CC-GS: Low-Memory 3D Gaussian Splatting Training via CPU-GPU Block-Wise Context Compositing

Jian Xu[1] **Siyi Wu**[1] **Yi Li**[2] **Bingzhe Li**[2] **Sian Jin**[3]
Wei Niu[4] **Sheng Di**[5] **Yuede Ji**[1] **Miao Yin**[1]
[1]The University of Texas at Arlington [2]The University of Texas at Dallas [3]Temple University
[4]University of Georgia [5]Argonne National Laboratory
{jxx3451, sxw8121}@mavs.uta.edu {yi.li3, bingzhe.li}@utdallas.edu
sian.jin@temple.edu wniu@uga.edu sdi1@anl.gov
{yuede.ji, miao.yin}@uta.edu

Abstract

Training a single 3D Gaussian Splatting (3DGS) scene routinely requires tens of gigabytes of GPU memory, placing standard 3DGS training beyond many low-memory GPU budgets commonly found on laptops and edge devices. Existing low-memory approaches mainly follow two directions: block-wise training and frustum-culling-based host offloading. Block-wise methods train spatial blocks independently and merge them afterward. However, because 3DGS renders each image through depth-ordered alpha compositing over all visible Gaussians, independent block training reduces optimization to block-local objectives, thereby losing full-image loss supervision. Frustum-culling-based host offloading retains full-image loss supervision by jointly rendering all visible Gaussians while uploading only the view-visible subset to the GPU. However, its GPU memory consumption still scales with the size of the visible set, which can exceed the memory budget of low-end GPUs for dense scenes or wide-coverage views. We propose CC-GS, a low-memory block-wise 3DGS training framework based on CPU-GPU block-wise context compositing, featuring three key designs. *i) Block-wise context compositing*: CC-GS partitions Gaussians into capacity-bounded blocks and represents non-optimized visible blocks using cached per-pixel color and residual transmittance, with depth used only for ordering, enabling full-image loss supervision while processing one capacity-bounded block at a time. *ii) Two-pass block-wise training*: each training view is processed through a lightweight context pass for caching compositing context and a differentiable block-optimization pass for updating individual blocks. *iii) Asynchronous CPU-GPU execution*: CC-GS overlaps block gathering, host-device transfers, rendering, backpropagation, gradient offloading, and CPU-side updates through an asynchronous CPU-GPU pipeline. Experiments on standard 3DGS benchmarks show that CC-GS keeps the measured peak GPU memory below 2 GB for all evaluated scenes while matching the rendering quality of standard 3DGS. Compared with standard 3DGS, CC-GS incurs at most a $1.54\times$ dataset-level training slowdown.

1 Introduction

3D Gaussian Splatting (3DGS) [14] has become a dominant representation for novel view synthesis, combining photorealistic rendering quality with real-time rendering speed. Training 3DGS, however, is memory-intensive: standard training keeps per-Gaussian parameters, gradients, and optimizer states on the GPU, and additionally allocates view-dependent rendering intermediates for backpropa-

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

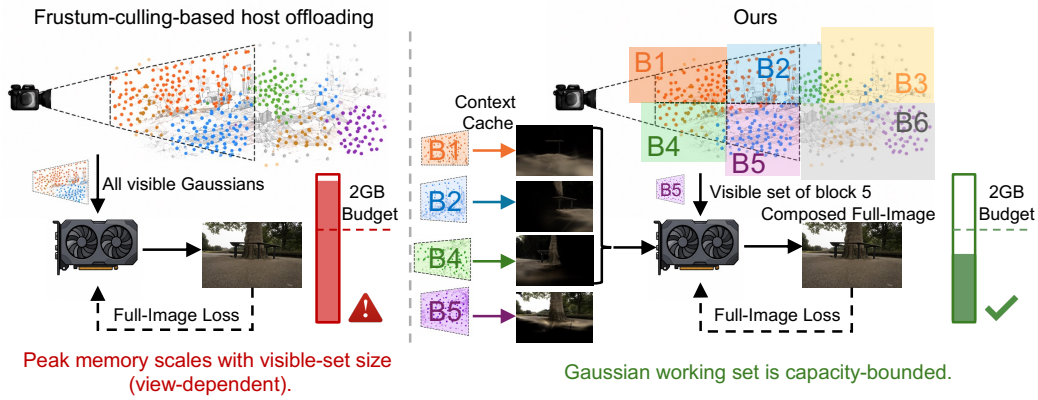


Figure 1: CC-GS shrinks the GPU training unit from the full view-visible set to a capacity-bounded block. Unlike frustum-culling-based offloading, whose memory scales with view coverage, CC-GS optimizes one block at a time and uses cached context to preserve full-image loss supervision.

gation [47, 48, 18]. With millions of Gaussians per scene, this GPU footprint routinely reaches tens of gigabytes, creating a substantial barrier for memory-constrained training settings [26, 46].

Existing methods that reduce GPU residency during training follow a common intuition: keeping only a subset of Gaussians on the GPU at a time. *Divide-and-conquer* methods [21, 24, 4, 38, 12] partition the scene into spatial blocks and train each block independently before merging them afterward. This reduces the GPU working set to a block-sized subproblem. However, because each 3DGS image is formed by compositing all visible Gaussians, each block is optimized in isolation from the others. As a result, full-image loss supervision is replaced by a collection of block-local objectives, which can lead to cross-block inconsistencies in the merged model [22, 39, 35, 18].

A more faithful strategy – *frustum-culling-based host offloading* [48, 18] keeps the full Gaussian set in host memory and uploads only the Gaussians visible to the current view. Because all visible Gaussians are still rendered together, this approach retains full-image loss supervision. However, frustum culling [9] removes only invisible Gaussians and does not explicitly bound the visible set itself. Consequently, dense scenes or wide-coverage views can still expose a large fraction of Gaussians, causing GPU memory usage to exceed low-memory budgets [48, 18].

Together, these methods expose a fundamental unresolved challenge: how to retain full-image loss supervision without treating the entire view-visible Gaussian set as the GPU-resident training unit. To address this and enable high-quality 3DGS training on *strict low-GPU-memory* hardware, we revisit the alpha-compositing [31] rule used in 3DGS rendering [14]. We observe that the alpha-blending rule is compositional. For each pixel, a consecutive segment of the depth-ordered compositing sequence can be compactly summarized by two per-pixel quantities: accumulated color and residual transmittance. With an associated per-pixel depth value used for ordering, such summaries can be recomposed with their surrounding context using the same front-to-back compositing rule.

Building on this observation, we propose CC-GS, a low-GPU-memory block-wise 3DGS training method with three key designs:

- **Block-wise context compositing.** CC-GS partitions Gaussians into capacity-bounded blocks and loads only the target block into GPU memory during optimization. The remaining visible blocks are represented using cached per-pixel color, residual transmittance, and depth, enabling full-image loss supervision while processing one capacity-bounded block at a time.
- **Two-pass block-wise training.** Each training view is processed in two passes: (1) a lightweight gradient-free context pass that renders visible blocks and caches their compositing context, and (2) a differentiable block-optimization pass that loads and optimizes one block at a time using the cached context. Gaussian parameters and optimizer states are stored in host memory, with parameter updates performed on the CPU.
- **Pipelined asynchronous CPU-GPU execution.** To mitigate the overhead introduced by repeated block-wise processing and data movement, CC-GS employs an asynchronous CPU-GPU pipeline that overlaps block gathering, host-device transfers, rendering, back-propagation, gradient offloading, and CPU-side updates.

We conduct comprehensive experiments on standard 3DGS benchmarks. The results show that CC-GS keeps measured peak GPU memory below 2 GB on every evaluated scene while matching the rendering quality of standard 3DGS. Compared with standard 3DGS, CC-GS incurs at most a $1.54\times$ dataset-level slowdown, whereas prior offloading baselines [48, 18] can still exceed this memory budget on dense or wide-coverage scenes.

2 Background and Related Work

2.1 3D Gaussian Splatting

3D Gaussian Splatting (3DGS) [14] encodes the target scene with a collection of anisotropic 3D Gaussians. Each Gaussian G_i is described by its 3D position $\boldsymbol{\mu}_i$, covariance $\boldsymbol{\Sigma}_i$ (parameterized by rotation and scaling in practice), learned opacity $o_i \in [0, 1]$, and view-dependent color $\mathbf{c}_i \in \mathbb{R}^3$. Its spatial distribution is defined as

$$G(\mathbf{x}) = \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right), \tag{1}$$

where $\mathbf{x}$ is a 3D position. For each training view, 3DGS frustum-culls Gaussians, projects visible ones to 2D splats, and sorts the splats covering each pixel front-to-back. For pixel p, let G'_i be the value of the i-th projected Gaussian at p, and $\alpha_i = o_i G'_i$ its rendering opacity. The pixel color is alpha-blended as

$$C_p = \sum_i \mathbf{c}_i \alpha_i T_i, \qquad T_i = \prod_{j=1}^{i-1}(1-\alpha_j), \tag{2}$$

where T_i is the transmittance of the i-th Gaussian, computed from preceding Gaussians' rendering opacities. This compositing underpins our block-wise reformulation in Sec. 3.1.

Training proceeds by minimizing a photometric loss between rendered and ground-truth images, with Gaussian parameters updated via gradient descent. An adaptive densification procedure periodically clones, splits, or prunes Gaussians based on gradient statistics, causing the scene's total Gaussian count to grow over the course of training.

2.2 Memory-Efficient 3DGS Training

Compact 3DGS Representations. Compact 3DGS methods are not primarily low-memory training systems, but can reduce training memory by shrinking the optimized Gaussian state. These methods mainly reduce either the number of Gaussians or the per-Gaussian state. The former controls densification to grow fewer redundant primitives [16, 5, 23, 26, 30], simplifies or sparsifies the Gaussian set through sampling, pruning, or optimization-based simplification [7, 6, 45], or organizes Gaussians into anchor-based [25, 36, 3, 43], hierarchical [15, 11, 33, 42], or level-of-detail structures [32, 40]. The latter compresses per-Gaussian attributes, such as view-dependent appearance parameters, SH coefficients, or full Gaussian parameter vectors, using more compact representations [19, 29, 27, 28, 8]. However, compaction does not by itself change the training residency pattern: standard training still keeps or materializes the parameters, gradients, optimizer states, and rendering intermediates of the current model on the GPU together. CC-GS is orthogonal to these methods by bounding the GPU-resident training unit via block-wise execution.

Reducing GPU Residency During Training. Another line of work changes the training residency pattern itself by avoiding keeping all Gaussians resident on the GPU. One approach is to partition the scene into spatial blocks or sub-scenes. VastGaussian [21], CityGaussian [24], DOGS [4], and BlockGaussian [38] follow this divide-and-conquer strategy, while LoBE-GS [12] further studies load imbalance across blocks. These methods reduce the GPU working set to a block-sized subproblem. Optimizing each block in isolation replaces full-image loss supervision with a collection of block-local objectives, often requiring special boundary handling when independently trained blocks are merged [22, 39, 35, 18]. Frustum-culling-based host offloading methods such as CLM [48] and GS-Scale [18] take a different approach. They offload the dominant SH coefficients and their optimizer states to host memory, while keeping the remaining Gaussian parameters and optimizer states on the GPU. For each training view, the view-visible subset of the offloaded states is transferred back to the GPU, enabling all visible Gaussians to be jointly rendered and optimized and thus retaining full-image

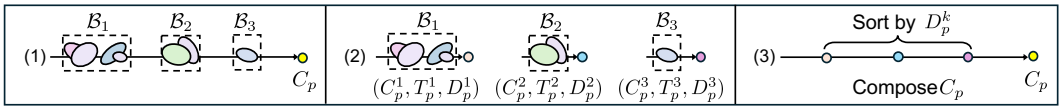


Figure 2: Compositional structure of alpha-blending in 3DGS. (1) Partition Gaussians into disjoint spatial blocks. (2) Render each visible block to obtain per-pixel color, residual transmittance, and depth. (3) Composite block-level results in depth order to obtain the final per-pixel color.

loss supervision. However, the GPU memory footprint still scales with the view-visible set, because dense or wide-coverage views materialize more visible states and rendering intermediates on the GPU, which can exceed the target budget on GPU-memory-limited devices. Splaxel [13] preserves global consistency across scene partitions by exchanging locally rendered pixel contributions and compositing them in depth order.

CC-GS develops this compositing principle for low-memory CPU-GPU execution, using spatial partitioning as an execution mechanism rather than as a decomposition of the training objective. It retains full-image loss supervision while reducing the GPU training unit from the full view-visible set to a capacity-bounded block, making it complementary to representation-level compaction and distinct from frustum-culling-based offloading.

3 Methodology

CC-GS substantially reduces the Gaussian working set on the GPU during 3DGS training by combining capacity-bounded spatial partitioning, block-wise context compositing, and an asynchronous CPU-GPU execution scheme. Instead of loading all Gaussians visible to a training view onto the GPU, CC-GS processes the view-visible subset of one spatial block at a time. For each training view, a lightweight context preparation pass renders each visible block in isolation without gradients and caches its block-local per-pixel color, residual transmittance, and depth value. A subsequent block-optimization pass loads one block at a time as the target block, re-renders it with gradients, composes its output with the cached contexts of the other blocks in per-pixel depth order, and computes the loss on the resulting full-image prediction. The asynchronous CPU-GPU execution scheme overlaps block preparation, data movement, GPU computation, gradient offloading, and CPU-side updates to hide the overhead introduced by block-wise execution.

3.1 Context Compositing for 3DGS Training

Spatial Partitioning. We build a KD-tree over Gaussian centers by recursively splitting over-capacity nodes at the count median along the longest spatial axis until each leaf contains at most N_{cap} Gaussians. The leaves form capacity-bounded spatial blocks $\{\mathcal{B}_k\}_{k=1}^{K}$. For a sampled view v, frustum culling [9, 1] is applied to each block, and only its view-visible subset is processed. As densification and pruning change the Gaussian set, blocks exceeding N_{cap} are dynamically re-partitioned by the same rule.

Context Compositing Formulation. The blocks serve as execution units rather than independent sub-scenes. As illustrated in Fig. 2, our method couples them through context compositing: per-block rendering results are summarized per pixel and then composited in depth order to form the full-image prediction. Splaxel [13] uses per-partition, per-pixel summaries for global compositing and gradient computation in distributed 3DGS training. We adapt this compositing principle to memory-bounded single-GPU training: cached summaries of non-target blocks provide context, while one capacity-bounded block is optimized at a time under full-image supervision. Formally, for a pixel p and a visible block $\mathcal{B}_k$, let the Gaussians covering p be evaluated in within-block front-to-back depth order. Rendering $\mathcal{B}_k$ with unit incoming transmittance yields its color contribution and residual transmittance:

$$C_p^k = \sum_{i \in \mathcal{B}_k} \mathbf{c}_i \alpha_i \prod_{j<i} (1-\alpha_j), \qquad T_p^k = \prod_{i \in \mathcal{B}_k} (1-\alpha_i). \tag{3}$$

Here C_p^k is the color contributed by $\mathcal{B}_k$, and T_p^k is the residual transmittance after compositing this block. The notation $j < i$ refers to Gaussians preceding i in the within-block order at pixel p.

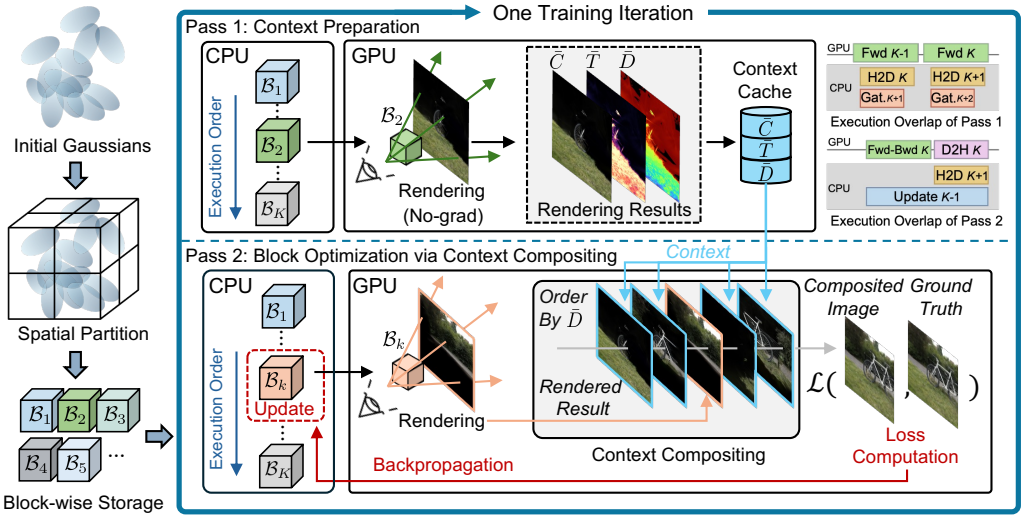


Figure 3: Overview of CC-GS. Each iteration first prepares view-specific context from all visible blocks, then revisits the blocks one at a time for context-composited optimization, keeping only one block's Gaussian state resident on the GPU.

To order block summaries, we also compute an opacity-weighted partial depth:

$$D_p^k = \sum_{i \in \mathcal{B}_k} d_i \alpha_i \prod_{j<i} (1 - \alpha_j). \tag{4}$$

Here d_i denotes the inverse view-space depth of Gaussian i. The scalar D_p^k is used only as a block-level ordering key at pixel p, rather than as a supervised or normalized depth map.

Given the summaries (C_p^k, T_p^k, D_p^k) of visible blocks, context compositing is performed at the block level. For each pixel, we sort the blocks by D_p^k in descending order and re-index them as $k = 1, \ldots, R$ in front-to-back block order. Gaussian extents can cross partition boundaries, so contributions from different blocks may interleave along the same viewing ray. In such cases, block-wise compositing provides an approximation to standard 3DGS rendering. Appendix E discusses the equivalence conditions and empirical effects. The final pixel color is

$$C_p = \sum_{k=1}^{R} C_p^k \prod_{m=1}^{k-1} T_p^m. \tag{5}$$

In Eq. (5), k denotes this sorted per-pixel block order, not the original spatial block index. Each term uses the color summary of one block, weighted by the residual transmittance of preceding blocks. Thus, non-target blocks can provide cached context during optimization, while only the target block needs to be re-rendered with gradients. This motivates the following two-pass training. A visualization of these cached per-block summaries is provided in App. A.

Two-Pass Training. We implement Eq. (5) with the following two-pass procedure. Fig. 3 gives a schematic overview of the workflow, while Alg. 1 details the procedure for one sampled view.

► *Pass 1: Context Preparation.* Each view-visible block subset is rendered without gradients to produce $(\bar{C}^k, \bar{T}^k, \bar{D}^k)$, which forms the cached context for the current view. Since no computation graph is recorded, this pass is lightweight.

► *Pass 2: Block Optimization via Context Compositing.* Each block is revisited as the target block and rendered with gradient tracking, producing differentiable outputs (C^k, T^k). These outputs replace the cached target-block summaries in Eq. (5), while all other contexts remain fixed. The cached depths $\{\bar{D}^j\}_{j=1}^K$ provide the per-pixel block order for recomposition. The recomposed image is compared with the ground truth, so gradients are computed only for the target block. After backpropagation, these gradients are copied back to host memory and applied by the CPU-side optimizer.

Context Cache Footprint. The cached context is compact. For each visible block, it contains the per-pixel summaries (C^k, T^k, D^k), i.e., five scalars per pixel: three for C^k, one for T^k, and one for D^k. For a view with M visible blocks and resolution $H \times W$, this costs $5MHW$ scalars, only $(5/3)M$ times an RGB image. This image-level cache is much smaller than the view-visible Gaussian parameters, whose footprint grows with the number of visible Gaussians.

Backpropagation. During a target-block update, only the rendering outputs of the target block are differentiable. The cached summaries of the other blocks provide fixed context for recomposition. For compactness, we define the accumulated transmittance before block $\mathcal{B}_k$ and the color context after it as

Algorithm 1: CC-GS Block-wise Training

Require: $\mathcal{P}$, $\mathcal{V}$, capacity N_{cap}, loss $\mathcal{L}$
Partition $\mathcal{P} \to \{\mathcal{B}_k\}_{k=1}^{K}$ under N_{cap}
for each iteration **do**
  Sample view $v \in \mathcal{V}$
  $\mathcal{S}_k(v) \leftarrow \textsc{FrustumCull}(\mathcal{B}_k, v)$ for all k
  for $k = 1$ to K **do** ▷ Pass 1: Context Preparation
    $\textsc{Load}\ \mathcal{S}_k(v)$ to GPU
    $(\bar{C}^k, \bar{T}^k, \bar{D}^k) \leftarrow \textsc{Render}(\mathcal{S}_k(v), v)$
    Cache $(\bar{C}^k, \bar{T}^k, \bar{D}^k)$ on GPU
  end for
  for $k = 1$ to K **do** ▷ Pass 2: Block Optimization
    $\textsc{Load}\ \mathcal{S}_k(v)$ to GPU
    $(C^k, T^k) \leftarrow \textsc{Render}(\mathcal{S}_k(v), v)$
    $\widehat{C}^{(k)} \leftarrow \textsc{Compose}\big((C^k, T^k, \bar{D}^k), (\bar{C}, \bar{T}, \bar{D})_{j \neq k}\big)$
    $\mathcal{L}_k \leftarrow \mathcal{L}(\widehat{C}^{(k)})$
    Backpropagate $\mathcal{L}_k$ and update $\mathcal{B}_k$ on CPU
  end for
  Re-partition blocks whose sizes exceed N_{cap}
end for

$$\Lambda_p^k = \prod_{m<k} T_p^m, \qquad \Phi_p^k = \sum_{\ell>k} C_p^\ell \prod_{s<\ell,\, s\neq k} T_p^s. \tag{6}$$

For a Gaussian $i \in \mathcal{B}_k$, the recomposed pixel color depends on its parameters only through C_p^k and T_p^k. Applying the chain rule gives

$$\frac{\partial C_p}{\partial \alpha_i} = \Lambda_p^k \frac{\partial C_p^k}{\partial \alpha_i} + \Phi_p^k \frac{\partial T_p^k}{\partial \alpha_i}, \qquad \frac{\partial C_p}{\partial \mathbf{c}_i} = \Lambda_p^k \frac{\partial C_p^k}{\partial \mathbf{c}_i}. \tag{7}$$

Here all block indices follow the sorted per-pixel compositing order in Eq. (5). The context terms Λ_p^k and Φ_p^k are computed from cached non-target blocks and are treated as constants during this update. The image loss provides $\partial \mathcal{L}/\partial C_p$ at each pixel, which is backpropagated only through the currently optimized block.

Implications. This block-wise formulation remains compatible with the standard 3DGS training recipe, including pruning, adaptive density control, and learning-rate scheduling. The statistics required by density control are accumulated across blocks, so these global training decisions do not require all Gaussians to be differentiable on the GPU at the same time. The additional rendering and data-movement overhead introduced by this two-pass process is handled by the asynchronous CPU-GPU execution scheme described next.

3.2 Pipelined CPU-GPU Asynchronous Parameter Update

The two-pass design in Sec. 3.1 gives each block update full-image loss supervision through block-wise context compositing, but a synchronous implementation would expose substantial CPU-GPU transfer latency and GPU idle time. CC-GS hides these costs behind GPU computation through a pipelined CPU-GPU execution scheme.

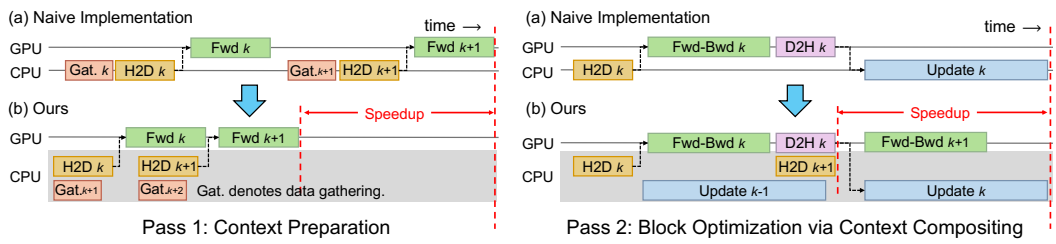
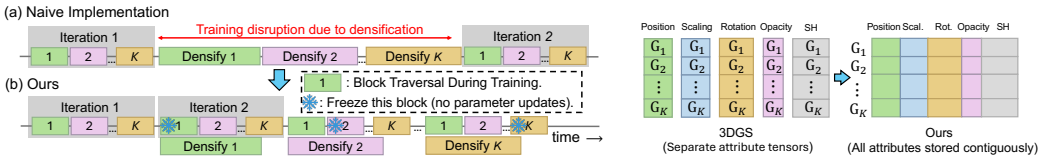


Figure 4: Execution timeline of the asynchronous pipeline in the two passes. Pass 1 overlaps data gathering, H2D transfer of block data, and no-gradient forward rendering. Pass 2 overlaps rendering/backpropagation, D2H transfer of gradients, and CPU-side updates. H2D and D2H denote host-to-device and device-to-host transfer, respectively; k denotes the k-th block.



(a) Asynchronous densification (b) Streaming-friendly layout

Figure 5: System optimizations for pipelined block-wise training. (a) Asynchronous densification avoids pipeline stalls by freezing only the affected block while densifying its copy in the background. (b) The streaming-friendly data layout packs Gaussian attributes into contiguous per-Gaussian rows to reduce visible-subset gathering overhead.

Block-Wise Overlap. Fig. 4 shows the asynchronous schedule. A naive implementation serializes block gathering, H2D transfer, GPU rendering/backpropagation, gradient offloading, and CPU-side updates, while CC-GS schedules these stages across CPU and GPU streams to hide data movement and CPU-side work behind GPU computation. Pass 1 has no backward stage, so the schedule overlaps forward rendering of block k with gathering and H2D preparation for block $k{+}1$. Pass 2 additionally overlaps rendering/backpropagation on block k with gradient writeback for block $k{-}1$ and preparation for block $k{+}1$. With two block-sized GPU buffers, one for the block being processed and one for prefetching the next block, the GPU can proceed without waiting for CPU-side gathering or updates.

Asynchronous Densification. The asynchronous pipeline above handles regular block updates, but densification requires separate treatment. In standard 3DGS training, it is triggered every 100 iterations during early training. Since densification changes Gaussian counts and optimizer states, a synchronous implementation would repeatedly interrupt the pipeline and introduce substantial overhead. As shown in Fig. 5(a), CC-GS avoids blocking the main pipeline through asynchronous densification. When density control is triggered for a block, the block is frozen in the main pipeline: it still provides rendering context, but is skipped for differentiable updates while a separate copy is densified in the background. Once densification finishes, the densified copy replaces the frozen block and resumes normal training. Since each block is frozen only briefly, the effect is limited to a small fraction of block updates while densification stalls are removed.

Streaming-Friendly Data Layout. The asynchronous pipeline can hide CPU-side work only if block gathering, gradient writeback, and CPU-side optimizer updates fit within the GPU computation window. Otherwise, these memory accesses become exposed on the critical path, starving the GPU and breaking the intended overlap. This makes CPU-side data layout critical for pipelined block-wise training. In the default 3DGS layout, Gaussian attributes are stored in separate tensors, so each visible subset requires multiple attribute-wise gathers, scattered gradient writes, and optimizer accesses. As shown in Fig. 5(b), we reduce this overhead by packing all attributes of each Gaussian into a single row-major block buffer $\Theta_k \in \mathbb{R}^{n_k \times D}$ for each block $\mathcal{B}_k$. With this layout, gathering a visible subset becomes a single row gather, and gradient writeback and CPU-side optimizer updates are applied to the same rows in place, keeping CPU-side work efficient enough to overlap with GPU computation.

4 Experiments

4.1 Experimental Setup

Datasets and Baselines. We evaluate on 13 scenes from three widely used benchmarks: Mip-NeRF 360 [2] (5 outdoor + 4 indoor), Tanks & Temples [17] (2 scenes), and Deep Blending [10] (2 scenes). Following prior work, outdoor scenes are trained at $4\times$ downsampled resolution and indoor scenes at $2\times$. For fairness, we do not modify the standard 3DGS evaluation protocol: each dataset uses its original train/test split, and each scene keeps the same input resolution across all compared methods. We compare against standard 3DGS, CLM [48], and GS-Scale [18], which are the closest frustum-culling-based host offloading baselines to our method.

Evaluation Metrics. We report PSNR, SSIM [37], and LPIPS [44] for rendering quality. For memory, we report the peak GPU memory usage during training, measured at the device level using NVIDIA NVML. For efficiency, we report total wall-clock training time, including data loading. For completeness, we also report peak CPU memory usage on Mip-NeRF 360 in App. C.

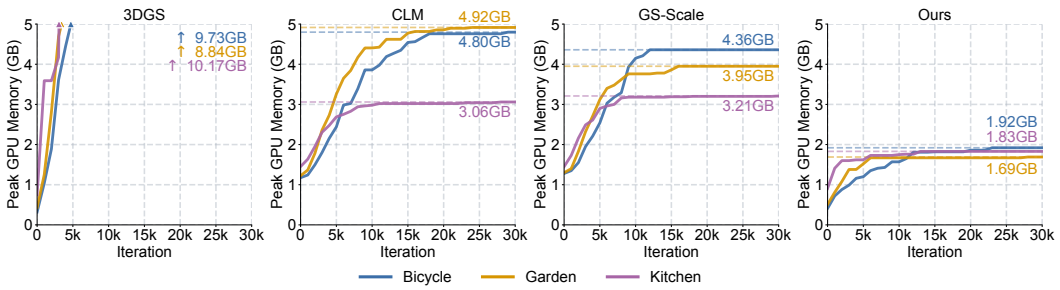


Figure 6: Running peak GPU memory (GB, lower is better) during training on *Bicycle*, *Garden*, and *Kitchen*, comparing standard 3DGS, GS-Scale, CLM, and our method.

Table 1: Peak GPU memory usage during training across all datasets (GB, lower is better)

	Mip-NeRF 360									Tanks & Temples		Deep Blending	
Method	bicycle	stump	garden	treehill	flowers	kitchen	room	counter	bonsai	truck	train	drjohnson	playroom
3DGS	9.73	8.13	8.84	6.40	6.10	10.17	6.02	10.85	6.86	5.08	4.97	10.53	6.08
CLM	4.80	3.56	4.92	3.43	3.53	3.06	2.29	2.27	2.07	2.92	2.42	4.07	2.63
GS-Scale	4.36	3.43	3.95	3.13	3.21	3.21	2.23	2.32	2.15	2.83	2.73	4.39	2.53
Ours	**1.92**	**1.63**	**1.69**	**1.55**	**1.73**	**1.83**	**1.80**	**1.66**	**1.44**	**1.49**	**1.78**	**1.98**	**1.72**

Implementation Details. All experiments are conducted on a workstation equipped with an NVIDIA GeForce RTX 4090 GPU, an Intel Xeon w5-3435X CPU, and a PCIe Gen4 connection. All methods follow the standard 3DGS training protocol unless otherwise stated, and baseline-specific memory-management strategies are adopted from their official implementations. We use a block capacity of $N_{\text{cap}} = 0.8$M Gaussians by default.

4.2 Experimental Results

Peak GPU Memory During Training. CC-GS substantially reduces the GPU memory required for 3DGS training compared with existing offloading-based baselines by differentiating only one capacity-bounded block at a time while using cached context to preserve the full-image loss. Fig. 6 plots the running peak GPU memory during training on three scenes. Standard 3DGS rapidly exceeds 5 GB early in training, while CLM, GS-Scale, and CC-GS operate in a much lower memory regime. Among these low-memory methods, CC-GS further reduces peak memory, with its curve remaining below 2 GB across all three scenes. On *Bicycle* and *Garden*, CC-GS uses less than half of CLM's peak memory (1.92 vs. 4.80 GB and 1.69 vs. 4.92 GB, respectively). Tab. 1 further reports the peak GPU memory across all 13 scenes. CC-GS consistently achieves substantially lower peak training memory than the baselines while keeping the measured peak below 2 GB in every case.

Reconstruction Quality. While keeping peak training memory below 2 GB, CC-GS maintains reconstruction quality comparable to standard 3DGS. Tab. 2 reports per-dataset averages of PSNR, SSIM, and LPIPS. Compared with standard 3DGS, CLM, GS-Scale, and CC-GS all preserve reconstruction quality within a small margin. CLM and GS-Scale preserve the original full-image loss by jointly rendering all Gaussians visible to the current view. In contrast, CC-GS further introduces block-wise execution, but still computes the loss on a recomposed full-image prediction by combining the block being optimized with cached context from the other visible blocks. Therefore, unlike divide-and-conquer training, CC-GS avoids reducing training to block-local objectives, which explains its stable reconstruction quality. The full per-scene breakdown is provided in App. B.

Table 2: Reconstruction quality averaged per dataset. The rightmost columns report each method's overall average minus that of standard 3DGS.

Method	Tanks & Temples			Deep Blending			Mip-NeRF 360			Δ vs 3DGS (avg)		
	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓
3DGS	23.81	0.850	0.170	29.72	0.903	0.241	27.44	0.814	0.217	+0.00	+0.000	+0.000
CLM	23.81	0.845	0.181	29.62	0.905	0.244	27.39	0.814	0.218	-0.05	-0.000	+0.003
GS-Scale	23.80	0.852	0.172	29.70	0.906	0.237	27.71	0.814	0.200	+0.18	+0.001	-0.012
Ours	23.76	0.848	0.169	29.75	0.912	0.242	27.36	0.808	0.218	-0.06	-0.003	+0.001

Training Time. CC-GS incurs modest training-time overhead over standard 3DGS. Fig. 7 compares training time across datasets. On Mip-NeRF 360 and Tanks & Temples, CC-GS is $1.26\times$ and $1.54\times$ slower than standard 3DGS, respectively, matching CLM and remaining faster than GS-Scale on both datasets. On Deep Blending, CC-GS is comparable to standard 3DGS and CLM, while GS-Scale is slightly faster. This efficiency is enabled by the system design in Sec. 3.2: block-wise overlap, asynchronous densification, and the streaming-friendly data layout together hide most overhead that block-wise training would otherwise expose. The remaining overhead mainly comes from the gradient-free context preparation pass, which is lightweight. GS-Scale slows more on dense scenes, consistent with its sensitivity to the size of the view-visible set.

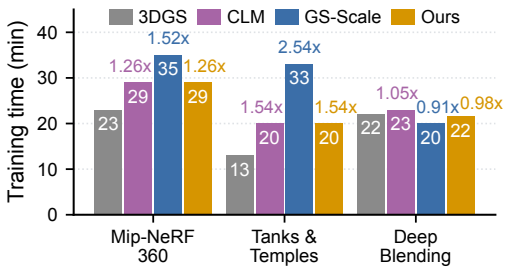


Figure 7: End-to-end training time across datasets. CC-GS adds modest overhead over 3DGS but avoids GS-Scale's large slowdowns on Mip-NeRF 360 and Tanks & Temples.

4.3 Ablation Study

We organize the ablations around the three roles of CC-GS. First, we test whether the core algorithmic components are necessary for preserving full-image supervision and the memory bound. Second, we sweep the block capacity to evaluate peak-memory control and its runtime cost. Finally, we ablate the runtime pipeline components that make block-wise training practical.

Context Compositing and Dynamic Block Maintenance. Tab. 3 ablates the two key components of CC-GS on the challenging *Bicycle* scene. Without context compositing, training reduces to optimizing each block under block-local supervision. Peak memory remains low, but PSNR drops by 6.04 dB, confirming that the full-image loss recomposed from cached context is essential to preserving quality. Without re-partitioning, reconstruction quality remains close to the full method, but peak memory grows from 1.92 GB to 3.35 GB as densification causes some blocks to exceed the capacity bound. Thus, context compositing preserves quality, while re-partitioning prevents densification from violating the capacity constraint behind memory control.

Table 3: Component ablation on *Bicycle*. Peak memory is reported in GB. We ablate context compositing and re-partitioning from the full method.

Variant	PSNR↑	SSIM↑	LPIPS↓	Peak Mem↓
Full (Ours)	25.10	0.756	0.220	1.92
w/o context comp.	19.06	0.484	0.417	1.63
w/o re-partitioning	25.02	0.762	0.219	3.35

Block Capacity as a Peak-Memory Knob. Tab. 4 shows that block capacity is the main knob for peak-memory control under a fixed rendering configuration. Reducing capacity from 1.2M to 0.4M lowers peak memory from 2.14 GB to 1.52 GB, but increases runtime from 18 to 47 minutes due to more blocks and streaming work. PSNR, SSIM, and LPIPS remain close across the sweep, suggesting that capacity mainly sets the memory-runtime trade-off rather than reconstruction quality. The default 0.8M setting keeps peak memory below 2 GB and is used in the main results.

Table 4: Effect of block capacity on *Bicycle*. Peak memory is reported in GB and runtime in minutes.

Capacity	#Blocks	Peak Mem↓	Time↓	PSNR↑	SSIM↑	LPIPS↓
0.4M	16	1.52	47	25.10	0.753	0.221
0.6M	12	1.68	34	25.09	0.752	0.222
0.8M	8	1.92	24	25.10	0.756	0.220
1.2M	6	2.14	18	25.10	0.754	0.220

Runtime Pipeline Components. Block-wise full-image supervision adds data movement and CPU-side bookkeeping, making a naive implementation slow. Fig. 8 decomposes this overhead on *Bicycle*. The Base version uses separate block tensors, synchronous transfers, and immediate structural updates, taking 5h34m. Asynchronous overlap reduces runtime to 1h18m, asynchronous densification further reduces it to 40 minutes, and packed block layout brings it to 24 minutes. Together, these components make block-wise supervision practical in wall-clock time.

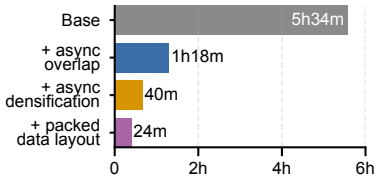


Figure 8: Runtime pipeline ablation on *Bicycle*. Bars report total training time.

5 Conclusion, Limitations, and Broader Impact

CC-GS introduces a low-memory 3DGS training method by partitioning Gaussians into capacity-bounded blocks and differentiating one target block at a time. By recomposing the target block with cached context, CC-GS computes the loss on a recomposed full-image prediction while bounding the GPU-resident Gaussian working set. The remaining overhead mainly comes from the additional gradient-free context preparation pass and the increased number of block operations when smaller capacities are used. As a result, lower memory usage can require longer training time. Context staleness and temporary block freezing can affect optimization; App. F evaluates these effects and discusses memory and runtime limitations. Future work will explore more efficient block streaming and context preparation. By lowering training-time memory requirements, CC-GS can make dense 3D reconstruction more accessible on memory-constrained hardware, while requiring responsible use in privacy-sensitive scenes.

Acknowledgments and Disclosure of Funding

We thank the NVIDIA Academic Research Program for its support. We declare no competing interests.

References

[1] Tomas Akenine-Moller, Eric Haines, and Naty Hoffman. *Real-time rendering*. AK Peters/CRC Press, 2019.

[2] Jonathan T. Barron, Ben Mildenhall, Dor Verbin, Pratul P. Srinivasan, and Peter Hedman. Mip-NeRF 360: Unbounded anti-aliased neural radiance fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5470–5479, June 2022.

[3] Yihang Chen, Qianyi Wu, Weiyao Lin, Mehrtash Harandi, and Jianfei Cai. Hac: Hash-grid assisted context for 3d gaussian splatting compression. In *European Conference on Computer Vision*, pages 422–438. Springer, 2024.

[4] Yu Chen and Gim Hee Lee. Dogs: Distributed-oriented gaussian splatting for large-scale 3d reconstruction via gaussian consensus. *Advances in Neural Information Processing Systems*, 37: 34487–34512, 2024.

[5] Kai Cheng, Xiaoxiao Long, Kaizhi Yang, Yao Yao, Wei Yin, Yuexin Ma, Wenping Wang, and Xuejin Chen. Gaussianpro: 3d gaussian splatting with progressive propagation. In *Forty-first International Conference on Machine Learning*, 2024.

[6] Zhiwen Fan, Kevin Wang, Kairun Wen, Zehao Zhu, Dejia Xu, and Zhangyang Wang. Light-gaussian: Unbounded 3d gaussian compression with 15x reduction and 200+ fps. *Advances in neural information processing systems*, 37:140138–140158, 2024.

[7] Guangchi Fang and Bing Wang. Mini-splatting: Representing scenes with a constrained number of gaussians. In *European conference on computer vision*, pages 165–181. Springer, 2024.

[8] Sharath Girish, Kamal Gupta, and Abhinav Shrivastava. Eagles: Efficient accelerated 3d gaussians with lightweight encodings. In *European Conference on Computer Vision*, pages 54–71. Springer, 2024.

[9] Gil Gribb and Klaus Hartmann. Fast extraction of viewing frustum planes from the world-view-projection matrix. *Online document*, 2001.

[10] Peter Hedman, Julien Philip, True Price, Jan-Michael Frahm, George Drettakis, and Gabriel Brostow. Deep blending for free-viewpoint image-based rendering. *ACM Transactions on Graphics (TOG)*, 37(6):257:1–257:15, 2018.

[11] He Huang, Wenjie Huang, Qi Yang, Yiling Xu, and Zhu Li. A hierarchical compression technique for 3d gaussian splatting compression. In *ICASSP 2025-2025 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1–5. IEEE, 2025.

[12] Sheng-Hsiang Hung, Ting-Yu Yen, Wei-Fang Sun, Simon See, Shih-Hsuan Hung, and Hung-Kuo Chu. LOBE-GS: Load-balanced and efficient 3d gaussian splatting for large-scale scene reconstruction. *arXiv preprint arXiv:2510.01767*, 2025.

[13] Wenqi Jia, Zhewen Hu, Ying Huang, Yu Gong, Stavros Kalafatis, Yuke Wang, Wei Niu, Chengming Zhang, Ang Li, Sheng Di, Yuede Ji, Bo Fang, and Miao Yin. Splaxel: Efficient distributed training of 3D gaussian splatting for large-scale scene reconstruction via pixel-level communication. *arXiv preprint arXiv:2606.18588*, 2026. URL `https://arxiv.org/abs/2606.18588`.

[14] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3D gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics (TOG)*, 42(4): 139:1–139:14, 2023.

[15] 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.

[16] Sieun Kim, Kyungjin Lee, and Youngki Lee. Color-cued efficient densification method for 3d gaussian splatting. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 775–783. IEEE, 2024.

[17] Arno Knapitsch, Jaesik Park, Qian-Yi Zhou, and Vladlen Koltun. Tanks and temples: Benchmarking large-scale scene reconstruction. *ACM Transactions on Graphics*, 36(4), 2017.

[18] Donghyun Lee, Dawoon Jeong, Jae W Lee, and Hongil Yoon. GS-Scale: Unlocking large-scale 3d gaussian splatting training via host offloading. In *Proceedings of the 31st ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pages 860–875, 2026.

[19] 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.

[20] Yixuan Li, Lihan Jiang, Linning Xu, Yuanbo Xiangli, Zhenzhi Wang, Dahua Lin, and Bo Dai. MatrixCity: A large-scale city dataset for city-scale neural rendering and beyond. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 3205–3215, 2023.

[21] 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 *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5166–5175. IEEE, 2024.

[22] Chuandong Liu, Huijiao Wang, Lei Yu, and Gui-Song Xia. Holistic large-scale scene reconstruction via mixed gaussian splatting. *Advances in Neural Information Processing Systems*, 38: 93297–93316, 2026.

[23] Rong Liu, Rui Xu, Yue Hu, Meida Chen, and Andrew Feng. Atomgs: Atomizing gaussian splatting for high-fidelity radiance field. *arXiv preprint arXiv:2405.12369*, 2024.

[24] 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*, pages 265–282. Springer, 2024.

[25] Tao Lu, Mulin Yu, Linning Xu, Yuanbo Xiangli, Limin Wang, Dahua Lin, and Bo Dai. Scaffold-gs: Structured 3d gaussians for view-adaptive rendering. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 20654–20664, 2024.

[26] 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.

[27] 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*, pages 330–349. Springer, 2024.

[28] 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*, pages 10349–10358, 2024.

[29] Panagiotis Papantonakis, Georgios Kopanas, Bernhard Kerbl, Alexandre Lanvin, and George Drettakis. Reducing the memory footprint of 3d gaussian splatting. *Proceedings of the ACM on Computer Graphics and Interactive Techniques*, 7(1):1–17, 2024.

[30] Stéphane Pateux, Matthieu Gendrin, Luce Morin, Théo Ladune, and Xiaoran Jiang. Bogauss: Better optimized gaussian splatting. In *2025 33rd European Signal Processing Conference (EUSIPCO)*, pages 765–769. IEEE, 2025.

[31] 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.

[32] 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. *arXiv preprint arXiv:2403.17898*, 2024.

[33] Shengji Tang, Weicai Ye, Peng Ye, Weihao Lin, Yang Zhou, Tao Chen, and Wanli Ouyang. Hisplat: Hierarchical 3d gaussian splatting for generalizable sparse-view reconstruction. *arXiv preprint arXiv:2410.06245*, 2024.

[34] 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 (CVPR)*, pages 12922–12931, 2022.

[35] Mengyi Wang, Beiqi Chen, Shengfang Pan, Niansheng Liu, and Jinhe Su. Consistency-preserving gaussian splatting for block-based large-scale scene reconstruction. *Computers & Graphics*, page 104493, 2025.

[36] Yufei Wang, Zhihao Li, Lanqing Guo, Wenhan Yang, Alex C Kot, and Bihan Wen. Contextgs: Compact 3d gaussian splatting with anchor level context model. *Advances in neural information processing systems*, 37:51532–51551, 2024.

[37] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4): 600–612, 2004.

[38] Yongchang Wu, Zipeng Qi, Zhenwei Shi, and Zhengxia Zou. Blockgaussian: Efficient large-scale scene novel view synthesis via adaptive block-based gaussian splatting. *arXiv preprint arXiv:2504.09048*, 2025.

[39] Shijie Xu and Qiulei Dong. CAG-GS: Consistent anchor guided gaussian splatting for large-scale scene rendering. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pages 11388–11396, 2026.

[40] Xijie Yang, Linning Xu, Lihan Jiang, Dahua Lin, and Bo Dai. Virtualized 3d gaussians: Flexible cluster-based level-of-detail system for real-time rendering of composed scenes. In *Proceedings of the Special Interest Group on Computer Graphics and Interactive Techniques Conference Conference Papers*, pages 1–11, 2025.

[41] Vickie Ye, Ruilong Li, Justin Kerr, Matias Turkulainen, Brent Yi, Zhuoyang Pan, Otto Seiskari, Jianbo Ye, Jeffrey Hu, Matthew Tancik, et al. gsplat: An open-source library for gaussian splatting. *Journal of Machine Learning Research*, 26(34):1–17, 2025.

[42] Fengyi Zhang, Yadan Luo, Tianjun Zhang, Lin Zhang, and Zi Huang. Gaussianforest: Hierarchical-hybrid 3d gaussian splatting for compressed scene modeling. *arXiv preprint arXiv:2406.08759*, 2024.

[43] Jiahui Zhang, Fangneng Zhan, Ling Shao, and Shijian Lu. Sogs: Second-order anchor for advanced 3d gaussian splatting. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11167–11176, 2025.

[44] 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*, pages 586–595, 2018.

[45] Yangming Zhang, Wenqi Jia, Wei Niu, and Miao Yin. Gaussianspa: An "optimizing-sparsifying" simplification framework for compact and high-quality 3d gaussian splatting. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 26673–26682, 2025.

[46] Yangming Zhang, Jian Xu, Chaojian Li, Kunxiong Zhu, Wei Niu, Gagan Agrawal, Yang Katie Zhao, Jian Wang, Yingyan Celine Lin, and Miao Yin. Gaussians on a diet: High-quality memory-bounded 3d gaussian splatting training. *arXiv preprint arXiv:2604.20046*, 2026.

[47] 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, pages 35642–35661, 2025.

[48] Hexu Zhao, Xiwen Min, Xiaoteng Liu, Moonjun Gong, Yiming Li, Ang Li, Saining Xie, Jinyang Li, and Aurojit Panda. CLM: Removing the gpu memory barrier for 3d gaussian splatting. In *Proceedings of the 31st ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pages 377–393, 2026.

A Visualization of Context Compositing

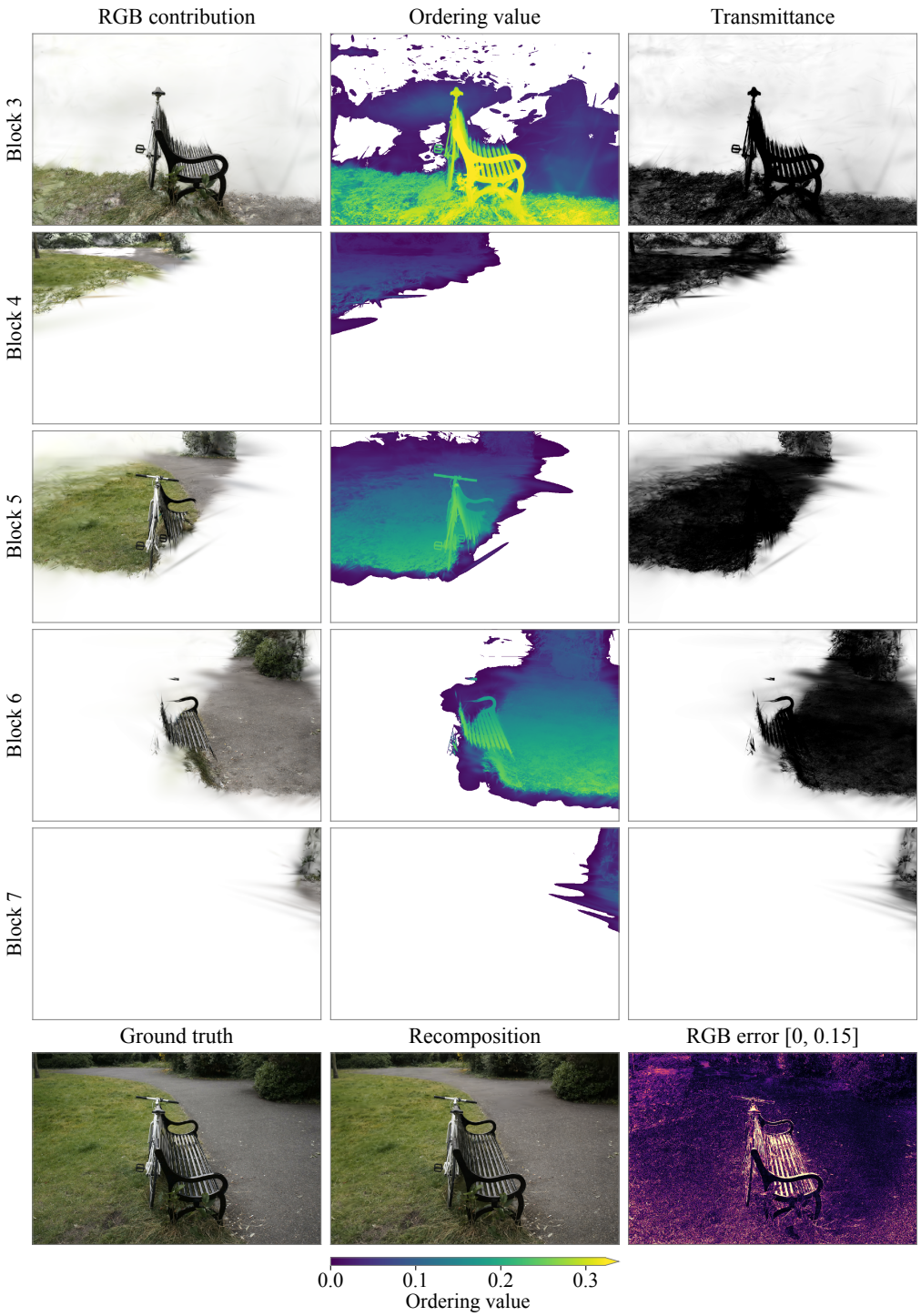


Figure 9: Per-block summaries and final composition. A held-out Bicycle view (CC-GS, 30k iterations). The model has eight blocks; blocks 0–2 do not contribute to this view and are omitted. Rows 1–5 show block RGB, opacity-weighted inverse depth, and transmittance; the last row shows ground truth, recomposition, and mean absolute RGB error. The recomposed image shows no conspicuous seam artifacts at the displayed scale.

Boundary-region inspection. Figure 10 compares block-wise recomposition with one-pass rendering of the same learned Gaussians using identical rendering settings. The full view in Fig. 9 and the enlarged crops show visually continuous scene structure, with no conspicuous seam artifacts at the displayed scales. At the pixel level, the RGB difference maps reveal local texture and brightness variations between the two rendering paths.

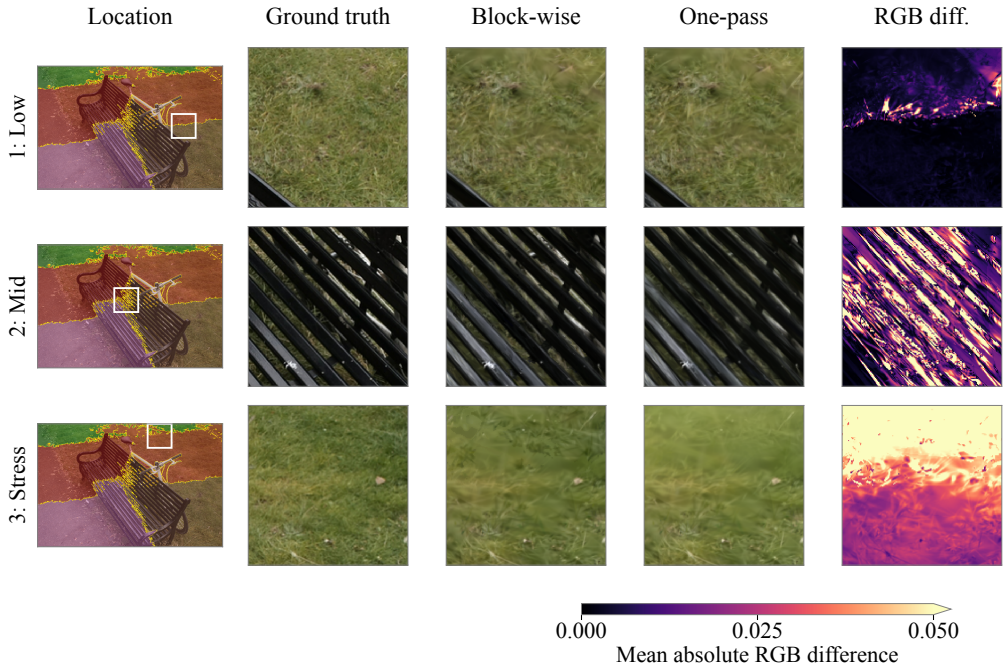


Figure 10: Boundary-region comparison on Bicycle. The crops compare ground truth, block-wise recomposition, and one-pass rendering of the same learned Gaussians. RGB diff shows the per-pixel absolute difference between the two renderings, averaged over the RGB channels. Yellow lines mark changes in the main contributing block. Low/Mid indicate local image detail; Stress shows a crop selected for large rendering differences.

B Per-Scene Rendering Quality

Table 5 expands the per-dataset averages of Table 2 into per-scene numbers for all 13 scenes. On every scene our method stays within the same PSNR/SSIM/LPIPS band as the strongest baseline.

C Host-Memory Usage

Context Compositing stores Gaussian parameters and optimizer states in host memory. Table 6 reports peak CPU memory usage across the nine Mip-NeRF 360 scenes. Peak usage ranges from 11.14 to 16.54 GB, with an average of 13.88 GB.

D Large-Scale Evaluation

We evaluate Mega-NeRF Rubble [34] and MatrixCity BigCity Aerial [20] with a 4 GB GPU-memory budget. A run meets the budget only if its device-level peak GPU memory stays within 4 GB throughout training.

Mega-NeRF Rubble. Training starts from standard initialization and follows the full densification schedule, reaching approximately 9–14M Gaussians, with up to 2.74M visible from one view. Table 7 reports memory, runtime, and rendering quality. CC-GS completes 30k iterations within budget and achieves 26.35 dB PSNR, compared with 26.40 dB for an unrestricted 3DGS reference run.

Table 5: Per-scene rendering quality across Tanks & Temples, Deep Blending, and Mip-NeRF 360.

Scene	3DGS			CLM			GS-Scale			Ours		
	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓
Tanks & Temples												
truck	25.39	0.880	0.143	25.33	0.882	0.149	25.34	0.884	0.144	25.30	0.876	0.141
train	22.23	0.820	0.196	22.30	0.809	0.213	22.25	0.820	0.199	22.21	0.821	0.196
Deep Blending												
drjohnson	29.40	0.903	0.238	29.12	0.901	0.244	29.34	0.905	0.234	29.29	0.913	0.240
playroom	30.04	0.903	0.243	30.11	0.908	0.243	30.05	0.907	0.240	30.21	0.910	0.244
Mip-NeRF 360												
bicycle	25.23	0.765	0.210	25.21	0.764	0.213	25.53	0.766	0.199	25.10	0.756	0.220
stump	26.66	0.771	0.216	26.56	0.771	0.219	26.87	0.773	0.206	26.61	0.785	0.221
garden	27.36	0.864	0.108	27.37	0.867	0.108	27.77	0.873	0.103	27.29	0.856	0.106
treehill	22.51	0.633	0.327	22.42	0.632	0.329	22.88	0.637	0.313	22.67	0.633	0.326
flowers	21.49	0.603	0.339	21.57	0.604	0.337	21.82	0.601	0.330	21.37	0.594	0.338
kitchen	31.28	0.925	0.127	31.00	0.926	0.127	31.34	0.922	0.113	31.23	0.913	0.126
room	31.49	0.918	0.220	31.40	0.918	0.221	31.88	0.916	0.188	31.40	0.907	0.222
counter	28.98	0.906	0.202	28.93	0.907	0.201	29.13	0.905	0.178	28.78	0.900	0.201
bonsai	31.96	0.940	0.206	32.03	0.940	0.207	32.18	0.935	0.174	31.82	0.929	0.204

Table 6: Peak CPU memory usage of CC-GS on Mip-NeRF 360 scenes. Values are reported in GB.

Metric	bicycle	stump	garden	treehill	flowers	kitchen	room	counter	bonsai	Avg.
Peak CPU Mem.	16.54	14.31	16.19	12.27	13.33	14.84	13.53	11.14	12.75	13.88

Table 7: Mega-NeRF Rubble. Times correspond to the reported end step. GS-Scale completes training but exceeds the budget; dashes denote unavailable quality results.

Method	4 GB budget	End step	Peak GPU (GB)	Time	PSNR↑	SSIM↑	LPIPS↓
3DGS	Exceeded	600	4.46	1m 52s	–	–	–
CLM	Exceeded	5,396	4.09	5m 00s	–	–	–
GS-Scale	Exceeded	30,000	4.77	2h 09m	25.52	0.811	0.274
CC-GS	Met	30,000	3.71	2h 28m	26.35	0.861	0.205

MatrixCity BigCity Aerial. This fixed-model-size protocol starts from preconstructed point clouds with 15.3M, 46.0M, or 102.2M Gaussians and tests training feasibility rather than full densification (Table 8). At 102.2M Gaussians, CC-GS completes training at 3.86 GB peak GPU memory with 25.08 dB PSNR; an unconstrained CLM reference at the same scale achieves 25.15 dB using approximately 20.8 GB.

Table 8: MatrixCity BigCity Aerial at fixed model sizes. Numeric entries are peak GPU memory (GB) for completed runs within budget; "Exceeded" denotes a run exceeding 4 GB.

Gaussians	3DGS	CLM	GS-Scale	CC-GS
15.3M	Exceeded	3.82	3.94	3.16
46.0M	Exceeded	Exceeded	Exceeded	3.47
102.2M	Exceeded	Exceeded	Exceeded	3.86

E Compositing Approximation and Its Impact

Source of approximation. For fixed Gaussians and a fixed view, Eq. (5) matches the globally sorted alpha-compositing expression if each block occupies a contiguous segment of the per-pixel sequence and the block order preserves that sequence. Grouping consecutive terms then leaves the sum unchanged. An interleaved sequence such as $A_1 \to B_1 \to A_2$ generally cannot be reproduced by either whole-block order. The depth summary D_p^k does not by itself guarantee correct block

ordering, even without interleaving. Thus, full-image loss supervision does not imply equivalence to standard 3DGS compositing.

Interleaving frequency and reconstruction quality. For diagnosis only, we globally depth-sort the contributing Gaussians of the CC-GS representation at each pixel p. The score I_p measures the fraction of accumulated alpha contribution associated with blocks re-entering the sequence after another block. This global sorting is not part of CC-GS training. Across all test views of the 13 scenes, 92.1% of pixels have $I_p < 1\%$. Table 9 compares both methods against ground truth on the same pixel subsets, selected using the CC-GS scores.

The magnitude of the PSNR gap grows from 0.05 dB overall to 0.45 dB on the 0.28% of pixels with $I_p > 10\%$. These gaps compare separately trained models and do not isolate compositing error. Figure 9 shows the full-view recomposition; Fig. 10 instead holds the learned Gaussians fixed and compares block-wise and one-pass rendering. The selected stress crops reveal local texture and brightness differences, complementing the aggregate quality measurements.

Table 9: Reconstruction quality by interleaving score. PSNR and its gap (CC-GS minus 3DGS) are in dB. Thresholded subsets are nested.

Pixel subset	Pixels (%)	3DGS	CC-GS	Gap
All pixels	100	27.23	27.18	-0.05
$I_p > 1\%$	7.9	24.92	24.82	-0.10
$I_p > 5\%$	1.5	22.67	22.43	-0.24
$I_p > 10\%$	0.28	20.31	19.86	-0.45

F Context Staleness and System Limitations

Within-view context staleness. For each sampled view, we cache summaries of the visible blocks once and keep this snapshot fixed during block updates. Earlier updates are therefore not reflected in the context used by later blocks. With $N_{\text{cap}} = 0.8$M, refreshing after every block increases training time by $7.3\times$ for 0.03 dB higher PSNR (Table 10).

Temporary freezing during densification. A frozen block is re-rendered for each new view using its current parameter snapshot; only its differentiable updates are skipped while its densified copy is prepared. Table 11 reports freeze events on Bicycle. Skipped updates differ from elapsed iterations because the block is not visible in every sampled view. During the densification stage, 8.9% of updates scheduled for affected blocks are skipped; freezing occurs only in the first half of this run. Asynchronous densification reduces training time from 78 to 40 minutes with 0.02 dB lower PSNR (Table 10). These measurements characterize the evaluated settings, not a bound on the effects of freezing or stale context.

Table 10: Cache-refresh and densification ablations. Cache refresh uses the full pipeline; densification timings correspond to the pre-packed-layout stages in Fig. 8. Time is in minutes and PSNR in dB.

Variant	Time↓	PSNR↑	SSIM↑	LPIPS↓
Cache refresh ($N_{\text{cap}} = 0.8$M)				
Once per view (CC-GS)	24	25.10	0.756	0.220
After every block	175	25.13	0.761	0.210
Densification on Bicycle				
Synchronous	78	25.12	0.757	0.219
Asynchronous	40	25.10	0.756	0.220

Scope of memory control. Block capacity bounds active and prefetched Gaussian state, not total GPU memory: context storage depends on image resolution and visible-block count, while rasterization buffers depend on projected Gaussian coverage. Parameters and optimizer states still occupy host memory (App. C), and CPU updates and transfers can become bottlenecks as models grow. Smaller blocks also increase rendering and compositing work. Thus, low GPU-memory usage does not imply low host-memory requirements or uniformly faster training.

Table 11: Per-event freezing statistics on Bicycle.

Measurement	Mean	95th percentile	Maximum
Freeze duration (s)	1.52	2.31	3.08
Elapsed training iterations	18.7	29	41
Skipped block updates	10.4	17	25

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: The abstract and introduction accurately describe the paper's scope as training-time memory reduction for 3DGS. The stated claims on bounded memory, block-wise compositing, and asynchronous execution are supported by the method design and experimental results, while limitations are discussed separately.

   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: The paper discusses limitations in the conclusion, including the extra no-gradient context preparation pass and the increased number of block operations when smaller capacities are used, which leads to a memory-runtime trade-off.

   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 present formal theoretical results such as theorems or lemmas. The compositing equations are used to describe the proposed block-wise training formulation rather than to state formal theoretical claims.

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: We describe the datasets, evaluation metrics, training settings, capacity configurations, and implementation details used in the experiments. The main algorithmic components, including capacity-bounded partitioning, block-wise compositing, and asynchronous CPU-GPU execution, are specified in the method section to support reproduction of the main results.

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 paper includes experiments requiring code, but we do not provide an anonymized code release in the supplemental material at submission time. The datasets used in the experiments are publicly available, and the paper provides the main implementation details, training settings, capacity configurations, and evaluation protocol to support reproducibility. We plan to release a cleaned and documented implementation after publication.

   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: The paper specifies the datasets, train/test splits, evaluation metrics, training iterations, capacity configurations, baseline settings, and hardware used in the experiments. These details are provided in the experimental setup and implementation sections to support understanding and reproduction of the reported 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. Due to the high computational cost of large-scale 3DGS training, each method is trained once per scene, and we instead report per-scene results as well as averages over multiple scenes. To ensure a fair comparison, all methods are evaluated under the same datasets, splits, training iterations, and metric protocol.

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: The paper reports the compute resources used in the experiments, including the GPU hardware, memory requirements, and training runtime. We also report dataset-level training time and per-scene peak GPU memory usage for the main experiments, which help characterize and compare the computational cost of the methods.

   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: We have reviewed the NeurIPS Code of Ethics and believe that the research conforms to it. The work uses public benchmark datasets and does not involve human subjects, private data, or safety-critical deployment.

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 both positive and negative societal impacts. The proposed memory-bounded training framework may reduce the hardware barrier for large-scale 3DGS training, but more efficient 3D reconstruction could also raise privacy or misuse concerns in some applications. The work is primarily a training-system contribution and is evaluated on public benchmark datasets.

    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: The paper does not release high-risk models or datasets that require additional safeguards. The method is evaluated on public benchmark datasets and does not involve private or scraped data.

    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 use publicly released benchmark datasets, including Mip-NeRF 360 [2], Tanks & Temples [17], Deep Blending [10], Mega-NeRF Rubble [34], and MatrixCity BigCity Aerial [20]. We use the official author-provided dataset releases, cite the original papers, and do not redistribute the datasets. Tanks & Temples is released under CC BY 4.0 with additional official terms restricting use to non-commercial scientific research. MatrixCity is released under CC BY-NC 4.0. For Mip-NeRF 360, Deep Blending, and Mega-NeRF Rubble, the official dataset release pages do not specify an explicit dataset-license name; we follow their author-provided usage terms. The official 3DGS [14] and CLM [48] implementations are distributed under the Gaussian-Splatting License, GS-Scale [18] under the MIT License, and gsplat [41] under the Apache License 2.0.

    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 release new datasets, models, or code assets as part of the submission. The proposed method is documented in the paper, and we plan to release a cleaned and documented implementation after publication.

    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: The paper does not involve crowdsourcing experiments or research with human subjects.

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: The paper does not involve crowdsourcing experiments or research with human subjects, so IRB approval or equivalent review is not applicable.

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 and experiments in this research do not involve LLMs as important, original, or non-standard components.

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.