SAS-LoD: Selective Attribute Sharing for Efficient Level-of-Detail Gaussian Splatting

Anonymous authors
Paper under double-blind review

Abstract

High-fidelity view synthesis using 3D Gaussian Splatting (3DGS) delivers exceptional visual realism with interactive performance. However, rendering large-scale scenes with millions of Gaussian primitives incurs substantial memory usage and computational overhead, often leading to out-of-memory failures and increased rendering latency on resource-constrained devices. Level-of-detail (LoD) techniques alleviate these bottlenecks, yet existing designs face a trade-off: optimizing separate Gaussian sets inflates the resident memory footprint, whereas sharing a single set causes noticeable quality degradation. In this paper, we overcome this trade-off by observing that scale and opacity, accounting for only 4 of 59 parameters, primarily adapt Gaussian footprints across LoD granularities, while all remaining attributes can be shared without quality loss. Building on these insights, we present an efficient 3DGS framework with Selective Attribute Sharing LoD (SAS-LoD). SAS-LoD maintains a single base Gaussian set with compact level-wise residual offsets for scale and opacity, enabling high-fidelity per-level adaptation with minimal resident memory overhead. Coupled with codebook compression and on-demand decoding, SAS-LoD substantially reduces peak memory footprint and improves rendering throughput with high visual fidelity, enabling real-time deployment even on mobile devices.

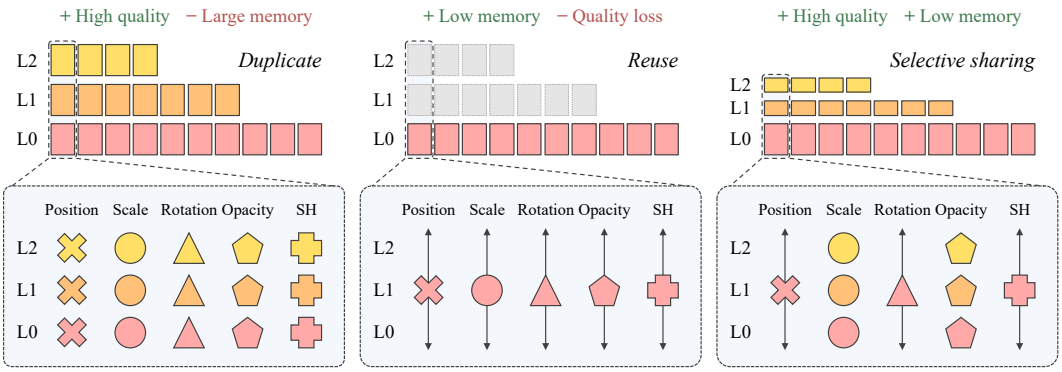
1 Introduction

3D Gaussian Splatting (3DGS) (Kerbl et al., 2023) has emerged as a leading representation for novel view synthesis, and recent work extends it to increasingly expansive domains, such as city-scale environments (Kerbl et al., 2024; Jiang et al., 2025; Lin et al., 2024; Liu et al., 2024). However, scaling to such large scenes inevitably requires a large number of fine-grained Gaussian primitives to cover wide spatial extents, introducing critical rendering bottlenecks in both memory and throughput. Recent 3DGS approaches have addressed this scalability challenge using level-of-detail (LoD) techniques (Kerbl et al., 2024; Ren et al., 2025; Seo et al., 2025; Kulhanek et al., 2025; Milef et al., 2025). By rendering nearby regions at fine granularity and distant regions at coarse granularity, LoD enables processing far fewer Gaussians per view. This reduction directly accelerates rendering and lowers the working memory demands of buffering projected Gaussians during rasterization.

An effective LoD hierarchy is required to reduce the rendered primitive count while maintaining high rendering quality at each level. Optimizing a separate Gaussian set at each LoD level delivers optimal representation for target granularities, yet storing attributes redundantly across levels inevitably inflates the resident memory footprint (Kerbl et al., 2024; Ren et al., 2025; Seo et al., 2025; Kulhanek et al., 2025). Conversely, maintaining a shared Gaussian set and selecting subsets for each hierarchy level avoids memory redundancy, but the retained Gaussians fail to adapt to target granularities, leading to noticeable quality degradation (Milef et al., 2025; Cheng et al., 2026; Guo et al., 2026). Consequently, achieving both per-level adaptability and resident memory efficiency remains a critical challenge for effective LoD design.

To address this trade-off, it is essential to evaluate whether each Gaussian attribute requires per-level adaptation. Because attributes such as position, scale, rotation, opacity, and spherical harmonics (SH) coefficients contribute differently during rendering, their required degree of adjustment across LoD levels varies accordingly. In this study, we thus investigate which attributes demand per-level adaptation and which can be safely shared without sacrificing rendering quality.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.



(a) Independent LoD (b) Shared LoD (c) SAS-LoD (Ours)

Figure 1: **Comparison of LoD designs.** (a) Independent LoD stores full Gaussian attributes per level. (b) Shared LoD reuses a single Gaussian representation across levels. (c) SAS-LoD only models scale and opacity separately at each level, balancing memory efficiency and adaptation capacity.

Remarkably, our empirical analysis reveals that adjusting scale and opacity for each level is critical to maintaining rendering quality, whereas position, rotation, and SH coefficients can be shared across levels with modest quality loss. When certain Gaussians are masked at coarser levels, the remaining primitives complement their lost contributions by adapting their spatial extents and opacities. In practice, adjusting only 4 of the 59 parameters (corresponding to scale and opacity) achieves visual quality comparable to optimizing full attribute sets at every level. This observation highlights that high-fidelity level adaptation is achievable while sharing most attributes across all levels.

Building on these observations, we propose a large-scale 3DGS rendering framework with **S**elective **A**ttribute **S**haring **LoD** (SAS-LoD). SAS-LoD maintains a single base Gaussian set, constructing each coarser level as its nested subset. While position, rotation, and SH coefficients are shared across levels, scale and opacity are adjusted to each level by adding compact level-wise residuals over the base attributes. Consequently, for each active Gaussian at an additional coarse level, we store only 4 residual offsets instead of the full 59 attributes. This selective sharing scheme enables high-fidelity per-level adaptation with minimal resident memory overhead. As a result, SAS-LoD drastically reduces inter-level memory redundancy while preserving LoD's efficiency benefits for both rendering throughput and working memory.

While SAS-LoD reduces working memory and improves rendering throughput with little per-level storage overhead, retaining the full base Gaussian set still requires substantial resident memory for large-scale scenes. This remaining memory cost can limit deployment on memory-constrained devices. To address this bottleneck, SAS-LoD also incorporates a codebook-based compression scheme with on-demand decoding for both the shared attributes and level-wise residuals. During rendering, we first select the Gaussians required for the current viewpoint and LoD levels, then decode only their attributes on the fly. This keeps the full scene compressed in memory while limiting decoding overhead to the selected Gaussians.

Experiments on large-scale benchmarks demonstrate that SAS-LoD achieves rendering quality comparable to per-level independent representations using significantly fewer parameters, while markedly outperforming fully shared representations. Furthermore, we validate that SAS-LoD, coupled with codebook compression and on-demand decoding, substantially reduces peak memory footprint and rendering latency for large-scale scenes while maintaining high visual fidelity.

Our contributions are summarized as follows:

- We empirically demonstrate that scale and opacity, only 4 of 59 parameters, primarily adapt Gaussian footprints across LoD granularities, while all remaining attributes can be shared across levels with modest quality loss.
- We introduce Selective Attribute Sharing LoD (SAS-LoD), an LoD representation that augments a single Gaussian base with compact level-wise residuals for scale and opacity, enabling high-fidelity per-level adaptation with minimal parameter overhead.
- We couple SAS-LoD with codebook compression and on-demand decoding to substantially reduce the total memory footprint (2.7×) and improve rendering speed (1.9×) for large-scale rendering.

2 Related Work

Reconstructing large-scale scenes. Large-scale scene synthesis has been approached along two directions. Divide-and-conquer methods partition the scene into sub-regions and reconstruct each independently. NeRF-based approaches train separate sub-networks per region (Turki et al., 2022; Tancik et al., 2022), and this idea has been extended to 3DGS by optimizing Gaussian primitives within each partition (Lin et al., 2024; Liu et al., 2024; Kerbl et al., 2024). In parallel, multi-GPU training methods parallelize reconstruction by distributing Gaussians or pixel workloads across multiple devices, scaling beyond a single GPU's memory and computational limits (Zhao et al., 2025; Gao et al., 2025). Both directions improve training scalability, but they ultimately reconstruct the scene as a massive set of Gaussians, leaving rendering efficiency unresolved.

Level-of-Detail representations. LoD approaches handle distant regions by selecting a smaller subset of Gaussians for each viewpoint, but differ in how they construct their multi-level hierarchies. Independent LoD approaches optimize a separate Gaussian set at each level through different mechanisms. H3DGS (Kerbl et al., 2024) merges Gaussians bottom-up into a tree, Octree-GS (Ren et al., 2025) anchors them at octree depths, FLoD (Seo et al., 2025) progressively trains levels from coarse to fine, and LODGE (Kulhanek et al., 2025) prunes and fine-tunes fine-level copies for coarser levels. While achieving high visual fidelity, these methods inflate the resident memory footprint due to parameter redundancy across levels. Alternatively, shared representations derive coarser levels as subsets of a single base set, selected via importance ordering (Milef et al., 2025; Guo et al., 2026) or distance-dependent opacity decay (Cheng et al., 2026), eliminating inter-level memory redundancy at the cost of quality loss. Both lines frame per-level adaptation as an all-or-nothing choice, adjusting either every attribute or none. In contrast, we propose selective attribute sharing, adapting only scale and opacity via compact residual offsets to achieve both memory efficiency and visual fidelity.

Compact representations. Recent work has investigated compact Gaussian representations to alleviate storage overhead. Vector quantization (VQ)-based approaches (Lee et al., 2024; Niedermayr et al., 2024; Navaneet et al., 2024; Fan et al., 2024) encode per-Gaussian attributes as codebook indices, enabling lightweight lookup decoding; their compression efficiency is bounded by codebook capacity. Neural encodings achieve higher compression ratios by parameterizing attributes with learned networks, decoding attributes either once over the entire set at load time (Shin et al., 2025; Lee et al., 2025) or per-frame for visible anchors (Lu et al., 2024; Ren et al., 2025). Even when decoding is restricted to the view frustum or the active LoD levels, it relies on non-trivial per-frame network evaluation over the corresponding anchors. Our scheme adopts attribute-wise codebook quantization to significantly lower the resident memory footprint, while combining lightweight lookup decoding with LoD-based selection to decode only a minimal subset of Gaussians on the fly.

3 Preliminaries: 3D Gaussian Splatting

3DGS (Kerbl et al., 2023) represents a volumetric scene as a collection of Gaussian primitives. Each Gaussian primitive $\mathcal{G}_i = \{\mathbf{p}_i, \mathbf{s}_i, \mathbf{r}_i, o_i, \mathbf{h}_i\}$ is parameterized by a position $\mathbf{p}_i \in \mathbb{R}^3$, a scale vector $\mathbf{s}_i \in \mathbb{R}^3$, a quaternion-based rotation vector $\mathbf{r}_i \in \mathbb{R}^4$, an opacity $o_i \in [0, 1]$, and k-order SH coefficients $\mathbf{h}_i \in \mathbb{R}^{3\times(k+1)^2}$, resulting in $D = 11 + 3(k+1)^2$ scalar parameters per Gaussian in total.[1] Building on this definition, each 3D Gaussian is represented in world space as $G(\mathbf{x}) = \exp(-\frac{1}{2}(\mathbf{x}-\mathbf{p})^\top \Sigma^{-1}(\mathbf{x}-\mathbf{p}))$, where Σ denotes the 3D covariance. To render a novel view for a given camera pose, each 3D Gaussian is projected into screen space yielding a per-pixel opacity α_i for each pixel, and its depth is computed. The color $\mathbf{c}_i$ of each primitive is derived from its SH coefficients under the current view direction. Finally, the projected Gaussians are sorted by depth and composited front-to-back using alpha blending

$$C(\mathbf{x}'; \mathcal{G}) = \sum_{i=1}^{M} \mathbf{c}_i \alpha_i \tau_i, \quad \tau_i = \prod_{j=1}^{i-1}(1-\alpha_j), \tag{1}$$

where M is the number of Gaussian primitives overlapping pixel location $\mathbf{x}'$, and τ_i represents the accumulated transmittance of the i-th Gaussian. The blending process terminates early when $\tau_i < \epsilon$, indicating negligible contribution from subsequent primitives.

[1] In our setup, we set $k = 3$, yielding $D = 59$ scalar parameters per Gaussian.

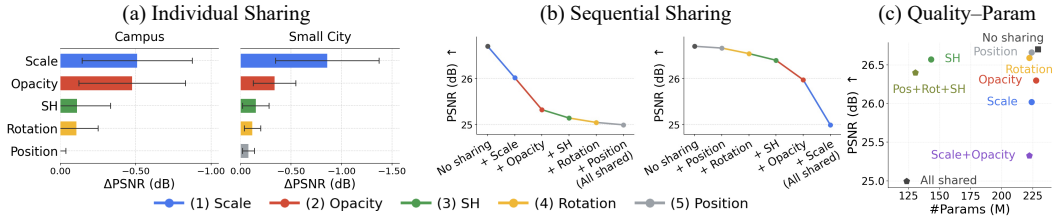


Figure 2: **Impact of attribute sharing across LoD levels.** We evaluate the degradation in rendering quality when Gaussian attributes are shared (a) individually across levels and (b) sequentially across levels, and (c) the trade-off between rendering quality and parameter efficiency. Error bars indicate the standard deviation of per-view PSNR drops within each scene.

4 METHOD

Given a pretrained 3DGS representation of a large-scale scene, our goal is to support high-quality rendering under limited memory and computation. LoD reduces the number of Gaussians processed per view, lowering working memory and rendering latency, but independently storing each level increases resident memory. Our approach separates the attributes that can be shared across levels from those requiring level-specific adaptation. We first establish this distinction empirically (Sec. 4.1) and use it to derive Selective Attribute Sharing LoD (SAS-LoD) (Sec. 4.2). We then construct and optimize its nested hierarchy (Sec. 4.3) and combine the resulting representation with compression and on-demand decoding to reduce the resident memory overhead of the full scene (Sec. 4.4).

4.1 UNDERSTANDING ATTRIBUTE SHARING ACROSS LOD LEVELS

A coarse LoD level uses fewer Gaussians to represent a region. One way to construct such a level is to remove Gaussians from a finer level. Removing Gaussians, however, changes both spatial coverage and alpha compositing: retained primitives must compensate for the contributions of excluded ones. Independently optimizing all attributes at each level provides this flexibility at the cost of duplicated storage, whereas reusing unchanged base attributes restricts adaptation. We therefore ask which attributes must remain level-dependent to preserve rendering quality.

Measuring the effect of sharing. We construct a three-level independent LoD hierarchy from a standard 3DGS reconstruction on the large-scale benchmark (Kerbl et al., 2024), following LODGE (Kulhanek et al., 2025). Each coarser level is initialized from the finest set, pruned, and fine-tuned, preserving a correspondence between its retained Gaussians and the finest-level Gaussians. For each attribute type z, we replace its optimized values at coarser levels with their finest-level counterparts, $z_i^{(l)} \leftarrow z_i^{(0)}$, while keeping all other attributes unchanged. We then render novel views to measure the quality loss caused by removing the level-specific adaptation of that attribute.

Adaptation is concentrated in scale and opacity. Fig. 2(a) shows that sharing position, rotation, or SH coefficients has little effect on rendering quality, whereas sharing scale or opacity causes a noticeable drop. This identifies scale and opacity as the attributes that require level-specific adaptation, while suggesting that position, rotation, and SH coefficients can be shared. The per-view error bars further show that sharing position and rotation has consistently small effects across the evaluated viewpoints. Sharing scale, in contrast, produces both larger average losses and greater variation across views, indicating that its level-specific adaptation is particularly important for some viewpoints. We therefore evaluate sequential sharing (Fig. 2(b)) to verify that these attributes can be shared together. Sharing position, rotation, and SH coefficients together preserves quality, while additionally sharing scale or opacity accounts for most of the degradation of the fully shared representation. The quality-parameter comparison in Fig. 2(c) further shows that retaining level-specific scale and opacity preserves most of the quality of independent levels at a small fraction of their additional parameter cost.

These results are consistent with the roles of scale and opacity in rendering. Scale controls a Gaussian's spatial footprint, while opacity controls its contribution to alpha compositing. Adjusting them allows retained Gaussians to compensate for excluded neighbors without requiring separate positions, rotations, or SH coefficients at every level. Our analysis thus identifies $D_{\text{adapt}} = 4$ parameters, three for scale and one for opacity, as the principal degrees of freedom for level-specific adaptation in the evaluated hierarchies. This motivates allocating additional parameters to these attributes while sharing the remaining $D - D_{\text{adapt}}$ parameters.

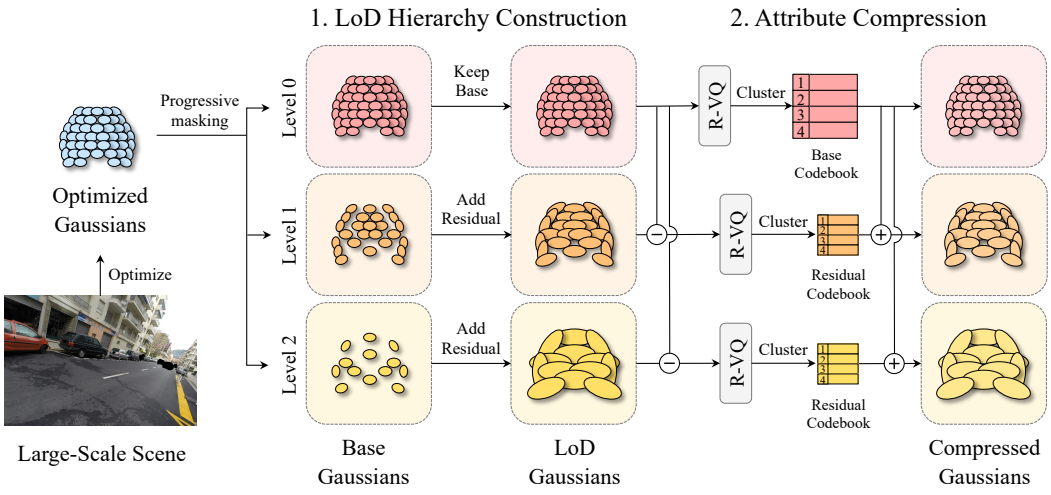


Figure 3: **Representation construction**. Given a set of optimized Gaussians, we construct multi-level hierarchies with SAS-LoD and compress Gaussian attributes at each level. After progressively masking the base Gaussians, we construct LoD Gaussians by adding residuals for each level. The resulting representation enables efficient rendering of large-scale scenes with minimal memory and computational overhead.

4.2 Selective Attribute Sharing LoD

SAS-LoD represents each level through two choices: which base Gaussians are active and how their scale and opacity adapt to that level. Let $\mathcal{G}^{(0)}$ contain N base Gaussians initialized from the input reconstruction. We consider L levels, with $l = 0$ denoting the finest. Let $\mathcal{I}^{(l)}$ be the active index set at level l, defined as:

$$\mathcal{I}^{(L-1)} \subseteq \cdots \subseteq \mathcal{I}^{(1)} \subseteq \mathcal{I}^{(0)} = \{0, \ldots, N-1\}. \tag{2}$$

The nesting applies only to Gaussian indices, while the attributes of an active Gaussian may differ across LoD levels.

Shared attributes and level-specific residuals. For each $i \in \mathcal{I}^{(l)}$, the corresponding Gaussian primitive is represented as:

$$\mathcal{G}_i^{(l)} = \{\mathbf{p}_i^{(0)},\ \mathbf{s}_i^{(0)} + \Delta\mathbf{s}_i^{(l)},\ \mathbf{r}_i^{(0)},\ o_i^{(0)} + \Delta o_i^{(l)},\ \mathbf{h}_i^{(0)}\}, \tag{3}$$

where $\Delta\mathbf{s}_i^{(l)} \in \mathbb{R}^3$ and $\Delta o_i^{(l)} \in \mathbb{R}$ are level-specific residuals, with $\Delta\mathbf{s}_i^{(0)} = \mathbf{0}$ and $\Delta o_i^{(0)} = 0$. Each level is defined as $\mathcal{G}^{(l)} = \{\mathcal{G}_i^{(l)} \mid i \in \mathcal{I}^{(l)}\}$, and the resulting hierarchy is $\{\mathcal{G}^{(l)}\}_{l=0}^{L-1}$. An active Gaussian therefore retains its base position, rotation, and SH coefficients, while its scale and opacity can adapt to the level's granularity, as illustrated in Fig. 1.

Parameter cost. Let $N_l = |\mathcal{I}^{(l)}|$ denote the number of active Gaussians at level l. For the same active Gaussian counts, the numbers of scalar attribute parameters before quantization are

$$\Theta_{\text{SAS}} = DN + D_{\text{adapt}} \sum_{l=1}^{L-1} N_l, \qquad \Theta_{\text{indep}} = DN + D \sum_{l=1}^{L-1} N_l, \qquad D_{\text{adapt}} \ll D. \tag{4}$$

These counts exclude indexing overhead. SAS-LoD retains only D_{adapt}/D of the additional attribute parameters required by independent levels ($4/59$ in our setting), while a fully shared representation stores only DN parameters but provides no level-specific attribute adaptation. Thus, SAS-LoD keeps the parameter overhead of high-quality LoD hierarchies small, while the base scene still accounts for DN parameters. We address that memory cost through compression in Sec. 4.4.

Residuals also provide a useful separation for compression. The base stores the common attribute values, while the additional levels store only their deviations. Before quantization, a residual and a separate level-specific value offer the same degrees of freedom with the base fixed. The benefit of the residual parameterization is that these deviations can be quantized separately from the base attributes, and their lower variance enables more effective quantization.

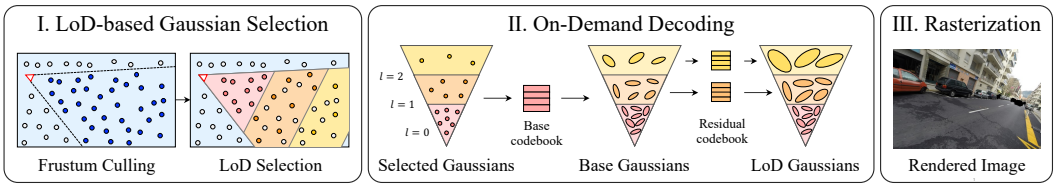


Figure 4: **Rendering Pipeline**. The first stage identifies a minimal set of Gaussians relevant to the current viewpoint, reducing workload for subsequent stages. Next, the selected Gaussians are decoded from their compact attribute format into full representations on the fly. Finally, the decoded Gaussians are rasterized to synthesize high-quality views efficiently, enabling fast and memory-conscious rendering of large-scale scenes.

4.3 Constructing and Optimizing SAS-LoD

We construct levels from fine to coarse, alternating Gaussian masking with fine-tuning. Each level must both use fewer primitives and represent the scene at its target granularity. We use depth-dependent smoothing to specify that granularity, importance-based masking to determine the active set, and optimization of the shared base and residuals to recover quality after masking.

Setting the target granularity. Each level has a predefined viewing-depth threshold $d^{(l)}$, with larger thresholds for coarser levels. Following LODGE (Kulhanek et al., 2025), we apply the depth-dependent 3D smoothing filter of Mip-Splatting (Yu et al., 2024) to impose a minimum spatial footprint appropriate to that threshold. The filter controls the granularity at which a level is evaluated, while the scale and opacity residuals provide learnable adaptation within that level. Filtering is evaluated during rendering without overwriting the stored attributes.

Selecting nested active sets. We initialize the finest level with all base Gaussians active. At each coarser level, we mask low-importance Gaussians using the importance score $\Psi(\cdot)$ from RadSplat (Niemeyer et al., 2025):

$$m_i^{(l)} = \mathbb{1}\Big[\Psi(\mathcal{G}_i^{(l)}) \geq \epsilon_{\text{mask}}\Big]\, m_i^{(l-1)}, \qquad m_i^{(0)} = 1, \tag{5}$$

where ϵ_{mask} is the masking threshold and $\mathcal{I}^{(l)} = \{i \mid m_i^{(l)} = 1\}$. By restricting selection to the preceding active set, this recursive masking ensures the nested structure defined in Eq. (2). Thus, once a Gaussian is excluded at a given level, it remains unavailable at all coarser levels.

Optimizing shared and level-specific parameters. We initialize newly introduced residuals to zero. After masking, we discard the residuals of inactive Gaussians at that level and jointly fine-tune the remaining residuals and shared base attributes using the 3DGS reconstruction objective (Kerbl et al., 2023), without densification. The residuals allow the retained Gaussians to adjust their spatial extent and opacity to compensate for the removed contributions. The base attributes are also optimized rather than frozen, so the learned representation includes both shared updates and level-specific corrections. We repeat masking and fine-tuning to construct the hierarchy. More details on the construction procedure are provided in the appendix.

Contiguous layout without per-level masks. Nesting also reduces the index overhead required at inference. Each Gaussian has a maximum active level $l_i^{\max} = \max\{l \mid i \in \mathcal{I}^{(l)}\}$, which indicates its coarsest active level. Sorting base Gaussians by decreasing $l_i^{\max}$ makes every active set a contiguous prefix of length N_l. We can therefore identify each level's candidates by a range boundary rather than an explicit mask. Level-specific residuals are stored only for active Gaussians, in the corresponding base order. This preserves access to shared attributes while avoiding separate Gaussian copies or per-level masks.

4.4 Compressed Representation and On-Demand Decoding

Even with a small LoD overhead, the base Gaussian set can occupy substantial resident memory in a large scene. We therefore compress SAS-LoD and keep it compressed during rendering. The representation must support selective access: decoding the full scene in advance would restore the memory cost that compression is intended to avoid. To this end, we use codebooks to recover the attributes of selected Gaussians through a small number of lookups.

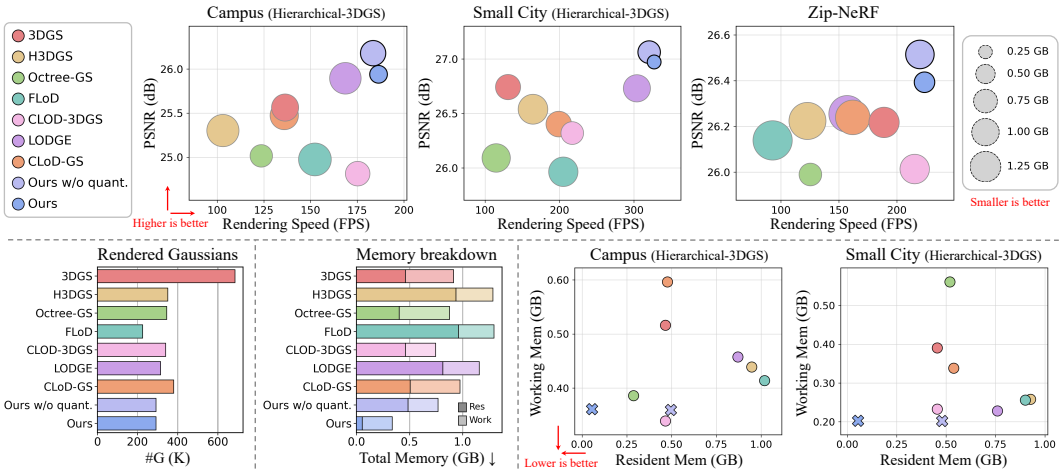


Figure 5: **Quantitative results.** We compare our method with baseline approaches on Hierarchical-3DGS (Kerbl et al., 2024) and Zip-NeRF (Barron et al., 2023). We report results for each scene on Hierarchical-3DGS and average scores on Zip-NeRF. Circle size denotes memory usage.

Shared base and residual compression. We quantize base scale, rotation, opacity, and SH coefficients using separate attribute codebooks, while retaining positions in full precision to preserve rendering accuracy. The scale-opacity residuals of all coarser levels use a shared residual codebook representation. Thus, base values are encoded once, and each additional level stores only codebook indices for its active residuals. Our ablation study (Sec. 5.3) compares residual quantization with directly quantizing level-specific attributes and shows the compression advantage of encoding deviations from the shared base.

To compress the Gaussian attributes, we use VQ and residual vector quantization (R-VQ) (Zeghidour et al., 2021). R-VQ approximates an attribute vector as a sum of entries from multiple small codebooks, allowing reconstruction through a sequence of lookups and additions. Note that these quantization residuals introduced by R-VQ are distinct from the level-wise scale-opacity residuals in SAS-LoD. Attribute-specific configurations are provided in the appendix. In our experiments, we achieve 8.5× resident memory compression while preserving rendering quality and throughput.

Selective rendering with on-demand decoding. As shown in Fig. 4, we first apply frustum culling and depth-based LoD selection. After culling, the Gaussian primitives are selected from each level based on their camera-centric depth d_i as follows:

$$\mathcal{G}_{\text{sel}} = \bigcup_{l=0}^{L-1} \left\{ \mathcal{G}_i^{(l)} \;\middle|\; i \in \mathcal{I}^{(l)}, \quad d^{(l)} \le d_i < d^{(l+1)} \right\}, \tag{6}$$

where $d^{(l)}$ is the predefined depth threshold for level l, with $d^{(0)} = 0$ and $d^{(L)} = \infty$. Because positions are shared, a Gaussian's depth d_i is invariant to its LoD level, allowing the same depth value to be used across all levels. The nested ranges identify candidates at each level, and the depth intervals select the appropriate granularity for the current view.

Once the LoD level is determined, only the selected Gaussians are decoded on demand. For a coarse-level Gaussian, we retrieve its base attributes and level-specific residual, combine them as in Eq. (3), and rasterize the resulting primitive. The full scene remains compressed, while attribute reconstruction is restricted to the active subset. SAS-LoD thereby limits the number of primitives processed per frame with little additional parameter overhead, and compression with selective on-demand decoding reduces resident memory without requiring a fully decoded scene.

5 Experiments

5.1 Experimental Setup

Benchmarks. We evaluate our framework on two representative large-scale novel view synthesis benchmarks: Hierarchical-3DGS (Kerbl et al., 2024) and Zip-NeRF (Barron et al., 2023). Specifically, we use two outdoor scenes from Hierarchical-3DGS, and four indoor scenes from Zip-NeRF. For Hierarchical-3DGS, we use a single evaluation chunk for each scene rather than the full dataset.



Figure 6: **Qualitative Results**. We compare our method with baseline methods on the outdoor 'Campus' and 'Small City' scenes from Hierarchical-3DGS (Kerbl et al., 2024). Peak memory usage, rendering speed, and quality are shown in the corners of each figure.

Baselines. We compare our methods with standard 3DGS (Kerbl et al., 2023), independent LoD representations including H3DGS (Kerbl et al., 2024), Octree-GS (Ren et al., 2025), FLoD (Seo et al., 2025), and LODGE (Kulhanek et al., 2025), and shared LoD representations including CLOD-3DGS (Milef et al., 2025) and CLoD-GS (Cheng et al., 2026).

Evaluation protocol. We report rendering speed (FPS), peak memory usage (Mem), and the number of rendered Gaussians (#G) to evaluate runtime efficiency. In addition, we report novel-view image quality using the PSNR, SSIM, and LPIPS (Zhang et al., 2018).

Implementation details. We optimize the base 3DGS for 30K iterations following the original scheme (Kerbl et al., 2023). For large-scale scenes, we adopt the pruning strategy of RadSplat (Niemeyer et al., 2025) and the densification criterion of H3DGS (Kerbl et al., 2024). We use three LoD levels. All experiments are conducted on an NVIDIA RTX 3090 GPU (24GB). Additional implementation details are provided in the appendix.

5.2 Results

Quantitative evaluation. Fig. 5 demonstrates quantitative comparisons on the Hierarchical-3DGS (Kerbl et al., 2024) and Zip-NeRF (Barron et al., 2023) datasets. The comparison results indicate that our approach shows superior rendering efficiency compared to any baselines, achieving both minimal runtime memory footprint and high rendering throughput. In addition to top-performing efficiency, our method also provides remarkable visual quality comparable to independent LoD representations. Scene-wise detailed results are provided in the appendix.

Fig. 5 further presents a detailed analysis of rendering efficiency in terms of resident memory, working memory, and rendering throughput. Standard 3DGS incurs high memory usage since all Gaussians are stored in full representation, whereas LoD representations achieve improved rendering efficiency with reduced per-view workload. Despite efficient sampling, independent representations, such as H3DGS, FLoD, and LODGE, suffer from excessive resident memory consumption, primarily due to the prohibitive number of Gaussians required to cover the full LoD hierarchy. In contrast, the LoD structure of Octree-GS achieves low resident memory overhead through anchor-based encoding, though its neural decoding increases working memory and slows rendering. While maintaining low resident memory, CLOD-3DGS and CLoD-GS reduce the number of rendered primitives, thereby alleviating per-view burden, but their shared structure leads to quality degradation.

In contrast, SAS-LoD requires minimal resident memory, while LoD-based selective rendering and on-demand decoding lower working memory and rendering latency. Importantly, this performance is attained without any runtime offloading, further enhancing its practicality for real-time deployment.

Qualitative evaluation. As shown in Fig. 6, we also conduct a qualitative comparison, highlighting geometric inaccuracies and visual artifacts. H3DGS, FLoD, and Octree-GS introduce jagged or blurry effects in nearby regions due to insufficient granularity. CLOD-3DGS, meanwhile, exhibits incomplete geometry due to gaps caused by removed primitives. Despite its minimal memory usage,

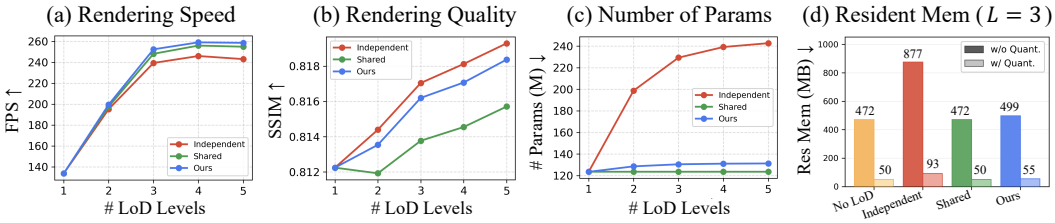


Figure 7: **Ablation study on LoD design.** We evaluate rendering quality and efficiency on Hierarchical-3DGS (Kerbl et al., 2024) using different LoD designs.

our method achieves high rendering quality and throughput comparable to LODGE, while recovering fine-grained geometry over a large spatial area without introducing noticeable visual artifacts.

5.3 Ablation Study

Overall pipeline. We conduct an ablation study to validate each component of our framework. As shown in Table 1, our SAS-LoD hierarchy significantly reduces per-view workload, lowering working memory by 1.6× and improving rendering speed by 1.9×, with only a slight increase in resident memory. Our compression method, in turn, reduces resident memory by 9.3× through a codebook-based representation, while on-demand decoding of the minimal Gaussian subsets selected by our LoD-based mechanism sustain rendering speed. Combining both techniques, our framework achieves 1.9× faster rendering with 2.7× lower overall memory consumption, while maintaining comparable quality.

Table 1: **Ablation study on overall pipeline.** We analyze rendering efficiency and quality on Hierarchical-3DGS.

Method	#G ↓	FPS ↑	Memory (MB)			Quality	
			Res ↓	Work ↓	Total ↓	PSNR ↑	SSIM ↑
3DGS	685K	134 (1.0×)	472 (1.0×)	464 (1.0×)	936 (1.0×)	26.15	0.812
+ SAS-LoD	292K	252 (1.9×)	499 (0.9×)	288 (1.6×)	787 (1.2×)	26.62	0.816
+ Compression	685K	133 (1.0×)	50 (9.3×)	461 (1.0×)	511 (1.8×)	26.06	0.810
+ Both (Ours)	292K	257 (1.9×)	55 (8.5×)	289 (1.6×)	344 (2.7×)	26.46	0.814

LoD design. As shown in Fig. 7, we evaluate rendering quality and parameter efficiency across different LoD designs. (a) Higher LoD levels improve rendering throughput. (b) While shared representations suffer from lower visual fidelity, our SAS-LoD achieves quality comparable to independent representations. (c) Independent representations incur substantial parameter growth as the number of LoD levels increases, whereas both shared representations and SAS-LoD exhibit minimal parameter overhead. (d) For $L = 3$, this parameter expansion leads to 1.8× higher resident memory than SAS-LoD, and still 1.7× after quantization. Overall, SAS-LoD effectively balances rendering quality and parameter efficiency, combining the strengths of existing LoD designs.

Residual parameterization. We compare two parameterizations for level-wise attributes: (a) direct level-wise values and (b) residuals from the shared base. Under the same quantization scheme, Table 2 shows that our residual parameterization yields a more compact distribution with lower variance, resulting in higher rendering quality at the same resident memory. The slight FPS drop with the residual parameterization arises from progressive decoding of base and residuals.

Table 2: **Ablation study on residual parameterization.** $\sigma(\cdot)$ measures the standard deviation of level-wise parameters. Best scores are underlined.

Parameterization	$\sigma(\mathbf{s}^{(l)})$ / $\sigma(o^{(l)})$ ↓	Res Mem	PSNR ↑	FPS ↑
Baseline (w/o quantization)	-	499 MB	26.62	252
(a) Direct $(\mathbf{s}_i^{(l)}, o_i^{(l)})$	1.35 / 2.28	55.5 MB	26.24	<u>262</u>
(b) Residual $(\Delta\mathbf{s}_i^{(l)}, \Delta o_i^{(l)})$	0.33 / 1.29	55.5 MB	<u>26.46</u>	257

6 Conclusion

We introduced SAS-LoD, an LoD framework that enables large-scale scene rendering on low-end devices with limited memory and computational capacity. Rather than partially addressing runtime overhead, we tackled three bottlenecks, such as resident memory, working memory, and rendering throughput, through an efficient LoD representation. Overall, SAS-LoD achieves outstanding runtime efficiency in both memory usage and rendering speed, while maintaining high visual fidelity.

Limitations. Our framework does not reuse previously decoded attributes across frames. Incorporating a fine-grained caching strategy could further reduce per-view workload by leveraging temporal redundancy. Future work may explore more effective LoD designs that jointly consider rendering and compression efficiency.

AI USE STATEMENT

We did not use generative AI tools for any research tasks requiring disclosure, including developing the method, designing or conducting experiments, implementing the method, formulating claims, or interpreting results. All research ideas, implementations, and analyses were developed and conducted by the authors. Generative AI tools were used to improve the manuscript's grammar and readability and to proofread the manuscript for inconsistencies. All AI-assisted text was reviewed and revised by the authors. We take responsibility for the final content of this work.

REFERENCES

Jonathan T Barron, Ben Mildenhall, Dor Verbin, Pratul P Srinivasan, and Peter Hedman. Zip-nerf: Anti-aliased grid-based neural radiance fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 19697–19705. IEEE, 2023.

Zhigang Cheng, Mingchao Sun, Liu Yu, Luyang Tang, Mu Xu, Yangyan Li, Peng Pan, et al. Clod-gs: Continuous level-of-detail via 3d gaussian splatting. In *International Conference on Learning Representations*, volume 2026, pp. 89275–89299, 2026.

Zhiwen Fan, Kevin Wang, Kairun Wen, Zehao Zhu, Dejia Xu, and Zhangyang Wang. Lightgaussian: unbounded 3d gaussian compression with 15x reduction and 200+ fps. In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, pp. 140138–140158, 2024.

Yuanyuan Gao, Hao Li, Jiaqi Chen, Zhengyu Zou, Zhihang Zhong, Dingwen Zhang, Xiao Sun, and Junwei Han. Citygs-$\mathcal{X}$: A scalable architecture for efficient and geometrically accurate large-scale scene reconstruction. In *2025 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 27187–27196. IEEE, 2025.

Zhilin Guo, Boqiao Zhang, Hakan Aktas, Kyle Fogarty, Jeffrey Hu, Nursena Koprucu Aslan, Wenzhao Li, Canberk Baykal, Albert Miao, Josef Bengtson, et al. Matryoshka gaussian splatting. *arXiv preprint arXiv:2603.19234*, 2026.

Lihan Jiang, Kerui Ren, Mulin Yu, Linning Xu, Junting Dong, Tao Lu, Feng Zhao, Dahua Lin, and Bo Dai. Horizon-gs: Unified 3d gaussian splatting for large-scale aerial-to-ground scenes. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 26789–26799, 2025.

Mark Kellogg. 3d gaussian splatting for three.js. https://github.com/mkkellogg/GaussianSplats3D, 2025. GitHub repository.

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

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

Jonas Kulhanek, Marie-Julie Rakotosaona, Fabian Manhardt, Christina Tsalicoglou, Michael Niemeyer, Torsten Sattler, Songyou Peng, and Federico Tombari. Lodge: Level-of-detail large-scale gaussian splatting with efficient rendering. *arXiv preprint arXiv:2505.23158*, 2025.

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*, pp. 21719–21728, 2024.

Joo Chan Lee, Jong Hwan Ko, and Eunbyung Park. Optimized minimal 3d gaussian splatting. *arXiv preprint arXiv:2503.16924*, 2025.

Jiaqi Lin, Zhihao Li, Xiao Tang, Jianzhuang Liu, Shiyong Liu, Jiayue Liu, Yangdi Lu, Xiaofei Wu, Songcen Xu, Youliang Yan, et al. Vastgaussian: Vast 3d gaussians for large scene reconstruction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 5166–5175, 2024.

Yang Liu, Chuanchen Luo, Lue Fan, Naiyan Wang, Junran Peng, and Zhaoxiang Zhang. Citygaussian: Real-time high-quality large-scale scene rendering with gaussians. In *European Conference on Computer Vision*, pp. 265–282. Springer, 2024.

Tao Lu, Mulin Yu, Linning Xu, Yuanbo Xiangli, Limin Wang, Dahua Lin, and Bo Dai. Scaffold-gs: Structured 3d gaussians for view-adaptive rendering. *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024.

Nicholas Milef, Dario Seyb, Todd Keeler, Thu Nguyen-Phuoc, A Božič, Sushant Kondguli, and Carl Marshall. Learning fast 3d gaussian splatting rendering using continuous level of detail. *Computer Graphics Forum*, pp. e70069, 2025.

KL Navaneet, Kossar Pourahmadi Meibodi, Soroush Abbasi Koohpayegani, and Hamed Pirsiavash. Compgs: Smaller and faster gaussian splatting with vector quantization. In *European Conference on Computer Vision*, pp. 330–349. Springer, 2024.

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

Michael Niemeyer, Fabian Manhardt, Marie-Julie Rakotosaona, Michael Oechsle, Daniel Duckworth, Rama Gosula, Keisuke Tateno, John Bates, Dominik Kaeser, and Federico Tombari. Radsplat: Radiance field-informed gaussian splatting for robust real-time rendering with 900+ fps. In *2025 International Conference on 3D Vision (3DV)*, pp. 134–144. IEEE, 2025.

Kerui Ren, Lihan Jiang, Tao Lu, Mulin Yu, Linning Xu, Zhangkai Ni, and Bo Dai. Octree-gs: Towards consistent real-time rendering with lod-structured 3d gaussians. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2025.

Johannes L Schonberger and Jan-Michael Frahm. Structure-from-motion revisited. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4104–4113, 2016.

Yunji Seo, Young Sun Choi, HyunSeung Son, and Youngjung Uh. Flod: Integrating flexible level of detail into 3d gaussian splatting for customizable rendering. *ACM Transactions on Graphics (TOG)*, 44(4):1–16, 2025.

Seungjoo Shin, Jaesik Park, and Sunghyun Cho. Locality-aware gaussian compression for fast and high-quality rendering. *arXiv preprint arXiv:2501.05757*, 2025.

Matthew Tancik, Vincent Casser, Xinchen Yan, Sabeek Pradhan, Ben Mildenhall, Pratul P Srinivasan, Jonathan T Barron, and Henrik Kretzschmar. Block-nerf: Scalable large scene neural view synthesis. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 8248–8258, 2022.

Haithem Turki, Deva Ramanan, and Mahadev Satyanarayanan. Mega-nerf: Scalable construction of large-scale nerfs for virtual fly-throughs. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 12922–12931, 2022.

Zehao Yu, Anpei Chen, Binbin Huang, Torsten Sattler, and Andreas Geiger. Mip-splatting: Alias-free 3d gaussian splatting. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 19447–19456, 2024.

Neil Zeghidour, Alejandro Luebs, Ahmed Omran, Jan Skoglund, and Marco Tagliasacchi. Soundstream: An end-to-end neural audio codec. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 30:495–507, 2021.

Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 586–595, 2018.

Hexu Zhao, Haoyang Weng, Daohan Lu, Ang Li, Jinyang Li, Aurojit Panda, and Saining Xie. On scaling up 3d gaussian splatting training. In *International Conference on Learning Representations*, volume 2025, pp. 35642–35661, 2025.

A Supplementary Material

We provide a supplementary video demonstrating real-time rendering results against baseline approaches, along with experiments on mobile devices.

B Implementation Detail

B.1 Representation Construction

In this section, we provide additional implementation details for constructing our representation. Our method consists of three optimization stages: base representation, level-of-detail (LoD) representation, and compact representation. Algorithm 1 outlines the overall representation construction process. Given a set of images and corresponding camera parameters, we first optimize Gaussian primitives as the base representation, using the original hyperparameter settings of 3D Gaussian Splatting (3DGS) (Kerbl et al., 2023). Building upon this, we construct a LoD hierarchy and compress Gaussian attributes at each LoD level. In the following, we describe the detailed experimental settings for these stages.

Base 3DGS representation. We optimize 3D Gaussians for 30K iterations, following the hyperparameter settings of the original scheme (Kerbl et al., 2023). To achieve robust reconstruction in large-scale scenes, we modify the densification metric and adopt a pruning strategy. Specifically, we employ the densification metric of H3DGS (Kerbl et al., 2024), which evaluates the maximum gradient over several iterations rather than the average gradient. In addition, the metric incorporates the maximum radius in screen space and the opacity value to identify target Gaussians for densification. For pruning, we employ the importance score $\Psi(\cdot)$ of RadSplat (Niemeyer et al., 2025):

$$m_i = \mathbb{1}\Big(\Psi(\mathcal{G}_i) \geq \epsilon_{\text{prune}}\Big), \tag{7}$$

where $m_i \in \{0, 1\}$ is a binary mask with 0 indicating to be pruned. We perform pruning two times for 16K iteration and 24K iteration with a threshold of $\epsilon_{\text{prune}} = 0.02$.

Level-of-Detail representation. We set the number of LoD levels to three, with depth thresholds $(d^{(1)}, d^{(2)}) = (10, 28)$ for the Hierarchical-3DGS dataset (Kerbl et al., 2024) and $(d^{(1)}, d^{(2)}) = (2, 4)$ for the Zip-NeRF dataset (Barron et al., 2023). These thresholds are determined to minimize the number of Gaussian-tile intersections, which directly impacts rendering overhead, following LODGE (Kulhanek et al., 2025). For LoD construction, we apply 3D smoothing filters across all levels according to the depth threshold of each level. Masking is then progressively performed from lower to higher levels, except for the lowest one, three times per level with thresholds $(\epsilon^0_{\text{mask}}, \epsilon^1_{\text{mask}}, \epsilon^2_{\text{mask}}) = (0.004, 0.012, 0.02)$. After each pruning step, we conduct 1K iterations of fine-tuning, resulting in a total of 6K iterations.

Compressed representation. For each LoD level, we define codebook representations and cluster each attribute into them using vector quantization (VQ). Specifically, we adopt residual vector quantization (R-VQ) for multi-dimensional attributes, including scale, rotation, SH coefficients, and scale-opacity residuals, while applying standard VQ to opacity, which is a scalar. Table 3 summarizes the vector quantization configuration of our compact representation. The codebooks are initialized using K-means clustering, and we further finetune the codebook entries for 1K iterations with fixed assignments.

Table 3: **Quantization configuration and resident memory.** We report the codebook configuration for each attribute and the resulting resident memory usage.

Attributes	Components			Size		Resident memory					
	Dim	#Stages	#Entries	Codebook	Per-Gaussian	Campus	Small City	Alameda	Berlin	London	NYC
Position	3	–	–	–	12.0 B	24.3 MB	23.6 MB	30.1 MB	34.4 MB	24.2 MB	27.4 MB
Scale	3	3	256	9.0 KB	3.0 B	6.1 MB	5.9 MB	7.5 MB	8.6 MB	6.1 MB	6.9 MB
Rotation	4	3	256	12.0 KB	3.0 B	6.1 MB	5.9 MB	7.5 MB	8.6 MB	6.1 MB	6.9 MB
Opacity	1	1	256	1.0 KB	1.0 B	2.0 MB	2.0 MB	2.5 MB	2.9 MB	2.0 MB	2.3 MB
SH DC	3	3	256	9.0 KB	3.0 B	6.1 MB	5.9 MB	7.5 MB	8.6 MB	6.1 MB	6.9 MB
SH rest	45	3	256	138.0 KB	3.0 B	6.2 MB	6.0 MB	7.7 MB	8.7 MB	6.2 MB	7.0 MB
Residuals	4	3	256	12.0 KB	3.0 B	5.7 MB	4.4 MB	7.3 MB	7.4 MB	6.1 MB	4.1 MB

Algorithm 1 Overall pipeline of representation construction.

Input: A set of optimized Gaussians $\mathcal{G}$
Output: A set of base Gaussians $\mathcal{G}^{(0)}$ and level-wise residuals $\{(\Delta\mathbf{s}^{(l)}, \Delta o^{(l)})\}_{l=1}^{L-1}$
$\mathcal{G}^{(0)} \leftarrow \mathcal{G}$ ▷ Initialize base Gaussians
$\mathcal{I}^{(0)} \leftarrow \{0, 1, \ldots, N-1\}$ ▷ Initialize active indices
for $l \leftarrow 1$ to $L-1$ **do**
  $\mathcal{G}^{(0)}, (\Delta\mathbf{s}^{(l)}, \Delta o^{(l)}) \leftarrow \text{ConstructLoD}(\mathcal{G}^{(0)}, d^{(l)}, \mathcal{I}^{(l-1)})$ ▷ Introduce residuals
  for $r \leftarrow 0$ to $R-1$ **do**
    $\mathcal{I}^{(l)} \leftarrow \text{Mask}(\mathcal{G}^{(0)}, (\Delta\mathbf{s}^{(l)}, \Delta o^{(l)}), \epsilon^{r}_{\text{mask}})$ ▷ Mask low-importance Gaussians
    $\mathcal{G}^{(0)}, (\Delta\mathbf{s}^{(l)}, \Delta o^{(l)}) \leftarrow \text{Finetune}(\mathcal{G}^{(0)}, (\Delta\mathbf{s}^{(l)}, \Delta o^{(l)}))$
  end for
end for
$\hat{\mathcal{G}}^{(0)}, \{(\Delta\hat{\mathbf{s}}^{(l)}, \Delta\hat{o}^{(l)})\}_{l=1}^{L-1} \leftarrow \text{Compress}(\mathcal{G}^{(0)}, \{(\Delta\mathbf{s}^{(l)}, \Delta o^{(l)})\}_{l=1}^{L-1})$ ▷ Compress attributes
$\hat{\mathcal{G}}^{(0)}, \{(\Delta\hat{\mathbf{s}}^{(l)}, \Delta\hat{o}^{(l)})\}_{l=1}^{L-1} \leftarrow \text{Finetune}(\hat{\mathcal{G}}^{(0)}, \{(\Delta\hat{\mathbf{s}}^{(l)}, \Delta\hat{o}^{(l)})\}_{l=1}^{L-1})$

Table 4: **Evaluation resolutions.** Each scene is evaluated at a different rendering resolution.

Hierarchical-3DGS		Zip-NeRF			
Campus	Small City	Alameda	Berlin	London	NYC
1436×1075	1024×690	1394×793	2000×1140	1392×793	1393×793

B.2 Evaluation Protocol

We reproduce most baselines, except LODGE (Kulhanek et al., 2025), using the released official code and configurations, and evaluate rendering speed under the same experimental setting to ensure fair comparison. For LODGE, whose official code is not publicly available, we re-implement the method following the released configurations. Each test image is rendered ten times, and the median rendering time is reported. For datasets, we use the single evaluation chunk of the Hierarchical-3DGS dataset (Kerbl et al., 2024) at its original resolution, while the Zip-NeRF dataset (Barron et al., 2023) is downsampled to half of its original resolution. Table 4 shows the rendering resolution used for each scene. Large-scale scenes can be processed in spatial chunks to accommodate their scale. We therefore evaluate each spatial chunk independently, focusing on rendering efficiency at the chunk level, which is the basic processing unit for large-scale scenes.

B.3 Baseline Methods

All methods are evaluated under the same rendering setup without out-of-core streaming.

H3DGS H3DGS (Kerbl et al., 2024) is a post-training LoD framework that constructs an independent LoD representation. As the input Gaussian set for H3DGS, we employ our base 3DGS representation trained with several techniques tailored for large-scale scene reconstruction for a fair comparison. Given a set of input Gaussians, it constructs a tree-structured LoD hierarchy and fine-tunes it for 15K iterations. During rendering, the entire LoD hierarchy is loaded into memory, and Gaussians relevant to the current viewpoint are selected and rendered at its default granularity ($\tau = 6$). In the experiment, we measure resident memory usage over all attributes in the LoD hierarchy and include both Gaussian selection and rasterization in the measured rendering time.

Octree-GS Octree-GS (Ren et al., 2025) is a from-scratch LoD framework that constructs an independent LoD representation for anchor-based Gaussians. We follow the official configuration of the released code, which determines hyperparameters, such as the number of LoD levels, according to the target dataset statistics. Initialized from Structure-from-Motion (SfM) (Schonberger & Frahm, 2016) points, we train the octree-based LoD hierarchy for 40K iterations. During rendering, the octree-based anchor representation with its MLPs is loaded into memory, anchors relevant to the current viewpoint are selected, and Gaussians are decoded from the selected anchors and rendered. In the experiment, we measure resident memory usage across all anchors and MLPs and include anchor selection, decoding, and Gaussian rasterization in the measured rendering time.

FLoD FLoD (Seo et al., 2025) is a from-scratch LoD framework that constructs an independent LoD representation in a coarse-to-fine manner. For a fair setup, we apply the same densification and pruning techniques that are used to train our base representation to FLoD training. Initialized from SfM points, it progressively constructs five levels of detail from coarse to fine. After training each LoD level, the next level is initialized from the preceding levels and trained for 10K, 15K, 20K, 25K, and 30K iterations, respectively (100K iterations in total). For rendering, we use only LoD Gaussians at levels 3, 4, and 5, and load all of them into memory. During rendering, we select Gaussians for each viewpoint according to the depth range of each LoD level and render only the selected Gaussians.

CLOD-3DGS CLOD-3DGS (Milef et al., 2025) is a post-training LoD framework that constructs a shared LoD representation through opacity-based sorting and stochastic budget training. As the input Gaussian set for CLOD-3DGS, we employ our base 3DGS representation trained with several techniques tailored for large-scale scene reconstruction for a fair comparison. Given a set of input Gaussians, it first sorts the Gaussians by opacity and fine-tunes the representation for 30K iterations by randomly selecting a high-opacity Gaussian prefix at each iteration. During rendering, we use the top 50% opacity prefix for Hierarchical-3DGS and 75% for Zip-NeRF, then select view-relevant Gaussians based on depth and render only the selected Gaussians. In the experiments, we measure resident memory usage over all Gaussians and include Gaussian selection and rendering in the measured throughput.

LODGE LODGE (Kulhanek et al., 2025) is a post-training LoD framework that constructs an independent LoD representation in a fine-to-coarse manner. As the input Gaussian set for LODGE, we employ our base 3DGS representation trained with several techniques tailored for large-scale scene reconstruction for a fair comparison. Given a set of input Gaussians, it duplicates them across all LoD levels and imposes a depth-dependent 3D smoothing filter (Yu et al., 2024) to induce different level granularities. From fine to coarse, it prunes less-contributing Gaussians for each level and fine-tunes them for 3K iterations per level. During these 3K iterations, pruning is conducted using the importance score (Niemeyer et al., 2025) every 1K iterations, with pruning thresholds $(\epsilon^0_{\text{prune}}, \epsilon^1_{\text{prune}}, \epsilon^2_{\text{prune}}) = (0.004, 0.012, 0.02)$. For rendering, we load all LoD Gaussians across levels into memory and select and render only the Gaussian subsets relevant to each spatial chunk. In the experiments, we measure resident memory for all LoD Gaussians and working memory only for the selected Gaussian subsets.

CLoD-GS CLoD-GS (Cheng et al., 2026) is a from-scratch LoD framework that constructs a shared LoD representation through distance-dependent opacity decay. For a fair setup, we apply the same densification and pruning techniques that are used to train our base representation to CLoD-GS training. Initialized from SfM points, CLoD-GS constructs an LoD hierarchy using a learnable distance-decay factor for each Gaussian, which progressively attenuates opacity with distance. In the experiments, we measure resident memory usage over all shared Gaussians and include both Gaussian selection and rendering in the reported throughput.

C Additional Results

C.1 Performance on Mobile Devices

As reported in Table 5, we evaluate the rendering efficiency on two mobile devices: iPhone 14 Pro (4 GB) and MacBook Air M3 (16 GB). For rendering on mobile platforms, we employ the Web-based 3DGS renderer (Kellogg, 2025) without any out-of-core rendering techniques to evaluate the efficiency of the underlying representations under the same rendering setup. On the MacBook Air M3, both baselines and SAS-LoD run successfully, while SAS-LoD achieves notably higher rendering speed and enables real-time performance (30+ FPS).

In contrast, on the iPhone 14 Pro, all baseline approaches fail to render due to memory limitations. This indicates that existing approaches require additional algorithmic optimization for mobile rendering (e.g., host streaming), while SAS-LoD achieves real-time performance without rendering-specific optimization. Such rendering optimizations are orthogonal to our approach and can likewise be applied to SAS-LoD, potentially improving the performance of both our method and the base-

lines. Thus, the observed efficiency advantage of SAS-LoD is not dependent on the absence of such optimizations.

Consequently, these results demonstrate that our highly efficient unified framework for 3DGS rendering is capable of scalable rendering even on mobile platforms.

Table 5: **Performance on mobile devices**. We evaluate the rendering speed (FPS) on mobile platforms. '✗' refers to rendering failures.

	iPhone 14 Pro		MacBook Air M3	
	Campus	Small City	Campus	Small City
3DGS	✗	✗	24	28
H3DGS	✗	✗	34	27
CLOD-3DGS	✗	✗	28	38
LODGE	✗	✗	45	52
Ours	31	49	52	54

C.2 Storage Size

Table 6 reports the resident memory and on-disk storage size of our representation for each scene. Beyond efficient runtime resident memory, the storage achieves higher compression ratios when combined with compression codecs. All scenes require less than 80 MB for modeling a large-scale scene. Therefore, our compact representation provides both runtime memory efficiency and on-disk storage efficiency, making it practical for deployment in resource-constrained devices, especially with respect to both memory and disk storage.

Table 6: **Detailed usage of storage size.** All values in MB.

	Campus	Small City	Alameda	Berlin	London	NYC
Position	22.4	22.1	28.0	32.0	22.7	25.5
Scale	5.6	5.5	7.1	8.1	5.7	6.5
Rotation	5.9	5.7	7.3	8.3	5.9	6.6
Opacity	2.0	1.9	2.5	2.8	2.0	2.2
SH DC	5.6	5.4	7.0	8.0	5.6	6.4
SH rest	5.8	5.6	7.3	8.2	5.8	6.6
Residual	5.3	4.1	7.0	7.0	5.8	3.9
Total	52.7	50.3	66.2	74.5	53.4	57.6

C.3 Ablation Study

LoD design. Table 7 reports detailed ablation results on the number of LoD levels. To isolate the impact of selective attribute sharing, we do not apply compression in this experiment. SAS-LoD achieves both efficient memory consumption and high rendering quality, while independent LoD representations incur increasing resident memory with the number of LoD levels, and shared LoD representations lack visual fidelity.

C.4 Per-Scene Results

Per-scene comparison. Table 8 and 9 demonstrate the quantitative comparison on each scene of Hierarchical-3DGS (Kerbl et al., 2024) and Zip-NeRF (Barron et al., 2023).

Per-scene ablation. Table 10 presents per-scene ablation results for the individual components of our framework on Hierarchical-3DGS (Kerbl et al., 2024) and Zip-NeRF (Barron et al., 2023).

Table 7: **Ablation study on LoD design.**

Method	#LoD	#Param ↓	#G ↓	FPS ↑	Memory (GB)			Quality		
					Res ↓	Work ↓	Total ↓	PSNR ↑	SSIM ↑	LPIPS ↓
Baseline (3DGS)	$L = 1$	123.5M	685K	134	0.46	0.46	0.92	26.15	0.812	0.303
Independent LoD	$L = 2$	198.7M	429K	195	0.74	0.51	1.25	26.46	0.814	0.301
Shared LoD		123.5M	424K	198	0.46	0.41	0.87	26.30	0.812	0.305
SAS-LoD (Ours)		128.5M	427K	199	0.48	0.41	0.89	26.38	0.814	0.303
Independent LoD	$L = 3$	229.3M	300K	239	0.86	0.28	1.14	26.71	0.817	0.298
Shared LoD		123.5M	298K	248	0.46	0.29	0.75	26.44	0.814	0.303
SAS-LoD (Ours)		130.5M	292K	252	0.49	0.28	0.77	26.62	0.816	0.299
Independent LoD	$L = 4$	239.2M	276K	246	0.89	0.28	1.18	26.81	0.818	0.296
Shared LoD		123.5M	274K	256	0.46	0.37	0.83	26.51	0.815	0.301
SAS-LoD (Ours)		131.0M	269K	259	0.49	0.28	0.77	26.67	0.817	0.298
Independent LoD	$L = 5$	242.7M	272K	243	0.90	0.29	1.19	26.94	0.819	0.294
Shared LoD		123.5M	270K	255	0.46	0.29	0.75	26.64	0.816	0.299
SAS-LoD (Ours)		131.2M	265K	259	0.49	0.28	0.77	26.85	0.818	0.296

Table 8: **Quantitative results on Hierarchical-3DGS.** We compare our method with baseline approaches on Hierarchical-3DGS (Kerbl et al., 2024). The first, second, and third best scores are highlighted.

Method	Campus							
	#G ↓	FPS ↑	Memory (GB)			Quality		
			Res ↓	Work ↓	Total ↓	PSNR ↑	SSIM ↑	LPIPS ↓
3DGS (Kerbl et al., 2023)	638K	136	0.47	0.52	0.98	25.56	0.815	0.327
H3DGS (Kerbl et al., 2024)	382K	103	0.95	0.44	1.39	25.30	0.811	0.337
Octree-GS (Ren et al., 2025)	282K	124	0.29	0.39	0.67	25.01	0.802	0.355
FLoD (Seo et al., 2025)	202K	152	1.02	0.41	1.43	24.98	0.799	0.360
CLOD-3DGS (Milef et al., 2025)	310K	175	0.47	0.34	0.81	24.81	0.800	0.351
LODGE (Kulhanek et al., 2025)	378K	169	0.87	0.46	1.33	25.89	0.817	0.326
CLoD-GS (Cheng et al., 2026)	370K	136	0.48	0.60	1.07	25.47	0.812	0.338
Ours	331K	186	0.06	0.36	0.42	25.94	0.817	0.327

Method	Small City							
	#G ↓	FPS ↑	Memory (GB)			Quality		
			Res ↓	Work ↓	Total ↓	PSNR ↑	SSIM ↑	LPIPS ↓
3DGS (Kerbl et al., 2023)	733K	131	0.46	0.39	0.85	26.74	0.809	0.279
H3DGS (Kerbl et al., 2024)	320K	164	0.93	0.26	1.19	26.54	0.806	0.289
Octree-GS (Ren et al., 2025)	409K	115	0.52	0.56	1.08	26.09	0.810	0.271
FLoD (Seo et al., 2025)	249K	205	0.90	0.26	1.16	25.97	0.783	0.332
CLOD-3DGS (Milef et al., 2025)	370K	217	0.46	0.23	0.69	26.32	0.801	0.297
LODGE (Kulhanek et al., 2025)	252K	304	0.76	0.23	0.99	26.73	0.809	0.283
CLoD-GS (Cheng et al., 2026)	391K	199	0.54	0.34	0.88	26.40	0.802	0.294
Ours	254K	327	0.05	0.20	0.26	26.97	0.812	0.278

Table 9: **Quantitative results on Zip-NeRF.** We compare our method with baseline approaches on Zip-NeRF (Barron et al., 2023). The first , second , and third best scores are highlighted.

Method	Alameda							
	#G ↓	FPS ↑	Memory (GB)			Quality		
			Res ↓	Work ↓	Total ↓	PSNR ↑	SSIM ↑	LPIPS ↓
3DGS (Kerbl et al., 2023)	631K	197	0.58	0.61	1.19	22.58	0.750	0.364
H3DGS (Kerbl et al., 2024)	445K	125	1.23	0.55	1.78	22.60	0.748	0.368
Octree-GS (Ren et al., 2025)	316K	100	0.37	0.47	0.84	23.05	0.753	0.364
FLoD (Seo et al., 2025)	265K	111	1.04	0.71	1.74	23.07	0.739	0.376
CLOD-3DGS (Milef et al., 2025)	478K	215	0.58	0.53	1.11	22.70	0.751	0.363
LODGE (Kulhanek et al., 2025)	495K	162	1.19	0.65	1.84	22.88	0.750	0.364
CLoD-GS (Cheng et al., 2026)	534K	176	0.85	0.65	1.50	22.95	0.752	0.360
Ours	349K	235	0.07	0.42	0.49	23.02	0.753	0.363

Method	Berlin							
	#G ↓	FPS ↑	Memory (GB)			Quality		
			Res ↓	Work ↓	Total ↓	PSNR ↑	SSIM ↑	LPIPS ↓
3DGS (Kerbl et al., 2023)	813K	121	0.66	0.99	1.65	27.91	0.902	0.295
H3DGS (Kerbl et al., 2024)	623K	80	1.44	0.96	2.39	27.92	0.901	0.297
Octree-GS (Ren et al., 2025)	140K	71	0.23	0.63	0.86	27.37	0.889	0.320
FLoD (Seo et al., 2025)	434K	40	1.10	2.13	3.22	28.01	0.897	0.303
CLOD-3DGS (Milef et al., 2025)	611K	134	0.66	0.94	1.60	27.68	0.901	0.296
LODGE (Kulhanek et al., 2025)	572K	95	1.30	1.00	2.30	27.81	0.900	0.296
CLoD-GS (Cheng et al., 2026)	677K	96	1.01	1.18	2.19	27.96	0.901	0.293
Ours	396K	138	0.08	0.78	0.85	28.09	0.902	0.295

Method	London							
	#G ↓	FPS ↑	Memory (GB)			Quality		
			Res ↓	Work ↓	Total ↓	PSNR ↑	SSIM ↑	LPIPS ↓
3DGS (Kerbl et al., 2023)	540K	232	0.47	0.56	1.03	26.56	0.830	0.335
H3DGS (Kerbl et al., 2024)	390K	150	1.05	0.51	1.56	26.60	0.829	0.337
Octree-GS (Ren et al., 2025)	226K	174	0.24	0.31	0.55	26.15	0.822	0.349
FLoD (Seo et al., 2025)	275K	117	0.94	0.74	1.68	26.25	0.817	0.353
CLOD-3DGS (Milef et al., 2025)	406K	273	0.47	0.48	0.95	26.17	0.824	0.339
LODGE (Kulhanek et al., 2025)	424K	186	0.94	0.63	1.58	26.61	0.830	0.335
CLoD-GS (Cheng et al., 2026)	468K	197	0.66	0.63	1.29	26.29	0.826	0.335
Ours	306K	273	0.06	0.43	0.48	26.66	0.831	0.334

Method	NYC							
	#G ↓	FPS ↑	Memory (GB)			Quality		
			Res ↓	Work ↓	Total ↓	PSNR ↑	SSIM ↑	LPIPS ↓
3DGS (Kerbl et al., 2023)	582K	206	0.53	0.54	1.07	27.81	0.859	0.294
H3DGS (Kerbl et al., 2024)	381K	137	1.11	0.49	1.60	27.78	0.859	0.296
Octree-GS (Ren et al., 2025)	163K	156	0.26	0.24	0.50	27.40	0.850	0.309
FLoD (Seo et al., 2025)	267K	103	0.88	0.72	1.60	27.23	0.845	0.308
CLOD-3DGS (Milef et al., 2025)	440K	239	0.53	0.45	0.98	27.51	0.857	0.298
LODGE (Kulhanek et al., 2025)	329K	185	0.89	0.53	1.42	27.73	0.857	0.294
CLoD-GS (Cheng et al., 2026)	530K	179	0.79	0.61	1.40	27.76	0.857	0.292
Ours	260K	249	0.06	0.36	0.42	27.81	0.857	0.294

Table 10: **Ablation study on overall pipeline.** We analyze rendering efficiency and quality on Hierarchical-3DGS (Kerbl et al., 2024) and Zip-NeRF (Barron et al., 2023).

Method	Campus							
	#G ↓	FPS ↑	Memory (GB)			Quality		
			Res ↓	Work ↓	Total ↓	PSNR ↑	SSIM ↑	LPIPS ↓
3DGS	638K	136 (1.0×)	0.47 (1.0×)	0.52 (1.0×)	0.98 (1.0×)	25.56	0.815	0.327
+ SAS-LoD	331K	183 (1.3×)	0.50 (0.9×)	0.36 (1.4×)	0.86 (1.1×)	26.18	0.819	0.323
+ Compression	638K	134 (1.0×)	0.05 (9.4×)	0.50 (1.0×)	0.55 (1.8×)	25.42	0.813	0.332
+ Both (Ours)	331K	186 (1.4×)	0.06 (8.5×)	0.36 (1.4×)	0.42 (2.4×)	25.94	0.817	0.327

Method	Small City							
	#G ↓	FPS ↑	Memory (GB)			Quality		
			Res ↓	Work ↓	Total ↓	PSNR ↑	SSIM ↑	LPIPS ↓
3DGS	733K	131 (1.0×)	0.46 (1.0×)	0.39 (1.0×)	0.85 (1.0×)	26.74	0.809	0.279
+ SAS-LoD	254K	320 (2.4×)	0.48 (1.0×)	0.20 (1.9×)	0.68 (1.2×)	27.06	0.814	0.275
+ Compression	733K	132 (1.0×)	0.05 (9.3×)	0.40 (1.0×)	0.45 (1.9×)	26.71	0.808	0.282
+ Both (Ours)	254K	327 (2.5×)	0.05 (8.6×)	0.20 (1.9×)	0.26 (3.3×)	26.97	0.812	0.278

Method	Alameda							
	#G ↓	FPS ↑	Memory (GB)			Quality		
			Res ↓	Work ↓	Total ↓	PSNR ↑	SSIM ↑	LPIPS ↓
3DGS	631K	197 (1.0×)	0.58 (1.0×)	0.61 (1.0×)	1.19 (1.0×)	22.58	0.750	0.364
+ SAS-LoD	349K	230 (1.2×)	0.62 (0.9×)	0.42 (1.5×)	1.04 (1.1×)	23.10	0.755	0.358
+ Compression	631K	199 (1.0×)	0.06 (9.4×)	0.59 (1.0×)	0.65 (1.8×)	22.48	0.747	0.369
+ Both (Ours)	349K	235 (1.2×)	0.07 (8.4×)	0.42 (1.5×)	0.49 (2.4×)	23.02	0.753	0.363

Method	Berlin							
	#G ↓	FPS ↑	Memory (GB)			Quality		
			Res ↓	Work ↓	Total ↓	PSNR ↑	SSIM ↑	LPIPS ↓
3DGS	813K	121 (1.0×)	0.66 (1.0×)	0.99 (1.0×)	1.65 (1.0×)	27.91	0.902	0.295
+ SAS-LoD	396K	138 (1.1×)	0.70 (0.9×)	0.77 (1.3×)	1.47 (1.1×)	28.27	0.904	0.292
+ Compression	813K	122 (1.0×)	0.07 (9.4×)	0.97 (1.0×)	1.04 (1.6×)	27.76	0.900	0.297
+ Both (Ours)	396K	138 (1.1×)	0.08 (8.5×)	0.78 (1.3×)	0.85 (1.9×)	28.09	0.902	0.295

Method	London							
	#G ↓	FPS ↑	Memory (GB)			Quality		
			Res ↓	Work ↓	Total ↓	PSNR ↑	SSIM ↑	LPIPS ↓
3DGS	540K	232 (1.0×)	0.47 (1.0×)	0.56 (1.0×)	1.03 (1.0×)	26.56	0.830	0.335
+ SAS-LoD	306K	268 (1.2×)	0.50 (0.9×)	0.43 (1.3×)	0.92 (1.1×)	26.74	0.833	0.332
+ Compression	540K	233 (1.0×)	0.05 (9.3×)	0.56 (1.0×)	0.61 (1.7×)	26.49	0.828	0.337
+ Both (Ours)	306K	273 (1.2×)	0.06 (8.3×)	0.43 (1.3×)	0.48 (2.1×)	26.66	0.831	0.334

Method	NYC							
	#G ↓	FPS ↑	Memory (GB)			Quality		
			Res ↓	Work ↓	Total ↓	PSNR ↑	SSIM ↑	LPIPS ↓
3DGS	582K	206 (1.0×)	0.53 (1.0×)	0.54 (1.0×)	1.07 (1.0×)	27.81	0.859	0.294
+ SAS-LoD	260K	244 (1.2×)	0.55 (1.0×)	0.36 (1.5×)	0.91 (1.2×)	27.95	0.860	0.291
+ Compression	582K	208 (1.0×)	0.06 (9.3×)	0.51 (1.1×)	0.57 (1.9×)	27.68	0.856	0.297
+ Both (Ours)	260K	249 (1.2×)	0.06 (8.7×)	0.36 (1.5×)	0.42 (2.5×)	27.81	0.857	0.294