU | LocAcc | FPS |
| 3DGS [19, 39] | 24.71 | **0.757** | **0.234** | 0.802 | 0.937 | 162.0 |
| 3DGS(+TC-GS) [29] | 24.66 | 0.749 | 0.234 | 0.802 | 0.937 | 294.8 |
| MatrixSplat | **24.89** | 0.752 | 0.239 | **0.804** | **0.985** | **510.9** |

On the Mip-NeRF 360 dataset, our method achieves 599.9 FPS, substantially outperforming the TC-GS by 2×. Despite this large speedup, it maintains high visual quality and even achieves the best PSNR and LPIPS among all methods. On the LERF dataset, our method reaches 702.6 FPS, nearly 3× faster than 3DGS. Tab. 4 shows the quantitative comparison on 3D-OVS. Per-scene analysis shows that the fidelity change is scene-dependent: PSNR drops by 0.23 dB on *teatime*, where heavy semi-transparent overlap makes exact ordering important, but rises by 0.73 dB on *counter*, where avoiding view-dependent sorting errors helps. Nevertheless, the overall results show that our method delivers significant speedups while largely maintaining the quality of rendered features, including both appearance and language embeddings across all three datasets.

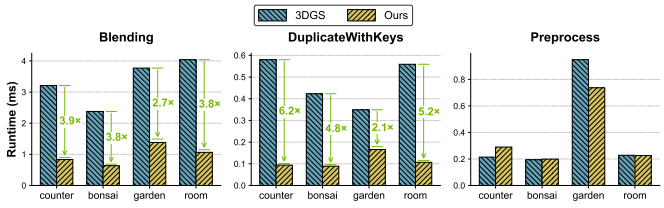


Figure 5: **Runtime breakdown of the rendering kernel on the Mip-NeRF 360 datasets.**

To further evaluate the improvement of our method for each stage, we conduct a comprehensive runtime breakdown of the most expensive kernels as listed in Sec. 4.1: blending, DuplicateWithKeys, and preprocessing. Here, our blending kernel handles both rendering and sorting, so we combine the sorting and rendering runtime for 3DGS. Fig. 5 shows that our speedup mainly comes from the blending stages, which are also the dominant bottlenecks in the original 3DGS pipeline. Our Tensor-Core-oriented MMA computation reduces the latency of the render kernel by 2.1–3.0×. In the 3DGS pipeline, each Gaussian-tile pair is associated with a depth-aware 64-bit key, making key construction and radix sorting expensive. In contrast, our method uses only the tile ID as key and completely removes depth from the sorting, eliminating the need for per-tile depth ordering and reducing the cost of key construction and radix sorting.

## 5.3 Comparison with Order-Independent Methods and Fast Renderers

**Order-independent methods.** Tab. 5 separates the formulation from the execution strategy. Sort-Free rendering uses the same weighted-sum formulation with scalar execution and gains little speed, even running slower than 3DGS on LERF and Mip-NeRF 360. Duplex-GS attains the best PSNR,

Table 5: **Comparison with order-independent methods.** Kernel-isolated protocol (Appendix C), $D$=8, RTX 4090. Speedups over 3DGS are in parentheses. Sort-Free uses a third-party implementation. *Duplex-GS on 3D-OVS uses the official pretrained checkpoint; other Duplex-GS entries are retrained with the official code and default settings.

| | 3D-OVS | | | LERF | | | Mip-NeRF 360 | | |
|---|---|---|---|---|---|---|---|---|---|
| Method | FPS | PSNR | LPIPS | FPS | PSNR | LPIPS | FPS | PSNR | LPIPS |
| 3DGS [19] | 210.2 | 24.78 | 0.239 | 357.0 | 23.91 | **0.217** | 278.9 | 30.13 | 0.183 |
| Sort-Free (WSR) [15] | 240.3 | 24.55 | 0.239 | 256.1 | 23.90 | 0.227 | 215.7 | 30.35 | **0.178** |
| Duplex-GS [31] | 319.1 | **25.40*** | **0.171*** | 332.9 | **24.46** | 0.223 | 127.9 | **30.46** | 0.222 |
| MatrixSplat | **521.8** (2.48×) | 24.88 | 0.234 | **891.9** (2.50×) | 23.80 | 0.228 | **660.0** (2.37×) | 30.35 | 0.181 |

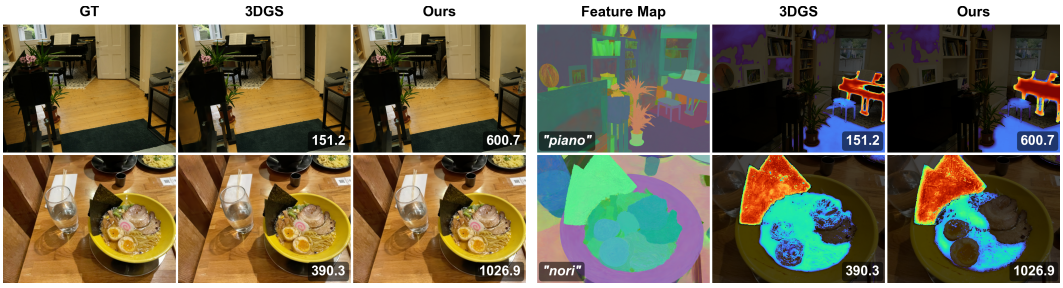


Figure 7: **Visual comparison of quality and language accuracy** on Room and Ramen. The number at the bottom right of each image is the rendering FPS.

but its MLP evaluation adds overhead. Our method is 2.2–3.5× faster than Sort-Free at comparable PSNR and 1.6–5.2× faster than Duplex-GS, so the speedup comes from changing the compute primitive rather than from the formulation alone.

**Integration with fast renderers.** Pipeline-level accelerators reduce the work around blending but keep serial scalar accumulation. Replacing only their blending kernel with ours gives an additional 1.69–2.84× speedup across FlashGS, GEMM-GS, Speedy-Splat, and AdR-Gaussian with PSNR changes of at most 0.24 dB, showing that the two kinds of optimization are complementary (Appendix E). Our speedup over 3DGS is also stable across GPU architectures: 2.37×, 2.36×, and 2.33× on RTX 4090, RTX 6000 Ada, and H100, respectively (Appendix F).

### 5.4 Scalability with Feature Dimensionality

Fig. 6 shows that the blending kernel runtime of the original 3DGS renderer increases rapidly as the semantic feature dimension grows. Our method consistently outperforms 3DGS across all evaluated feature dimensions, reducing blending-kernel time by about 3× on average at low dimensions and by up to 36.7× at 256 dimensions. The large gain at high feature dimensions validates our lazy batched-channel feature fetching strategy. Instead of loading the full feature vector of each Gaussian into shared memory, we load only the 8-channel slice needed by the current MMA flush, so the shared-memory footprint and warp occupancy stay constant as $D$ grows. Our method maintains high efficiency even for high-dimensional semantic features, leading to better scalability for the render kernel.

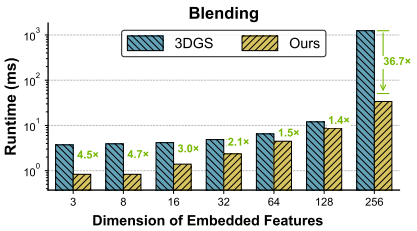


Figure 6: **Embedded feature scalability.** Our method consistently reduces blending kernel time over 3DGS as the feature dimension scales.

### 5.5 Visual Quality Comparisons

Fig. 7 compares 3DGS and our method on the room and ramen scenes. Our method better preserves the transparent water glass and the ramen texture while rendering at over 1,026 FPS, and it localizes queried regions such as the piano and nori with similar accuracy and sharpness. Additional comparisons are provided in Appendix G.

Table 6: **Component ablation across feature dimensions $D$.** Average FPS over 3D-OVS and Mip-NeRF 360 scenes at a lower resolution than Tab. 5; each row adds one component.

| Setting | $D$=8 | 16 | 32 | 64 | 128 | 256 | 512 |
|---|---|---|---|---|---|---|---|
| Vanilla 3DGS | 253.8 | 237.2 | 194.1 | 116.3 | 59.5 | 2.3 | 1.1 |
| + OIEFB | 423.5 | 363.2 | 273.5 | 143.7 | 64.9 | 2.1 | 1.2 |
| + MMA execution | 741.1 | 500.9 | 302.8 | 148.1 | 78.8 | 38.0 | 17.4 |
| + Warp-level packing | 830.5 | 568.2 | 369.0 | 170.8 | 79.3 | 35.7 | 22.4 |
| + Lazy channel-batched fetching | **830.8** | **622.6** | **399.5** | **269.1** | **144.5** | **72.1** | **41.1** |

### 5.6 Ablation Study

Tab. 6 adds our components one at a time, averaged over all 3D-OVS and Mip-NeRF 360 scenes. Order-independent embedded feature blending (OIEFB, Sec. 4.2) removes sorting and gives up to 1.67× at $D$=8, but it still uses scalar FMAs, so its benefit shrinks as $D$ grows. MMA execution is the main source of speedup. Warp-level packing reduces wasted MMA lanes and adds 12–22% for $D \leq 64$. Lazy channel-batched fetching matters most at high dimensions, adding 1.6–2.0× for $D \geq 64$. Overall, our method is 2.1–3.3× faster than 3DGS for $D \leq 128$ and 31–37× faster for $D \geq 256$. Reduced precision alone does not explain the gain: running 3DGS with FP16 features changes its speed by at most 17%, while our kernel is 2.2–4.8× faster on the same scenes; the gain instead comes from matrix execution and data reuse (Appendix D).

To demonstrate that order-independent blending alleviates the instruction dependencies introduced by alpha blending, we report the stall breakdown in Fig. 8 on 3DGS and our method. Our method reduces wait stalls by about 10% by replacing the serial accumulation in alpha blending with order-independent accumulation. Since all lanes within a warp cooperatively execute the same MMA operation, our method reduces synchronization overhead. Therefore, barrier stall also drops about 10%. Meanwhile, the MMA-based pipeline requires additional shared-memory staging for blending weights and intermediate matrices, which leads to increased Short Scoreboard stalls.

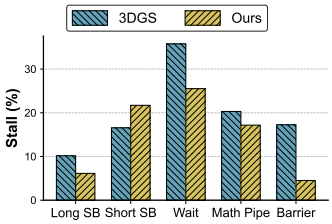

Figure 8: **Stall breakdown.**

## 6 Conclusion

**Summary.** We propose MatrixSplat, a matrix-parallel framework for feature-scalable 3D Gaussian Splatting rendering. We reformulate semantic feature rendering as an order-independent weighted accumulation and map it to warp-level packed MMA execution. We further introduce a lazy channel-batched feature-fetching strategy to reduce shared-memory pressure and maintain efficiency for high-dimensional features. Experiments on Mip-NeRF 360, LERF, and 3D-OVS show that MatrixSplat achieves up to 3.8× end-to-end speedup over 3DGS and 2× over TC-GS while largely maintaining visual quality and semantic accuracy, reduces render-kernel time by up to 36.7× for high-dimensional features, and provides an additional 1.69–2.84× speedup when integrated into existing fast renderers.

**Limitations and Broader Impact.** First, order-independent accumulation approximates depth-ordered compositing, so its effect on fidelity is scene-dependent: it degrades quality under extreme multi-layer semi-transparent overlap (e.g., −0.23 dB on *teatime*) and can improve it under moderate occlusion, and thresholds tuned for alpha compositing may need re-tuning. Second, we accelerate forward rendering; training also becomes faster, but by less, because the backward pass still relies on scattered gradient updates and atomic operations; register-interleaved gradient accumulation with warp-level reduction is a promising next step. Third, tile binning with radix sort, per-Gaussian preprocessing, and the warp-level packing scan remain non-matrix stages, which bound the end-to-end speedup once blending is accelerated. Finally, we target NVIDIA Tensor Cores via warp-level MMA; other matrix units such as NPUs are not evaluated. We will release our CUDA implementation and evaluation scripts. For broader impact, our method enables more efficient feature rendering for applications such as interactive scene understanding [47, 24], language-guided 3D editing and generation [4, 5], robotics [32], AR/VR, and 3D medical imaging [42].

## References


[1] Milena T Bagdasarian, Paul Knoll, Yi-Hsin Li, Florian Barthel, Anna Hilsmann, Peter Eisert, and Wieland Morgenstern. 3DGS.zip: A survey on 3D Gaussian splatting compression methods. *arXiv preprint arXiv:2407.09510*, 2024.

[2] Jonathan T Barron, Ben Mildenhall, Matthew Tancik, Peter Hedman, Ricardo Martin-Brualla, and Pratul P Srinivasan. Mip-NeRF: A multiscale representation for anti-aliasing neural radiance fields. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 5855–5864, 2021.

[3] Guikun Chen and Wenguan Wang. A survey on 3D Gaussian splatting. *arXiv preprint arXiv:2401.03890*, 2024.

[4] Haozhe Chi, Zhicheng Sun, Yang Jin, Yi Ma, Jing Wang, and Yadong Mu. Enhancing consistency of flow-based image editing through Kalman control. *Advances in Neural Information Processing Systems*, 38:141866–141898, 2025.

[5] Haozhe Chi, Jinghan Li, Hao Jiang, Wu Sheng, Yi Ma, Jing Wang, and Yadong Mu. Balancing image compression and generation with bootstrapped tokenization. *arXiv preprint arXiv:2606.05552*, 2026.

[6] Jifeng Dai, Haozhi Qi, Yuwen Xiong, Yi Li, Guodong Zhang, Han Hu, and Yichen Wei. Deformable convolutional networks. In *Proceedings of the IEEE international conference on computer vision*, pages 764–773, 2017.

[7] Xiaobin Deng, Changyu Diao, Min Li, Ruohan Yu, and Duanqing Xu. Efficient density control for 3D Gaussian splatting. *arXiv preprint arXiv:2411.10133*, 2024.

[8] Xiaobiao Du, Yida Wang, Kun Zhan, and Xin Yu. Mobile-GS: Real-time Gaussian splatting for mobile devices. *arXiv preprint arXiv:2603.11531*, 2026.

[9] Cass Everitt. Interactive order-independent transparency. *White paper, nVIDIA*, 2(6):7, 2001.

[10] Zhiwen Fan, Kevin Wang, Kairun Wen, Zehao Zhu, Dejia Xu, and Zhangyang Wang. LightGaussian: Unbounded 3D Gaussian compression with 15x reduction and 200+ FPS. *arXiv preprint arXiv:2311.17245*, 2023.

[11] Guofeng Feng, Siyan Chen, Rong Fu, Zimu Liao, Yi Wang, Tao Liu, Zhilin Pei, Hengjie Li, Xingcheng Zhang, and Bo Dai. FlashGS: Efficient 3D Gaussian splatting for large-scale and high-resolution rendering. *arXiv preprint arXiv:2408.07967*, 2024.

[12] Jun Guo, Xiaojian Ma, Yue Fan, Huaping Liu, and Qing Li. Semantic Gaussians: Open-vocabulary scene understanding with 3D Gaussian splatting. *IEEE Transactions on Circuits and Systems for Video Technology*, 2026.

[13] Seongbo Ha, Sibaek Lee, Kyungsu Kang, Joonyeol Choi, Seungjun Tak, and Hyeonwoo Yu. LangGS-SLAM: Real-time language-feature Gaussian splatting SLAM. *arXiv preprint arXiv:2602.06991*, 2026.

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

[15] Qiqi Hou, Randall Rauwendaal, Zifeng Li, Hoang Le, Farzad Farhadzadeh, Fatih Porikli, Alexei Bourd, and Amir Said. Sort-free Gaussian splatting via weighted sum rendering. *arXiv preprint arXiv:2410.18931*, 2024.

[16] Xiaotong Huang, He Zhu, Zihan Liu, Weikai Lin, Xiaohong Liu, Zhezhi He, Jingwen Leng, Minyi Guo, and Yu Feng. SEELE: A unified acceleration framework for real-time Gaussian splatting. *arXiv preprint arXiv:2503.05168*, 2025.

[17] Wenqi Jia, Zhewen Hu, Ying Huang, Yu Gong, Stavros Kalafatis, Yuke Wang, Wei Niu, Chengming Zhang, Ang Li, Sheng Di, et al. Splaxel: Efficient distributed training of 3D Gaussian splatting for large-scale scene reconstruction via pixel-level communication. *arXiv preprint arXiv:2606.18588*, 2026.

[18] Ying Jiang, Chang Yu, Tianyi Xie, Xuan Li, Yutao Feng, Huamin Wang, Minchen Li, Henry Lau, Feng Gao, Yin Yang, et al. VR-GS: A physical dynamics-aware interactive Gaussian splatting system in virtual reality. In *ACM SIGGRAPH 2024 conference papers*, pages 1–1, 2024.

[19] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3D Gaussian splatting for real-time radiance field rendering. *ACM Trans. Graph.*, 42(4):139–1, 2023.

[20] Justin Kerr, Chung Min Kim, Ken Goldberg, Angjoo Kanazawa, and Matthew Tancik. LERF: Language embedded radiance fields. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 19729–19739, 2023.

[21] Shakiba Kheradmand, Delio Vicini, George Kopanas, Dmitry Lagun, Kwang Moo Yi, Mark Matthews, and Andrea Tagliasacchi. StochasticSplats: Stochastic rasterization for sorting-free 3D Gaussian splatting. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 26326–26335, 2025.

[22] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C Berg, Wan-Yen Lo, et al. Segment anything. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 4015–4026, 2023.

[23] Joo Chan Lee, Daniel Rho, Xiangyu Sun, Jong Hwan Ko, and Eunbyung Park. Compact 3D Gaussian representation for radiance field. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 21719–21728, 2024.

[24] Boyi Li, Yifan Shen, Yuanzhe Liu, Yifan Xu, Jiateng Liu, Xinzhuo Li, Zhengyuan Li, Jingyuan Zhu, Yunhan Zhong, Fangzhou Lan, et al. Toward cognitive supersensing in multimodal large language model. *arXiv preprint arXiv:2602.01541*, 2026.

[25] Haomin Li, Yue Liang, Fangxin Liu, Bowen Zhu, Zongwu Wang, Yu Feng, Liqiang Lu, Li Jiang, and Haibing Guan. ORANGE: Exploring ockham's razor for neural rendering by accelerating 3DGS on NPUs with GEMM-friendly blending and balanced workloads. In *2026 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 1–15, 2026.

[26] Haomin Li, Bowen Zhu, Fangxin Liu, Zongwu Wang, Xinran Liang, Li Jiang, and Haibing Guan. GEMM-GS: Accelerating 3D Gaussian splatting on tensor cores with GEMM-compatible blending. In *2026 63rd ACM/IEEE Design Automation Conference (DAC)*, 2026.

[27] Qijing Li, Jingxiang Sun, Liang An, Zhaoqi Su, Hongwen Zhang, and Yebin Liu. SemanticSplat: Feed-forward 3D scene understanding with language-aware Gaussian fields. *arXiv preprint arXiv:2506.09565*, 2025.

[28] Wanhua Li, Yujie Zhao, Minghan Qin, Yang Liu, Yuanhao Cai, Chuang Gan, and Hanspeter Pfister. LangSplatV2: High-dimensional 3D language Gaussian splatting with 450+ FPS. *arXiv preprint arXiv:2507.07136*, 2025.

[29] Zimu Liao, Jifeng Ding, Siwei Cui, Ruixuan Gong, Boni Hu, Yi Wang, Hengjie Li, Hui Wang, Xingcheng Zhang, and Rong Fu. TC-GS: A faster Gaussian splatting module utilizing tensor cores. In *Proceedings of the SIGGRAPH Asia 2025 Conference Papers*, pages 1–9, 2025.

[30] Kunhao Liu, Fangneng Zhan, Jiahui Zhang, Muyu Xu, Yingchen Yu, Abdulmotaleb El Saddik, Christian Theobalt, Eric Xing, and Shijian Lu. Weakly supervised 3D open-vocabulary segmentation. *Advances in Neural Information Processing Systems*, 36:53433–53456, 2023.

[31] Weihang Liu, Yuke Li, Yuxuan Li, Jingyi Yu, and Xin Lou. Duplex-GS: Proxy-guided weighted blending for real-time order-independent Gaussian splatting. *IEEE Transactions on Circuits and Systems for Video Technology*, 2026.

[32] Yuanzhe Liu, Jingyuan Zhu, Yuchen Mo, Gen Li, Xu Cao, Jin Jin, Yifan Shen, Zhengyuan Li, Tianjiao Yu, Wenzhen Yuan, et al. PALM: Progress-aware policy learning via affordance reasoning for long-horizon robotic manipulation. *arXiv preprint arXiv:2601.07060*, 2026.

[33] Saswat Subhajyoti Mallick, Rahul Goel, Bernhard Kerbl, Markus Steinberger, Francisco Vicente Carrasco, and Fernando De La Torre. Taming 3DGS: High-quality radiance fields with limited resources. In *SIGGRAPH Asia 2024 Conference Papers*, pages 1–11, 2024.

[34] Morgan McGuire and Louis Bavoil. Weighted blended order-independent transparency. *Journal of Computer Graphics Techniques*, 2(4), 2013.

[35] Wenhui Ou, Zhuoyu Wu, Yipu Zhang, Dongjun Wu, Freddy Ziyang Hong, and Chik Patrick Yue. FLICKER: A fine-grained contribution-aware accelerator for real-time 3D Gaussian splatting. *arXiv preprint arXiv:2603.01158*, 2026.

[36] Dawid Pająk, Martin Bisson, and Rodolfo Lima. HiGS: A hierarchical rendering architecture for real-time 3D Gaussian splatting. *arXiv preprint arXiv:2606.00352*, 2026.

[37] Yuning Peng, Haiping Wang, Yuan Liu, Chenglu Wen, Zhen Dong, and Bisheng Yang. GAGS: Granularity-aware feature distillation for language Gaussian splatting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pages 8376–8384, 2026.

[38] Thomas Porter and Tom Duff. Compositing digital images. In *Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, pages 253–259, 1984.

[39] Minghan Qin, Wanhua Li, Jiawei Zhou, Haoqian Wang, and Hanspeter Pfister. LangSplat: 3D language Gaussian splatting. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 20051–20060, 2024.

[40] Lukas Radl, Michael Steiner, Mathias Parger, Alexander Weinrauch, Bernhard Kerbl, and Markus Steinberger. Stopthepop: Sorted Gaussian splatting for view-consistent real-time rendering. *ACM Transactions on Graphics (TOG)*, 43(4):1–17, 2024.

[41] Yifan Shen, Yuanzhe Liu, Jingyuan Zhu, Xu Cao, Xiaofeng Zhang, Yixiao He, Wenming Ye, James Rehg, and Ismini Lourentzou. Fine-grained preference optimization improves spatial reasoning in VLMs. *Advances in Neural Information Processing Systems*, 38:17929–17960, 2025.

[42] Kegeng Tang, Jingbo Wang, Shaogang Ren, and Zihao Wang. CT-$\Delta$Bench: A benchmark for longitudinal 3D medical imaging difference reporting with vision-language models. *arXiv preprint arXiv:2608.11534*, 2026.

[43] Xinzhe Wang, Ran Yi, and Lizhuang Ma. AdR-Gaussian: Accelerating Gaussian splatting with adaptive radius. In *ACM SIGGRAPH Asia 2024 Conference Papers*, pages 1–10, 2024.

[44] Jian Xu, Yi Li, Ying Huang, Siyi Wu, Bingzhe Li, Yuede Ji, and Miao Yin. ClunoGS: Compact language 3D Gaussian splatting via unified optimization. *arXiv preprint arXiv:2610.30013*, 2026.

[45] Mingqiao Ye, Martin Danelljan, Fisher Yu, and Lei Ke. Gaussian grouping: Segment and edit anything in 3D scenes. In *European conference on computer vision*, pages 162–179. Springer, 2024.

[46] Zhifan Ye, Yonggan Fu, Jingqun Zhang, Leshu Li, Yongan Zhang, Sixu Li, Cheng Wan, Chenxi Wan, Chaojian Li, Sreemanth Prathipati, et al. Gaussian blending unit: An edge GPU plug-in for real-time Gaussian-based rendering in AR/VR. In *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 353–365. IEEE, 2025.

[47] Tianjiao Yu, Xinzhuo Li, Yifan Shen, Onkar Susladkar, Yuanzhe Liu, Xiaona Zhou, and Ismini Lourentzou. ELSA3D: Elastic semantic anchoring for unified 3D understanding and generation. *arXiv preprint arXiv:2607.06565*, 2026.

[48] Yangming Zhang, Wenqi Jia, Wei Niu, and Miao Yin. GaussianSpa: An "optimizing-sparsifying" simplification framework for compact and high-quality 3D Gaussian splatting. *arXiv preprint arXiv:2411.06019*, 2024.

[49] Shijie Zhou, Haoran Chang, Sicheng Jiang, Zhiwen Fan, Zehao Zhu, Dejia Xu, Pradyumna Chari, Suya You, Zhangyang Wang, and Achuta Kadambi. Feature 3DGS: Supercharging 3D Gaussian splatting to enable distilled feature fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 21676–21685, 2024.

# A Derivation of Order-Independent Embedded Features Blending

We provide the detailed mathematical formulation and gradient derivation for the weighted order-independent embedded features blending used in Sec. 4.2.

**Forward Rendering.** Let $\mathcal{N}$ be the set of Gaussians overlapping a pixel $\boldsymbol{p}$, consistent with Eq. 4. Rasterizing Eq. 3 onto the image plane via EWA splatting projects the 3D covariance $\boldsymbol{\Sigma}_i$ to a 2D screen-space covariance $\boldsymbol{\Sigma}_{2D,i} = \boldsymbol{J}\boldsymbol{T}\boldsymbol{\Sigma}_i\boldsymbol{T}^\top\boldsymbol{J}^\top$, where $\boldsymbol{T}$ is the upper-left $3\times3$ submatrix of the view matrix and $\boldsymbol{J}$ is the Jacobian of the local affine approximation of the projective transformation. This yields the per-pixel spatial response

$$G_i(\boldsymbol{p}) = \exp\Big(-\tfrac{1}{2}\,\Delta\boldsymbol{p}_i^\top\,\boldsymbol{\Sigma}_{2D,i}^{-1}\,\Delta\boldsymbol{p}_i\Big), \quad \Delta\boldsymbol{p}_i = \boldsymbol{p} - \boldsymbol{\mu}_i^{2D}, \tag{13}$$

where $\boldsymbol{\mu}_i^{2D}$ is the projected 2D center of Gaussian $i$. The projected (vanilla) alpha $\alpha_i = o_i \cdot G_i(\boldsymbol{p})$ is the 2D rasterization counterpart of Eq. 3, with $o_i \in (0, 1)$ the raw opacity. Following previous work [15] and Eq. 2, the effective weight is

$$w_i = w(\alpha_i, d_i) = \alpha_i \max\left(0,\, 1 - \frac{d_i}{\sigma}\right) v_i, \tag{14}$$

where $v_i$ is the view-dependent visibility scalar evaluated from an additional spherical harmonics (SH), $d_i$ is the view-space depth of Gaussian $i$, and $\sigma$ is a learnable scene-scale parameter. Gaussians with $d_i \geq \sigma$ receive zero weight and are excluded from rendering. The normalization factor includes a learnable background weight $w_{\mathrm{bg}}$:

$$Z = \sum_{j\in\mathcal{N}} w_j + w_{\mathrm{bg}}. \tag{15}$$

The rendered pixel color is then

$$\boldsymbol{C} = \frac{\displaystyle\sum_{i\in\mathcal{N}} \boldsymbol{c}_i\, w_i + \boldsymbol{c}_{\mathrm{bg}}\, w_{\mathrm{bg}}}{Z}, \tag{16}$$

where $\boldsymbol{c}_i$ is the view-dependent color of Gaussian $i$ decoded from its SH coefficients, consistent with Eq. 4.

Let $L$ be the loss function. We derive the gradients with respect to $\boldsymbol{c}_i$, $w_i$, $\alpha_i$, $o_i$, $v_i$, $\sigma$, and $w_{\mathrm{bg}}$ using the chain rule.

**Gradient w.r.t. Color $\boldsymbol{c}_i$.** The partial derivative of the pixel color with respect to Gaussian $i$'s color is its normalized weight:

$$\frac{\partial L}{\partial \boldsymbol{c}_i} = \frac{\partial L}{\partial \boldsymbol{C}} \cdot \frac{w_i}{Z}. \tag{17}$$

**Gradient w.r.t. Effective Weight $w_i$.** Applying the quotient rule to Eq. 16 and summing over color channels $ch$:

$$\frac{\partial L}{\partial w_i} = \frac{1}{Z}\sum_{ch} \frac{\partial L}{\partial \boldsymbol{C}^{ch}} \cdot \big(\boldsymbol{c}_i^{ch} - \boldsymbol{C}^{ch}\big). \tag{18}$$

This formulation ensures that Gaussians whose colors $\boldsymbol{c}_i$ differ significantly from the current pixel estimate $\boldsymbol{C}$ receive larger gradients to adjust their contribution.

**Gradient w.r.t. Projected Alpha $\alpha_i$.** Let $\delta_i = \mathbf{1}[d_i < \sigma]$ be the indicator that Gaussian $i$ is within the depth cutoff. Since $w_i = \alpha_i \max(0, 1 - d_i/\sigma)v_i$, by the chain rule:

$$\frac{\partial L}{\partial \alpha_i} = \frac{\partial L}{\partial w_i} \cdot \delta_i \left(1 - \frac{d_i}{\sigma}\right) v_i. \tag{19}$$

**Gradient w.r.t. Raw Opacity $o_i$.** Since $\alpha_i = o_i \cdot G_i$, and neglecting the boundary case of the $0.99$ clamp:

$$\frac{\partial L}{\partial o_i} = \frac{\partial L}{\partial \alpha_i} \cdot G_i = \frac{\partial L}{\partial w_i} \cdot \delta_i\, G_i \left(1 - \frac{d_i}{\sigma}\right) v_i. \tag{20}$$

**Gradient w.r.t. View-Dependent Scalar $v_i$.**

$$\frac{\partial L}{\partial v_i} = \frac{\partial L}{\partial w_i} \cdot \delta_i \, \alpha_i \left(1 - \frac{d_i}{\sigma}\right). \tag{21}$$

This gradient is backpropagated through the SH basis to update the additional SH coefficients $\{a_k^{(i)}\}$:

$$\frac{\partial L}{\partial a_k^{(i)}} = \frac{\partial L}{\partial v_i} \cdot Y_k(\hat{\boldsymbol{d}}_i), \tag{22}$$

where $Y_k$ are the real spherical harmonic basis functions and $\hat{\boldsymbol{d}}_i$ is the unit view direction toward Gaussian $i$.

**Gradient w.r.t. Scale Parameter $\sigma$.** The gradient is accumulated over all Gaussians within the depth cutoff ($d_i < \sigma$):

$$\frac{\partial L}{\partial \sigma} = \sum_{i \in \mathcal{N}} \frac{\partial L}{\partial w_i} \cdot \delta_i \, \alpha_i \, v_i \cdot \frac{d_i}{\sigma^2}. \tag{23}$$

**Gradient w.r.t. Background Weight $w_{\mathrm{bg}}$.** Analogously to Eq. 18:

$$\frac{\partial L}{\partial w_{\mathrm{bg}}} = \frac{1}{Z} \sum_{ch} \frac{\partial L}{\partial \boldsymbol{C}^{ch}} \cdot \left(\boldsymbol{c}_{\mathrm{bg}}^{ch} - \boldsymbol{C}^{ch}\right). \tag{24}$$

This formulation remains fully end-to-end differentiable, which changes the compositing model from depth-ordered alpha blending in Eq. 4 to an order-independent weighted approximation. We empirically study its impact on reconstruction and semantic rendering quality.

## B Impact of MMA Shape

Tab. 7 reports the rendering performance of two MMA shapes: `mma.m16n8k8` and `mma.m16n8k16`. Compared with `mma.m16n8k8`, `mma.m16n8k16` processes twice the reduction dimension per MMA instruction, thereby reducing the number of K-tiles and flush operations. In our implementation, each flush stage starts with warp-level `ldmatrix` operations and a synchronization barrier. Therefore, `mma.m16n8k16` amortizes fixed pipeline overheads over more useful computation, leading to higher computational efficiency.

Table 7: **Rendering performance on two MMA shapes.**

| MMA Shape | FPS |
|---|---|
| mma.m16n8k8 | 534.0 |
| mma.m16n8k16 | 564.4 |

## C Timing Protocols

We use two timing protocols. The *end-to-end* protocol (Tabs. 3 and 4) times the full per-frame `render()` call. The *kernel-isolated* protocol (Tabs. 5 and 6 and the tables in this appendix) follows the measurement script of FlashGS [11], which is also used by Speedy-Splat [14] and GEMM-GS [26]. The two protocols differ in timing scope. The end-to-end protocol measures the full per-frame `render()` path, including per-frame activations, buffer setup, and rasterizer construction. The kernel-isolated protocol prepares the view-independent state once and times only the CUDA kernels; for the same checkpoint, GPU, and view, this changes the per-frame runtime from 1.32 ms to 0.93 ms. The end-to-end protocol reflects the cost an application experiences in practice, which is why we use it for the main results, while the kernel-isolated protocol allows a fair comparison with acceleration methods under their own convention. Absolute FPS therefore differs between the two protocols (e.g., 156.0 vs. 278.9 FPS for 3DGS on Mip-NeRF 360), while speedups within each table are directly comparable.

## D Hardware Profiling: Precision versus Execution Structure

Since our MMA kernel uses FP16 operands, we check whether its speedup comes from reduced precision alone by running the standard 3DGS blending with FP16 features, converted once outside the timed region.

Table 8: **Nsight Compute counters** per launch ($D$=32, *garden*).

| Counter | Scalar FP32 | Scalar FP16 | Warp-packed MMA |
|---|---|---|---|
| Executed warp instr. | 1341 M | 1575 M | **501 M** (−63%) |
| FMA-pipe instr. | 630 M | 763 M | 171 M |
| ALU-pipe instr. | 217 M | 217 M | 175 M |
| Tensor-Core instr. | 0 | 0 | 4.17 M |
| DRAM bytes | 391 MB | 319 MB | 306 MB |
| L2 bytes | 844 MB | 679 MB | 745 MB |
| L1 bytes | 7.99 GB | 7.99 GB | **0.98 GB** (−88%) |
| DRAM BW utilization | 14.5% | 12.2% | 30.5% |

FP16 storage alone has little influence on 3DGS: on *counter* and *garden* with $D$ from 8 to 128, it changes throughput by $-3\%$ to $+17\%$, whereas our kernel is 2.2–4.8× faster than FP32 3DGS. Tab. 8 shows why: FP16 storage reduces DRAM and L2 traffic, but values are still converted inside the serial blending loop. Even a redesigned `half2` kernel reaches only 1.14×, because the blending weights $w_i = \alpha_i T_i$ depend on $T_{i+1} = T_i(1 - \alpha_i)$ and cannot be vectorized across Gaussians. Our order-independent formulation removes this recurrence, so the warp-packed MMA kernel executes 63% fewer warp instructions and moves 88% less L1 data.

# E Integration with Fast Renderers

Table 9: **Integration into fast renderers.** Kernel-isolated protocol (Appendix C), $D$=8, RTX 4090. Speedups in parentheses are relative to the host renderer.

| | 3D-OVS | | LERF | | Mip-NeRF 360 | |
|---|---|---|---|---|---|---|
| Method | FPS | PSNR | FPS | PSNR | FPS | PSNR |
| FlashGS [11] | 794.7 | 24.71 | 1432.2 | 23.99 | 996.4 | 30.06 |
| FlashGS + MatrixSplat | **2122.4** (2.67×) | 24.81 | **3000.5** (2.09×) | 23.81 | **2302.5** (2.31×) | 30.30 |
| GEMM-GS [26] | 229.2 | 24.63 | 439.8 | 23.97 | 324.1 | 30.10 |
| GEMM-GS + MatrixSplat | **651.0** (2.84×) | 24.79 | **1248.0** (2.84×) | 23.80 | **911.4** (2.81×) | 30.31 |
| Speedy-Splat [14] | 556.7 | 24.65 | 1000.2 | 23.96 | 704.6 | 30.13 |
| Speedy-Splat + MatrixSplat | **1252.0** (2.25×) | 24.79 | **1774.3** (1.77×) | 23.80 | **1360.3** (1.93×) | 30.35 |
| AdR-Gaussian [43] | 489.1 | 24.64 | 952.9 | 23.97 | 654.6 | 30.09 |
| AdR-Gaussian + MatrixSplat | **944.0** (1.93×) | 24.75 | **1607.4** (1.69×) | 23.81 | **1201.4** (1.84×) | 30.31 |

Tab. 9 reports the four pipeline-level accelerators with and without our blending kernel. In each case we replace only the blending kernel and keep the host renderer's culling, scheduling, and sorting. The additional speedup ranges from 1.69× to 2.84×, and PSNR changes by at most 0.24 dB. FlashGS + MatrixSplat reaches 2302.5 FPS on Mip-NeRF 360.

# F Cross-GPU Results

Table 10: **Cross-GPU results.** FPS on Mip-NeRF 360 under the kernel-isolated protocol ($D$=8). Speedups in parentheses are relative to the baseline in the row above.

| Method | RTX 4090 | RTX 6000 Ada | H100 80GB |
|---|---|---|---|
| 3DGS | 278.9 | 252.1 | 252.5 |
| 3DGS + MatrixSplat | **660.0** (2.37×) | **595.4** (2.36×) | **587.5** (2.33×) |
| FlashGS | 996.4 | 932.7 | 769.7 |
| FlashGS + MatrixSplat | **2302.5** (2.31×) | **2190.3** (2.35×) | **1904.7** (2.47×) |

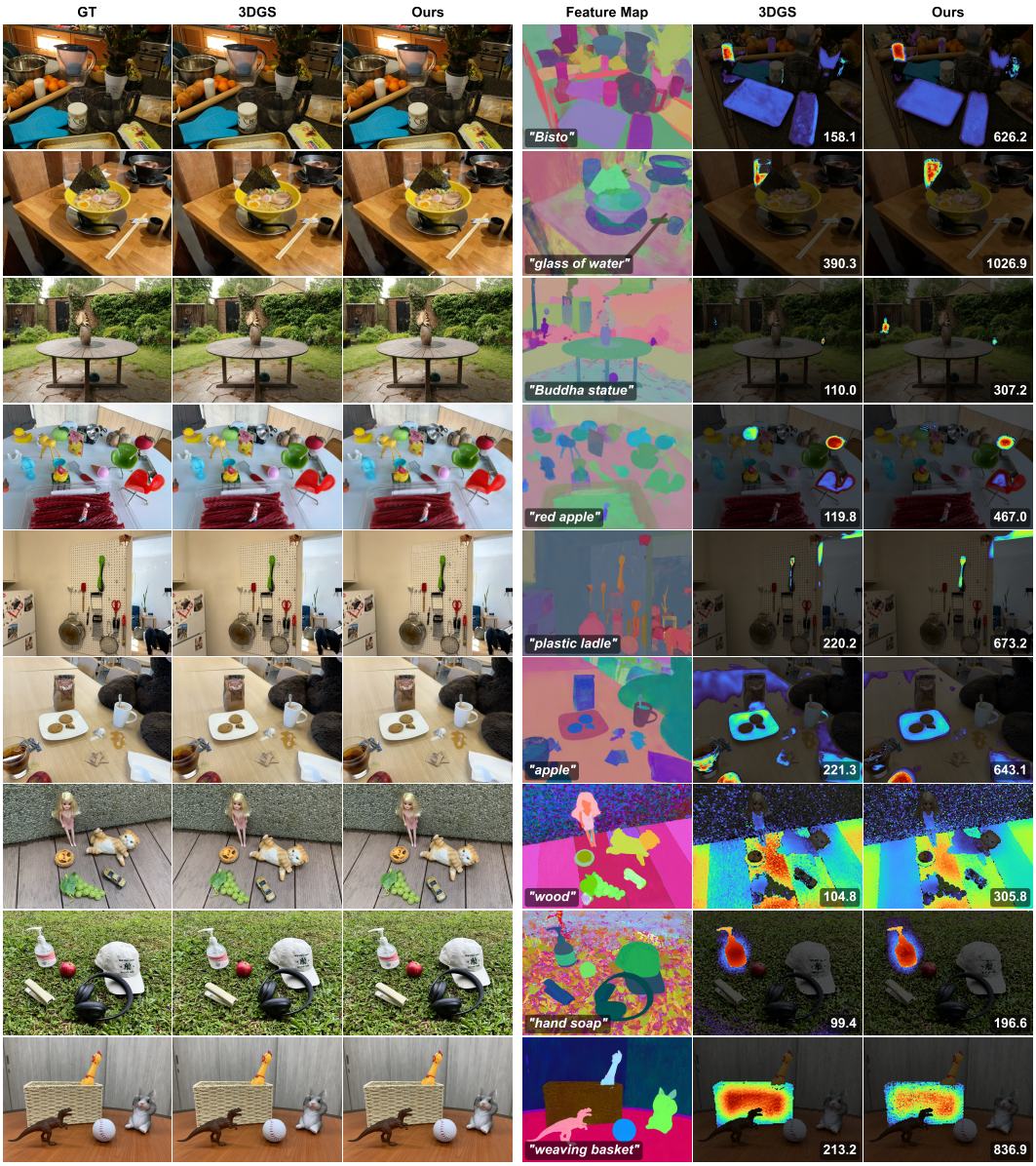


Figure 9: **Visual comparison of quality and language accuracy.** We compare the 3DGS and our method on varying datasets. The number shown at the bottom right is rendering FPS. Following LangSplat [39], the relevancy map is evaluated based on rendered language features.

Tab. 10 reports results on consumer Ada (RTX 4090), workstation Ada (RTX 6000 Ada), and datacenter Hopper (H100) GPUs. The speedups are stable at 2.33–2.37× over 3DGS and 2.31–2.47× over FlashGS. Our implementation relies only on standard warp-level MMA instructions and shared memory, and does not depend on features specific to the RTX 4090.

## G Additional Visual Quality Comparisons

We provide additional visual comparisons in Fig. 9. The number shown at the bottom right is rendering FPS. Following LangSplat [39], the relevancy map is evaluated based on rendered language features.

## NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: We clearly state the scope of our method and support the reported efficiency and quality improvements with quantitative evaluations, ablations, and profiling analysis.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We explain our limitations in the section Conclusion, Limitation, and Broader Impact. We evaluate our method on varying datasets and show our scalability.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [N/A]

Justification: The paper does not include formal theoretical results such as theorems, lemmas, or propositions.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: The paper specifies datasets, evaluation metrics, baselines, and implementation details.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [No]

Justification: The datasets used in our experiments are publicly available, and we provide implementation details and experimental settings in the paper. However, the code is not released at submission time because it is still being cleaned and anonymized. We plan to release the code publicly after publication to facilitate reproducibility.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

Answer: [Yes]

Justification: We specify the datasets, evaluation protocols, metrics, baseline methods, and implementation details used in our experiments. We follow the official training settings of the corresponding 3DGS-based baselines and focus on reporting the rendering-time configurations, including feature dimensions, MMA settings, evaluated scenes. These details are sufficient to understand the experimental setup and reproduce the main results.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [No]

Justification: We do not report error bars or statistical significance tests. The experiments follow the standard evaluation for 3DGS-based rendering methods, where results are typically reported as per-scene metrics and averages over benchmark scenes. Since our main claims are based on consistent improvements across multiple datasets, we provide detailed per-scene quantitative results.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: All main experiments are conducted on a single NVIDIA RTX 4090 GPU, and the cross-GPU study additionally uses an NVIDIA RTX 6000 Ada and an NVIDIA H100 80GB GPU (Appendix F). We provide implementation details, timing protocols, and experimental settings in the paper and appendix.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The research conforms to the NeurIPS Code of Ethics. It does not involve human-subject experiments, new personal data collection, or real-world decision-making deployment. We use publicly available datasets and standard evaluation.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: The paper discusses positive impacts such as enabling efficient feature rendering for interactive scene understanding, robotics, AR/VR, and reducing computational cost. We emphasize responsible deployment with proper consent and privacy protection.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: Our work does not release new datasets, pre-trained generative models, scraped data, or other artifacts with a high risk for misuse. Therefore, dedicated safeguards for releasing high-risk data or models are not applicable.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: We cite the original papers for all existing datasets and code used in our experiments. The datasets are Mip-NeRF 360 (with GAGS segmentation labels), LERF, and 3D-OVS. The code includes 3DGS, LangSplat, TC-GS, and the official implementations of FlashGS, GEMM-GS, Speedy-Splat, AdR-Gaussian, and Duplex-GS; for Sort-Free Gaussian Splatting, which has no official implementation, we use a publicly available third-party implementation. We use these assets following their respective licenses and terms of use.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [N/A]

Justification: The paper does not introduce or release new datasets, pretrained models, or other standalone assets at submission time. We plan to release the code after publication with documentation, usage instructions, and licensing information.

Guidelines:

- The answer [N/A] means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [N/A]

Justification: This paper does not involve crowdsourcing experiments or research with human subjects. We do not collect data from human participants, conduct user studies, or employ crowd workers.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.

- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: This paper does not involve crowdsourcing experiments or research with human subjects. We do not collect data from human participants, conduct user studies, or employ crowd workers.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

    Justification: The core method development in this work does not involve LLMs as an important, original, or non-standard component. Any use of LLMs was limited to writing, editing, or formatting assistance.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.