NanoGS: Training-Free Gaussian Splat Simplification

Butian Xiong[1⋆], Rong Liu[1⋆], Tiantian Zhou[1],
Meida Chen[1], Zhiwen Fan[2], and Andrew Feng[1]

[1] USC Institute for Creative Technologies
[2] Texas A&M University

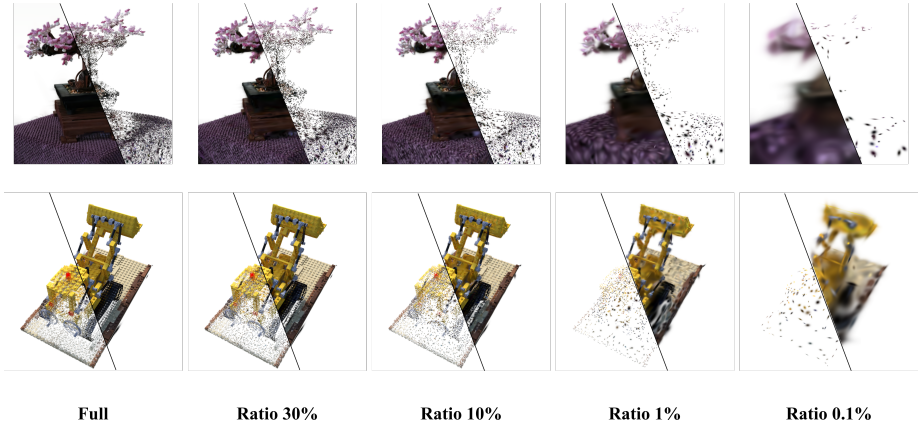

Fig. 1: NanoGS reduces Gaussian Splat primitive count from the full model to increasingly compact representations. **NanoGS** achieves substantial compaction ratio while preserving visual fidelity and geometric structure without GPU-intensive optimization or calibrated images.

Abstract. 3D Gaussian Splat (3DGS) enables high-fidelity, real-time novel view synthesis by representing scenes with large sets of anisotropic primitives, but often requires millions of Splats, incurring significant storage and transmission costs. Most existing compression methods rely on GPU-intensive post-training optimization with calibrated images, limiting practical deployment. We introduce **NanoGS**, a training-free and lightweight framework for Gaussian Splat simplification. Instead of relying on image-based rendering supervision, NanoGS formulates simplification as local pairwise merging over a sparse spatial graph. The method approximates a pair of Gaussians with a single primitive using mass preserved moment matching and evaluates merge quality through a principled merge cost between the original mixture and its approximation. By restricting merge candidates to local neighborhoods and

⋆ Co-first authors, equal technical contribution.

selecting compatible pairs efficiently, NanoGS produces compact Gaussian representations while preserving scene structure and appearance. NanoGS operates directly on existing Gaussian Splat models, runs efficiently on CPU, and preserves the standard 3DGS parameterization, enabling seamless integration with existing rendering pipelines. Experiments demonstrate that NanoGS substantially reduces primitive count while maintaining high rendering fidelity, providing an efficient and practical solution for Gaussian Splat simplification. Our project website is available at https://saliteta.github.io/NanoGS/ .

1 Introduction

Real-time radiance field rendering has been fundamentally reshaped by 3D Gaussian Splatting (3DGS) [24]. By representing scenes as collections of anisotropic 3D Gaussians optimized with adaptive density control, 3DGS achieves photorealistic novel view synthesis while sustaining real-time 1080p rendering performance. Its explicit, rasterization-compatible formulation bridges Neural Radiance Field (NeRF) [36] and classical graphics pipelines, eliminating costly volumetric ray sampling and enabling practical deployment. As a result, Splat-based representations have rapidly become a dominant alternative to NeRF-style models [4,6,36,38], supporting applications in immersive exploration [16,23], dynamic scene reconstruction [50, 53, 54], 3D content creation [45, 55], semantic distillation [42, 52, 59], and cinematic production [46].

Despite this success, a fundamental bottleneck persists: **scalability**, as high-fidelity reconstructions routinely require millions of primitives to capture fine geometric structures and complex view-dependent appearance. In practice, Splat models frequently occupy hundreds of megabytes or even gigabytes, while large primitive sets increase sorting and rasterization overhead during rendering. This structural scaling limits rendering speed, efficient transmission, edge-device deployment, and large-scale distribution of Splat-based content.

Existing methods mitigate this issue through optimization of the parameter dimensionality, quantization of attribute precision, and reduction of primitive counts [1, 2]. A large body of work focuses on parameter- and bit-level compression through vector quantization, entropy coding, adaptive spherical-harmonic reduction, structured parameter layouts, and progressive encoding schemes [15,17,30,39,40]. Other methods integrate pruning or sparsification into retraining pipelines, leveraging importance metrics, regularization, distillation, or optimal-transport-based merging to reduce the number of Splats [29, 47, 56]. However, these approaches often rely on GPU-intensive optimization and calibrated image supervision. This assumption does not hold in many practical scenarios where Splat models are produced through alternative pipelines, such as 3DGS generation [45,55], edition [18,41], or conversions from meshes or point clouds [14, 58], where guided images are not be available. In addition, several compression schemes modify the underlying Gaussian Splat parameterization by introducing codec-specific layouts or structured representations, breaking compatibility with standard Splat rendering pipelines and preventing the compressed

models from being directly reused. Consequently, a training-free, lightweight, and representation-preserving simplification framework that operates *post hoc* on existing 3DGS models remains largely unexplored.

A natural objection is that one could simply re-optimize a smaller model under a tighter primitive budget rather than compact an existing one post hoc. This is viable for object-level or synthetic scenes where the original capture pipeline is on hand, but breaks down under three conditions common at deployment time:

- **Disjoint create/consume hardware.** Reconstruction relies on CUDA-enabled clusters, whereas end users render on smartphones, AR headsets, and WebGL clients that lack the VRAM and stack for an on-device optimization loop, where even efficient on-device training and rendering remain open [13,19]. Under a *create once, publish everywhere* model [22,26], and with transmission formats carrying only compiled geometry rather than source imagery [27], a single asset must serve all devices.
- **Missing supervision.** The dense multi-view supervision that optimization-based compaction needs is often unavailable: source images may be permanently inaccessible (scanners and drone surveys gatekeep raw imagery), and diffusion-driven scene generation produces no ground-truth views [10, 51]. Even feed-forward single-image pipelines [35] supply one view and pose but not the coverage needed to constrain a photometric optimization, leaving re-optimization under-determined.
- **Poor fit to on-the-fly decimation.** A creator rarely knows a client's memory ceiling in advance, and re-running an hours-long optimization per budget is impractical. Resampling new views does not help, as view planning is isomorphic to set cover and thus NP-hard [44], and heuristic sampling introduces occlusion and floater artifacts requiring dedicated repair [49].

In this work, we introduce **NanoGS**, a training-free and lightweight *framework* for Gaussian Splat simplification. Rather than a single merging algorithm, NanoGS decouples simplification into three interchangeable stages—a *candidate topology* that proposes which splats may merge, a *cost objective* that scores each candidate, and a *splat operator* that fuses a selected pair—so that each can be swapped without redesigning the others. We deliberately instantiate them with lightweight choices (a sparse k-nearest-neighbor graph, a Monte Carlo I-divergence merge cost, and mass-preserving moment matching) to show that strong extreme-compaction quality follows from the decoupled structure rather than from operator complexity. Concretely, NanoGS first prunes low-opacity splats, then iteratively builds the KNN graph, scores candidate edges by how well a two-Gaussian mixture is approximated by a single Gaussian via moment matching, and greedily collapses low-cost disjoint pairs into mass-preserving Gaussians until the target ratio is reached. Experiments show that NanoGS substantially reduces primitive count while maintaining high visual fidelity compared to state-of-the-art (SOTA) compression methods. Specifically, across four standard benchmarks (NeRF-Synthetic, Mip-NeRF 360, Tanks&Temples, Deep Blending; 21 scenes total) and three compaction budgets $\rho \in \{0.1, 0.01, 0.001\}$,

NanoGS achieves the best PSNR at every budget and improves over the compared methods by **+2.40 dB**, **+4.84 dB**, and **+5.46 dB** on average, respectively. Importantly, NanoGS operates directly on existing Gaussian Splat models without requiring training images or optimization, preserving the standard representation and remaining fully compatible with existing rendering pipelines. Unlike level-of-detail schemes that couple spatial structure to merging logic for transient rendering [25], NanoGS keeps the two separable and targets permanent asset compaction, a distinction we revisit in Sec. 5 once the stages are formalized.

In summary, our contributions are as follows:

1. **A modular simplification framework** that casts training-free, post-hoc simplification as a decoupled pipeline of three interchangeable stages candidate topology, cost objective, and splat operator. This structure, rather than any single operator, drives quality under aggressive compaction and separates our permanent-compaction setting from rendering-time level-of-detail schemes.
2. **Efficient graph pairing** (topology stage): a sparse spatial neighbor graph with greedy disjoint-pair selection for scalable simplification.
3. **An I-divergence merge cost** (objective stage): scoring edges by the I-divergence between the two-splat mixture and its moment-matched Gaussian, plus an appearance discrepancy.
4. **Mass-Preserved Moment Matching** (operator stage): a principled fusion of two splats into one Gaussian preserving geometric and appearance consistency. Stages 2–4 are deliberately lightweight instantiations chosen to isolate the framework's contribution.
5. **State-of-the-art extreme compaction** with accessible, CPU-only, representation preserving deployment, attaining the best PSNR at every tested budget and the largest margins in the most aggressive regimes.

2 Related Works

Recent work has explored improving the efficiency of 3D Gaussian Splatting (3DGS) models along two complementary directions: *compression* and *compaction* [1, 2]. *Compression* methods aim to reduce the storage footprint and transmission bandwidth of 3DGS assets through efficient parameter representations and attribute quantization. In contrast, *compaction* methods focus on reducing the number of Gaussian primitives while preserving rendering quality, thereby lowering both memory consumption and rasterization cost during rendering. These two directions address different aspects of efficiency and are largely orthogonal: compression minimizes the bit-level storage of its parameters, while compaction simplifies the structural complexity of the representation.

2.1 3DGS Compression: Bit-Level Storage Reduction

While 3D Gaussian Splatting (3DGS) enables real-time photorealistic rendering, early models often produce serialized assets ranging from hundreds of megabytes

to several gigabytes, creating a major bottleneck for storage, transmission, and web-based deployment [1, 8, 39]. To address this issue, a large body of work focuses on *bit-level compression* of Gaussian attributes. These methods typically quantize and entropy-code per-Gaussian parameters such as position, covariance, color coefficients, and opacity, often combined with learned or engineered context models and auxiliary coding structures, including anchor-based predictors and hierarchical entropy models [8, 9, 31, 39, 48].

Representative approaches include Compressed3DGS [39], which employs sensitivity-aware clustering and quantization-aware fine-tuning to achieve up to 31× compression while maintaining rendering quality, and provides a WebGPU-based renderer for efficient browser visualization. Other methods improve storage efficiency by reducing the *per-primitive parameter payload* through compact parameterizations or shared representations, such as reduced bases, codebooks, or alternative primitive formulations, sometimes combined with auxiliary pruning strategies for additional memory savings [11, 32, 33, 40]. Note that our method belongs to the compaction category, reducing the number of Gaussian primitives while preserving the standard 3DGS representation. As a result, it remains fully compatible with existing compression techniques, which can be applied on top of the compacted model to further reduce the overall storage footprint.

While these approaches significantly reduce the storage cost of 3DGS assets, many rely on access to the calibrated training images to optimize compression modules. This assumption does not always hold in practice, particularly when Gaussian Splat models are generated, edited, or converted through alternative pipelines where training images are not required. Moreover, several methods introduce codec-specific parameterizations or auxiliary structures that may limit direct compatibility with standard 3DGS rendering pipelines.

2.2 3DGS Compaction: Primitive-Level Count Reduction

While sharing the goal of reducing the number of Gaussian primitives, existing compaction approaches differ substantially in their operational requirements and optimization strategies.

Training-based compaction. A line of work incorporates primitive reduction directly into the 3DGS training pipeline so that the model converges to fewer splats or a constrained primitive budget [30, 34, 57]. These methods typically introduce additional mechanisms such as learned masking, compact parameterizations, or resource-aware training schedules. For example, Compact3DGS [30] integrates masking and compact representations into the optimization process, while Taming 3DGS [34] explicitly targets high-quality reconstruction under limited resource budgets by controlling training-time allocation. While effective when retraining is possible, such approaches require access to the original training pipeline and therefore cannot serve as drop-in simplification tools for arbitrary pre-trained 3DGS assets.

Image-guided pruning. Another dominant strategy estimates the importance of each Gaussian by measuring its contribution to rendered pixels in the training views. These methods typically require training images, camera poses,

and differentiable rendering signals during the compaction process. Representative examples include LightGaussian and PUP 3D-GS [15, 20], which compute per-splat importance scores based on reconstruction sensitivity and employ prune-and-recover or prune-and-refine stages to maintain rendering quality under high pruning ratios. Although effective, such pipelines depend on rendering supervision and access to the training images, limiting their applicability when only the trained splat model is available. In contrast, our approach is *model-only* and operates directly on the Gaussian set without requiring training images or camera parameters.

Model-only primitive reduction. More closely related to our formulation are classical approaches for Gaussian mixture reduction and clustering based on distributional similarity measures, including Bregman divergences and information-theoretic co-clustering [3, 7, 12, 43]. These techniques provide principled tools for approximating a mixture of Gaussians with a smaller set while preserving key statistical properties. GHAP [47] is the most closely related work in the 3DGS literature: it formulates the geometry of a splat model as an unnormalized Gaussian mixture and performs blockwise reduction using an optimal-transport-based objective, followed by an appearance refinement stage. While effective, it relies on subsequent optimization to restore appearance quality.

Also related, though more narrowly, is Hierarchical 3DGS (H3DGS) [25], which is comparable to our method specifically at the *merge operator*: it fuses primitives into a bounding-volume hierarchy (BVH) used for initial downsampling and level-of-detail rendering. That our operators resemble each other is unsurprising, since moment matching is the natural way to approximate a two-Gaussian mixture by one Gaussian; any principled pairwise merge arrives at essentially the same operation. The methods differ elsewhere. H3DGS is an optimization-based pipeline that retrains the hierarchy, with the BVH chosen a priori as a downsampling and LOD structure, so it exposes no standalone merge cost or candidate-selection criterion—these are absorbed into training. Our framework is instead training-free and makes *which* pairs merge and *by what cost* the primary objects, leaving the operator interchangeable. The hierarchy we obtain is thus an emergent byproduct of cost-driven merging rather than a prescribed structure, and the topology can be swapped without touching the rest of the pipeline (Sec. 5).

In contrast, our method focuses on lightweight *post-hoc* simplification of trained splat sets. Rather than solving a transport problem, we employ an efficient local merging strategy that approximates pairs of Gaussians via moment matching, enabling training-free compaction that operates directly on existing models while preserving geometric and appearance consistency.

3 Methods

In this section, we describe the proposed splat simplification framework built around a progressive graph-based merging pipeline. We first organize the unstructured splat set into a sparse k-nearest-neighbor (KNN) merge graph. The

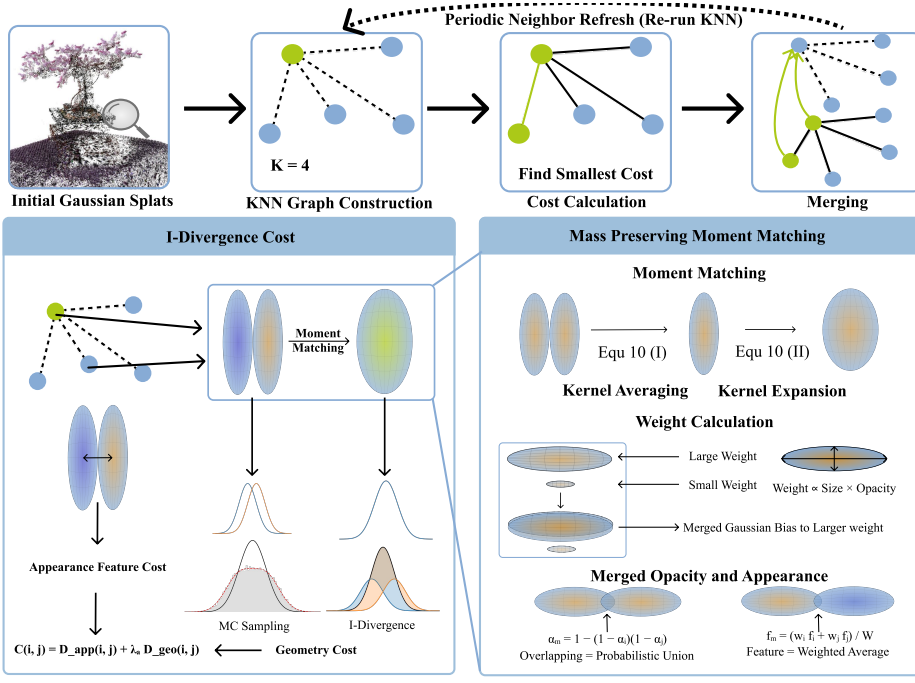

Fig. 2: NanoGS pipeline overview. *(Top)* Starting from an initial set of Gaussian splats, NanoGS constructs a sparse k-nearest-neighbor graph, evaluates a merge cost for each candidate edge, and collapses the lowest-cost disjoint pairs progressively. *(Bottom-left)* The merge cost combines an appearance term and a geometry term. The geometry cost measures the I-divergence between the original two-splat mixture and its single-Gaussian approximation. *(Bottom-right)* The Mass Preserved Moment Matching(MPMM) merge operator fuses a selected pair by computing mass-weighted moments, so the merged Gaussian is biased toward the larger primitive. Opacity is aggregated as a probabilistic union, and appearance features are blended as a weighted average.

key idea is to turn an unstructured set of N splats into a sparse *merge graph* and then repeatedly collapse low-cost edges to construct a coarse-to-fine hierarchy of splat representations. Our pipeline comprises four modules: **(i) sparse merge-graph construction** (Sec. 3.1), **(ii) merge-cost evaluation** (Sec. 3.2), **(iii) a merge operator** that fuses a selected pair into one splat (Sec. 3.3), and **(iv) progressive batched edge collapses** with periodic neighborhood refresh to reach any target compression level (Sec. 3.4).

3.1 Sparse Merge Graph Construction

Our merge objective assigns a cost to any pair of splats, but evaluating all $\binom{N}{2}$ pairs is prohibitive for large scenes. We therefore construct a sparse *merge graph* $G = (V, E)$ that restricts candidates to spatially local neighbors, reducing candidate evaluation from $\mathcal{O}(N^2)$ to $\mathcal{O}(kN)$. Figure 2 (top, second panel) illustrates

this step: each node connects to its k nearest neighbors (dashed edges), forming the candidate set that we will score and prune.

KNN neighborhood graph. Let each current splat $i \in V$ have center $\mu_i \in \mathbb{R}^3$. For each i, we query its k nearest neighbors in 3D and add undirected edges (i, j) to E. This yields $|V| = N$ and $|E| = \mathcal{O}(kN)$, which keeps candidate merges local while making edge scoring tractable.

What the graph represents. Each node corresponds to a (possibly already merged) splat, and each edge is a *candidate merge*. The rest of the method operates on this graph: we score edges, pick a set of disjoint edges, merge them in parallel, and update the graph.

3.2 Merge Cost

Intuitively (Fig. 2, bottom-left), $\mathcal{D}_{\text{merge}}$ is small when the two Gaussians overlap enough that a single Gaussian can cover them with little mass mismatch, and large when the pair is multi-modal (well-separated means), forcing the approximation to either miss probability mass or over-inflate. We define the cost of merging two splat sets i and j as a combination of a **geometric merge distortion** and an **appearance discrepancy**:

$$\mathcal{C}(i, j) = \mathcal{D}_{\text{geo}}(i, j) + \mathcal{D}_{\text{app}}(i, j), \tag{1}$$

Geometry discrepancy = merge distortion. For a candidate pair (i, j), the geometric term measures the distortion incurred when replacing the two-splat mixture by a single merged splat:

$$\mathcal{D}_{\text{geo}}(i, j) \doteq \mathcal{D}_{\text{merge}}(i, j \to m), \tag{2}$$

where m denotes the merged splat produced by our merge operator (Sec. 3.3).

Merge distortion (mixture→Gaussian). We regard each splat as an *unnormalized* Gaussian mass density

$$p_i(x) \doteq w_i \mathcal{N}(x; \mu_i, \Sigma_i), \qquad w_i \doteq (2\pi)^{3/2} \alpha_i \prod_k s_{i,k}, \qquad i \in \mathcal{S}, \tag{3}$$

where $\mathcal{S}$ denotes the current splat set and $\alpha_i \in [0, 1]$ is opacity. In Monte Carlo evaluation we draw S samples per candidate pair, with S a small fixed constant.

Let $p_{ij}(x) = p_i(x) + p_j(x)$ denote the unnormalized mixture, and define the *normalized* mixture distribution

$$\tilde{p}_{ij}(x) \doteq \frac{p_{ij}(x)}{W} = \pi \mathcal{N}(x; \mu_i, \Sigma_i) + (1 - \pi) \mathcal{N}(x; \mu_j, \Sigma_j), \qquad \pi \doteq \frac{w_i}{W}. \tag{4}$$

We approximate $\tilde{p}_{ij}$ by the merged Gaussian $q_m(x) = \mathcal{N}(x; \mu_m, \Sigma_m)$ produced by our VM moment matching, and define the merge distortion as the I-divergence

$$\mathcal{D}_{\text{merge}}(i, j \to m) \doteq \mathrm{I}(\tilde{p}_{ij} \,\|\, q_m) = \int \tilde{p}_{ij}(x) \log \frac{\tilde{p}_{ij}(x)}{q_m(x)} \, dx. \tag{5}$$

Because the mixture term $\log(\pi\mathcal{N}_i + (1-\pi)\mathcal{N}_j)$ has no closed form, we estimate (5) with a small fixed number of Monte Carlo samples per candidate pair.

Appearance difference. Each splat also carries an appearance feature vector $f_i \in \mathbb{R}^F$ (e.g., spherical harmonics coefficients). We define the appearance discrepancy as the squared ℓ_2 distance:

$$\mathcal{D}_{\text{app}}(i,j) \doteq \|f_i - f_j\|_2^2 . \tag{6}$$

3.3 Merging Operation

Mass-preserving moment matching (MPMM). Given a candidate pair of splats (i,j), we construct a merged splat m by matching *mass-weighted* spatial moments and applying a saturating opacity composition. The key design choice is the weighting: rather than using opacity alone, we weight each splat by its *mass* (the integral of its normalized Gaussian density) so that splats with larger spatial extents and/or higher opacities contribute proportionally more to the merged moments. This is especially important under aggressive compression, where the remaining primitives must preserve coverage without collapsing toward small, high-variance outliers. This update has a direct geometric interpretation (Fig. 2, bottom-right): it consists of *kernel averaging* (preserving typical local shape) plus *kernel expansion* (inflating uncertainty to cover between-mean dispersion). We therefore define:

Mass weights. Interpreting each splat as an unnormalized Gaussian mass density, the mass of splat i is

$$w_i \doteq (2\pi)^{3/2}\,\alpha_i \prod_k s_{i,k}, \qquad w_j \doteq (2\pi)^{3/2}\,\alpha_j \prod_k s_{j,k}, \qquad W \doteq w_i + w_j. \tag{7}$$

Intuitively, w approximates the total mass $\int p_i(x)\,dx$ of an unnormalized Gaussian, and thus increases with both *size* (scale) and *opacity*.

Mass-matched moments. The merged mean and appearance feature are mass-weighted averages:

$$\mu_m = \frac{w_i\mu_i + w_j\mu_j}{W}, \quad f_m \doteq \frac{w_i f_i + w_j f_j}{W}. \tag{8}$$

The merged covariance matches mass-weighted second moments:

$$\Sigma_m = \frac{w_i\big(\Sigma_i + (\mu_i-\mu_m)(\mu_i-\mu_m)^\top\big) + w_j\big(\Sigma_j + (\mu_j-\mu_m)(\mu_j-\mu_m)^\top\big)}{W}. \tag{9}$$

Equivalently, introducing $\mathcal{K} \doteq \{i,j\}$ and $W \doteq \sum_{k\in\mathcal{K}} w_k$, we obtain

$$\Sigma_m = \underbrace{\frac{1}{W}\sum_{k\in\mathcal{K}} w_k\,\Sigma_k}_{\text{(I) mass-weighted covariance average}} + \underbrace{\frac{1}{W}\sum_{k\in\mathcal{K}} w_k\,(\mu_k-\mu_m)(\mu_k-\mu_m)^\top}_{\text{(II) dispersion term}}. \tag{10}$$

The first term preserves the average *within-splat* shape, while the second term captures *between-mean dispersion*, expanding the merged Gaussian so that it covers the union of the two components.

Opacity and appearance aggregation. We compose opacities using the Porter–Duff *source-over* ("over") rule, equivalently multiplying transmittances $T = 1 - \alpha$

$$\alpha_m \doteq 1 - (1 - \alpha_i)(1 - \alpha_j) = \alpha_i + \alpha_j - \alpha_i \alpha_j. \tag{11}$$

We refer to this operator as **Mass Preserved Moment Matching (MPMM)**. Compared to classical moment matching and linear opacity conservation, MPMM biases merges toward *coverage* by accounting for splat volume, which empirically reduces holes and improves rendering stability under extreme compression.

3.4 Progressive Batched Merging

We perform simplification by repeatedly collapsing low-cost edges in the merge graph, analogous to edge-collapse procedures in mesh simplification.

Batched non-overlapping edge collapses. Given edge costs $\mathcal{C}(i, j)$ on E (Eq. 1), each pass selects a set of *non-overlapping* low-cost edges (a matching) and merges all selected pairs in parallel using MPMM (Sec. 3.3). Collapsing edges produces a new splat set and a coarser graph. Iterating this process yields a hierarchy of representations from fine to coarse, and we can stop once we reach a target number of splats or target compression ratio.

Local neighborhood refresh. Merges change local geometry and density, so a fixed initial KNN graph can miss newly adjacent merge candidates. We therefore periodically refresh candidate edges by re-running KNN queries around newly formed splats (and/or their local neighborhoods). This preserves scalability while allowing newly relevant pairs to become eligible candidates, improving merge quality versus a fully fixed edge set.

4 Experiments

We evaluate on four widely used 3DGS benchmarks spanning both synthetic and real captured scenes: NeRF-Synthetic [36] (8 scenes), Mip-NeRF 360 [5] (9 unbounded real-world scenes), Tanks and Temples [28] (2 large-scale real scenes), and Deep Blending [21] (2 challenging indoor scenes). This combination covers diverse geometry, appearance complexity, and camera trajectories.

Evaluation Metrics. We report Peak Signal-to-Noise Ratio (PSNR) and Structural Similarity Index Measure (SSIM) as primary visual fidelity metrics, and also report rendering speed (FPS) in Table 1. All metrics are computed on held-out test views following each benchmark's standard evaluation protocol.

Comparison Protocol. Table 1 compares against LightGS [15], PUP3DGS [20], and GHAP [47] at three representative budgets, $\rho \in \{0.1, 0.01, 0.001\}$: $\rho = 0.1$ follows the common GHAP setting, while $\rho = 0.01$ and $\rho = 0.001$ reflect more aggressive LOD-style compression. Since our objective is *training-free* post-hoc compaction, we do not perform any post-compression fine-tuning; for methods typically paired with a fine-tuning stage (e.g., LightGS and PUP3DGS), we evaluate only their pruning/selection stage to isolate compaction quality.

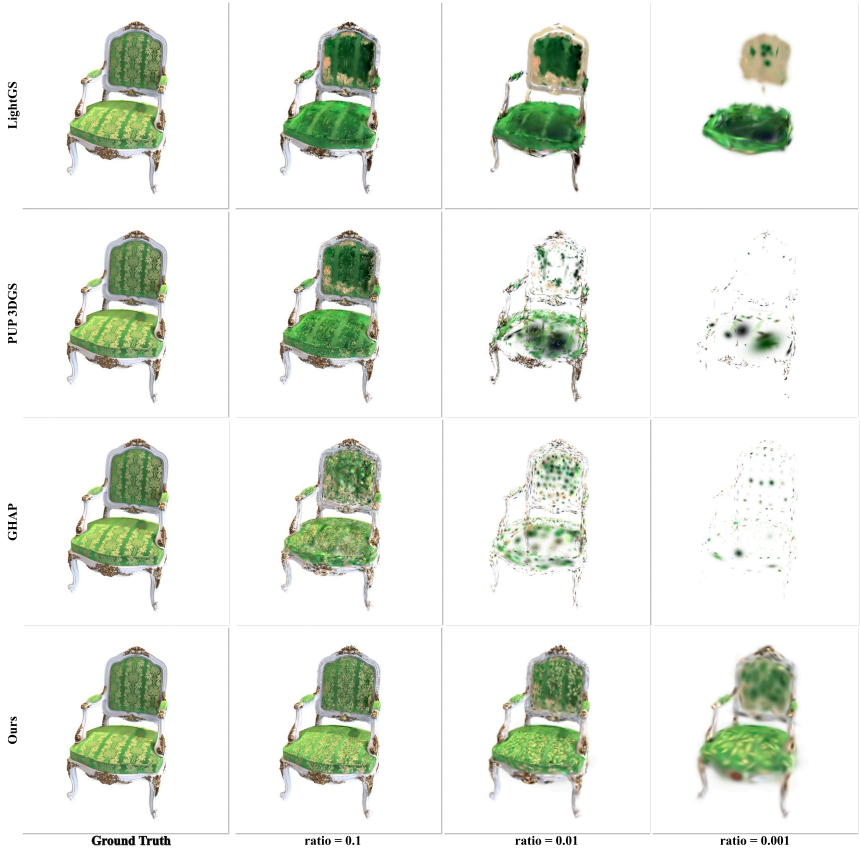

Fig. 3: Qualitative comparison on chair from NeRF Synthetic dataset. We show two representative test views. Columns correspond to the compaction ratio $\rho \in \{0.1, 0.01, 0.001\}$ (leftmost: ground truth), and rows compare [15, 20, 47], and Ours.

Implementation Details. Our method operates directly on pretrained 3DGS assets and preserves the original 3DGS representation, so outputs remain compatible with standard Gaussian splatting renderers. Monte Carlo merge-cost estimation uses S=1 sample by default (S=128 yields no noticeable quality gain); we set the KNN graph size to K=16, merge $N/2$ disjoint pairs per pass, and refresh the graph each pass. Unless otherwise noted, all experiments use a single NVIDIA RTX 4090 GPU with CUDA 12.8, with FPS measured on the same machine, and we report final results at each target ratio without retraining.

4.1 Comparison with the State of the Art

Qualitatively (Fig. 4, Fig. 3), all methods are broadly faithful at moderate compression (ρ=0.1). As the budget decreases (ρ=0.01, 0.001), baselines collapse into sparse high-intensity splats and lose coherent surfaces, consistent with unstable merges and floater-dominated candidates, whereas our filtering and mass-

Fig. 4: Qualitative comparison on Mip-NeRF 360 dataset. We show two representative test views. Columns correspond to the compaction ratio $\rho \in \{0.1, 0.01, 0.001\}$ (leftmost: ground truth), and rows compare [15, 20, 47], and Ours.

Dataset (PSNR/SSIM/FPS) Full	Method	$\rho = 0.1$			$\rho = 0.01$			$\rho = 0.001$		
		PSNR	SSIM	FPS	PSNR	SSIM	FPS	PSNR	SSIM	FPS
NeRF Synth [36] 33.66/0.970/854.04	LightGS [15]	21.25	0.875	2204.97	15.76	0.807	2571.07	12.64	0.796	2623.73
	PUP3DGS [20]	20.24	0.860	2341.34	13.26	0.786	2676.07	11.31	0.792	2838.60
	GHAP [47]	20.09	0.838	2486.94	12.81	0.776	2679.19	11.07	0.791	2916.30
	Ours	25.81	0.910	2450.17	22.28	0.858	2598.03	19.04	0.822	2641.46
Mips-360 [4] 27.46/0.815/182.43	LightGS [15]	19.38	0.588	727.19	14.39	0.389	1525.92	11.97	0.278	2070.79
	PUP3DGS [20]	15.59	0.513	996.41	10.19	0.161	2407.68	8.81	0.056	2650.41
	GHAP [47]	17.35	0.444	1224.46	10.62	0.174	2557.02	8.52	0.035	2622.83
	Ours	21.97	0.582	1015.20	19.39	0.470	2005.76	17.20	0.430	2193.87
T&T [28] 23.63/0.847/299.07	LightGS [15]	17.50	0.642	1108.22	12.29	0.441	1995.98	9.36	0.341	2418.99
	PUP3DGS [20]	12.71	0.564	1283.38	8.22	0.266	2433.64	6.78	0.162	2593.01
	GHAP [47]	15.34	0.487	1665.64	8.43	0.220	2659.11	5.33	0.026	2670.40
	Ours	17.94	0.626	1429.88	15.29	0.501	2325.87	13.54	0.457	2487.31
D&B [21] 29.56/0.903/233.25	LightGS [15]	24.28	0.816	891.18	18.28	0.712	1878.64	13.41	0.616	2295.95
	PUP3DGS [20]	19.65	0.755	1325.03	8.83	0.282	2478.12	7.06	0.073	2672.21
	GHAP [47]	21.75	0.739	1375.59	11.36	0.435	2583.11	7.06	0.061	2627.05
	Ours	26.29	0.839	1202.23	23.12	0.780	2132.31	19.42	0.739	2321.92

Table 1: Quantitative benchmarks across datasets and sampling ratios ρ. Full-quality references (PSNR/SSIM/FPS at $\rho = 1.0$) are listed in the Dataset column. For $\rho \in \{0.1, 0.01, 0.001\}$, color indicates rank within each dataset/metric column: red =1st, pink =2nd, yellow =3rd.

preserving moment matching maintain contiguous coverage and degrade gracefully. Additional comparisons are in the appendix.

Quantitatively (Table 1), our method achieves the best PSNR at every budget across all four benchmarks, and its advantage widens as compression becomes more aggressive: averaged over datasets, PSNR improves over the best baseline by $+$**2.40 dB** at ρ=0.1, $+$**4.84 dB** at ρ=0.01, and $+$**5.46 dB** at ρ=0.001. Gains are largest on NeRF-Synthetic (up to $+$**6.52 dB**) and grow consistently with compression on the real-scene benchmarks (Mip-NeRF 360, Tanks & Temples, Deep Blending), where they reflect fewer missing-geometry and floater failures. Overall, baselines degrade rapidly as ρ decreases while our merge-graph pipeline retains substantially higher fidelity, most visibly in the most aggressive regimes.

4.2 Runtime and Storage

NanoGS is lightweight enough to run on CPU. Table 2 reports wall-clock time to compact to ρ=0.1: even the largest outdoor Mip-NeRF 360 scenes finish in under 90 s on CPU and under 20 s on a single GPU, with smaller scenes completing in seconds. Because NanoGS preserves the standard 3DGS representation, it composes with orthogonal bit-level compression: Table 3 chains our compaction with Self-Organized Gaussians (SOG) [37], yielding a further $\sim$6.3$\times$ mean storage reduction on top of compaction across all 60 models, for a combined reduction of several orders of magnitude at the most aggressive ratios.

4.3 Ablation Study

Table 4 evaluates four variants: **w/o KNN graph** replaces KNN-graph candidate selection with Octree-based neighborhood selection, **w/o filtering**

Table 2: Wall-clock runtime to compact to ρ=0.1, CPU vs. single GPU.

Dataset	CPU (s)	GPU (s)	Orig. (MB)
M360 Out	88.10	17.14	1122
M360 In	25.27	5.07	353
T & T	19.41	4.12	255
D & B	51.69	9.75	663
NeRF-Syn	4.57	1.63	67

Table 3: Combined storage (MB): NanoGS → SOG [37]. Cells: compacted → compressed (× from SOG).

Dataset	r=0.1	r=0.01	r=0.001
M360 Out	112.23→16.07	11.22→1.61	1.12→0.18
M360 In	34.29→4.80	3.43→0.48	0.34→0.06
T & T	25.48→3.68	2.55→0.39	0.26→0.04
D & B	66.27→9.95	6.63→1.02	0.66→0.11
NeRF-Syn	6.69→1.11	0.67→0.11	0.07→0.01
Mean	16.83→2.45 (6.35×), 60 models		

disables the initial opacity-based pruning, **w/o I-divergence** replaces the I-divergence geometric term with a fast Mean Squared Error (MSE) surrogate, and **Full** uses the complete formulation.

Dataset	Method	$\rho = 0.1$		$\rho = 0.01$		$\rho = 0.001$	
		PSNR	SSIM	PSNR	SSIM	PSNR	SSIM
NeRF	w/o KNN graph	25.53	0.905	20.74	0.839	15.85	0.800
	w/o filtering	24.99	0.904	22.01	**0.858**	18.87	**0.822**
	w/o I-divergence	25.65	0.908	22.15	0.854	18.80	0.819
	Full	**25.81**	**0.910**	**22.28**	**0.858**	**19.04**	**0.822**
M360	w/o KNN graph	21.53	0.537	17.82	0.444	14.11	0.382
	w/o filtering	18.24	0.504	15.47	0.421	14.67	0.401
	w/o I-divergence	21.61	0.566	19.19	0.467	16.98	0.426
	Full	**21.97**	**0.582**	**19.39**	**0.470**	**17.20**	**0.430**

Table 4: Ablation study across datasets and compaction ratios. **w/o KNN graph** replaces the KNN-graph candidate selection with the Octree-based one; **w/o filtering** disables opacity-based filtering; **w/o I-divergence** replaces the principled I-divergence geometric cost with an Mean Squared Error surrogate; **Full** uses the complete proposed formulation. Bold indicates the best result within each dataset/metric column.

Candidate selection. Replacing the KNN-graph candidate construction with Octree-based selection leads to a clear quality drop, especially at aggressive budgets: on Mip-NeRF 360 at ρ=0.001, PSNR drops from **17.20** (Full) to **14.11**, and on NeRF-Synthetic from **19.04** to **15.85**, indicating that KNN-graph locality provides more reliable merge candidates than the Octree alternative.

Opacity filtering. Removing filtering consistently hurts performance, particularly on Mip-NeRF 360 (**17.20**→**14.67** at ρ=0.001), while NeRF-Synthetic is less sensitive, suggesting that filtering suppresses the low-opacity "floaters" common in real-world reconstructions.

Merge cost. Replacing the I-divergence geometric distortion with the fast surrogate consistently degrades quality across both datasets and all ratios, indicating that the principled I-divergence formulation is a more faithful geometric fidelity measure than the MSE approximation.

In sum, KNN-graph candidate selection improves merge reliability, filtering removes noisy floaters early, and the I-divergence cost adds geometry-aware guidance. Notably, the first and third of these are stage swaps—topology and cost objective—whose effect on quality directly demonstrates that the stages are independently consequential.

5 Conclusion and Discussion

We presented **NanoGS**, a training-free, CPU-friendly framework for Gaussian Splat simplification that operates directly on pretrained 3DGS models without calibrated images or scene-specific optimization. By formulating simplification as progressive local pairwise merging over a sparse k-nearest-neighbor graph, NanoGS avoids the GPU-intensive retraining pipelines that limit existing compaction methods, and our GPU implementation of the graph-construction and edge-collapse stages further reduces wall-clock time by roughly 5× (Table 2). Across four standard benchmarks and varying compression budgets, it consistently outperforms state-of-the-art compaction methods, with particularly stable degradation at extreme ratios where prior methods collapse into floater-dominated artifacts. Crucially, NanoGS is not merely a merging algorithm but a *structural system*: rather than forcing merges through a rigid spatial grid, it decouples candidate selection, merge cost, and the merge operator into independent, interchangeable stages, so that strong compaction follows from the structure rather than any single component—and we regard this framework, which our ablations exercise along the topology and cost axes, as the central contribution. Each stage is also a substitution point that opens future directions: the analytic cost could be replaced by a learned neural criterion capturing perceptually or semantically important splats; the operator could extend beyond pairwise fusion to many-to-one merging, or, by *inverting* from compression to generation, become a super-resolution procedure that splits primitives to add detail. Finally, extending the framework from static to dynamic 3DGS representations is a natural and practically important next step.

6 Acknowledgment

The project or effort depicted was sponsored by the U.S. Army Combat Capabilities Development Command under contract number W912CG-24-D-0001 and a research agreement with Coolant Climate, Inc. The content of the information does not necessarily reflect the position or the policy of the Government, and no official endorsement should be inferred.

References

1. Ali, M.S., Zhang, C., Cagnazzo, M., Valenzise, G., Tartaglione, E., Bae, S.H.: Compression in 3d gaussian splatting: A survey of methods, trends, and future directions. arXiv preprint arXiv:2502.19457 (2025)
2. Bagdasarian, M.T., Knoll, P., Li, Y., Barthel, F., Hilsmann, A., Eisert, P., Morgenstern, W.: 3dgs.zip: A survey on 3d gaussian splatting compression methods. Computer Graphics Forum (2025)
3. Banerjee, A., Merugu, S., Dhillon, I.S., Ghosh, J.: Clustering with bregman divergences. Journal of machine learning research **6**(Oct), 1705–1749 (2005)
4. Barron, J.T., Mildenhall, B., Tancik, M., Hedman, P., Martin-Brualla, R., Srinivasan, P.P.: Mip-nerf: A multiscale representation for anti-aliasing neural radiance fields. In: CVPR (2022)
5. Barron, J.T., Mildenhall, B., Verbin, D., Srinivasan, P.P., Hedman, P.: Mip-nerf 360: Unbounded anti-aliased neural radiance fields. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. pp. 5470–5479 (2022)
6. Barron, J.T., Mildenhall, B., Verbin, D., Srinivasan, P.P., Hedman, P.: Zip-nerf: Anti-aliased grid-based neural radiance fields. In: ICCV (2023)
7. Bregman, L.M.: The relaxation method of finding the common point of convex sets and its application to the solution of problems in convex programming. USSR computational mathematics and mathematical physics **7**(3), 200–217 (1967)
8. Chen, Y., Wu, Q., Lin, W., Harandi, M., Cai, J.: Hac: Hash-grid assisted context for 3d gaussian splatting compression. In: European Conference on Computer Vision. Springer (2024)
9. Chen, Y., Wu, Q., Lin, W., Harandi, M., Cai, J.: Hac++: Towards 100x compression of 3d gaussian splatting. arXiv preprint arXiv:2501.12255 (2025)
10. Chung, J., Lee, S., Nam, H., Lee, J., Lee, K.M.: Luciddreamer: Domain-free generation of 3d gaussian splatting scenes. arXiv preprint arXiv:2311.13384 (2023)
11. Deng, T., Chen, Y., Zhang, L., Yang, J., Yuan, S., Liu, J., Wang, D., Wang, H., Chen, W.: Compact 3d gaussian splatting for dense visual slam. arXiv preprint arXiv:2403.11247 (2024)
12. Dhillon, I.S., Mallela, S., Modha, D.S.: Information-theoretic co-clustering. In: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 89–98 (2003)
13. Du, X., Wang, Y., Zhan, K., Yu, X.: Mobile-gs: Real-time gaussian splatting for mobile devices (2026), `https://arxiv.org/abs/2603.11531`
14. electronicarts: mesh2splat: Fast mesh to 3d gaussian splat conversion (2025), `https://github.com/electronicarts/mesh2splat`
15. Fan, Z., Wang, K., Wen, K., Zhu, Z., Xu, D., Wang, Z., et al.: Lightgaussian: Unbounded 3d gaussian compression with 15x reduction and 200+ fps. Advances in neural information processing systems **37**, 140138–140158 (2024)
16. Franke, L., Fink, L., Stamminger, M.: Vr-splatting: Foveated radiance field rendering via 3d gaussian splatting and neural points. Proc. ACM Comput. Graph. Interact. Tech. (2025)
17. Girish, S., Gupta, K., Shrivastava, A.: Eagles: Efficient accelerated 3d gaussians with lightweight encodings. In: European Conference on Computer Vision. pp. 54–71. Springer (2024)
18. Guédon, A., Lepetit, V.: Gaussian frosting: Editable complex radiance fields with real-time rendering. ECCV (2024)

19. Guo, W., Fang, G., Yang, S., Wang, B.: Pocketgs: On-device training of 3d gaussian splatting for high perceptual modeling. arXiv preprint arXiv:2601.17354 (2026)
20. Hanson, A., Tu, A., Singla, V., Jayawardhana, M., Zwicker, M., Goldstein, T.: Pup 3d-gs: Principled uncertainty pruning for 3d gaussian splatting. In: Proceedings of the Computer Vision and Pattern Recognition Conference. pp. 5949–5958 (2025)
21. Hedman, P., Philip, J., Price, T., Frahm, J.M., Drettakis, G., Brostow, G.: Deep blending for free-viewpoint image-based rendering. ACM Transactions on Graphics (ToG) **37**(6), 1–15 (2018)
22. Jacobson, D., Brail, G., Woods, D.: APIs: A strategy guide. " O'Reilly Media, Inc." (2012)
23. Jiang, Y., Yu, C., Xie, T., Li, X., Feng, Y., Wang, H., Li, M., Lau, H., Gao, F., Yang, Y., Jiang, C.: Vr-gs: A physical dynamics-aware interactive gaussian splatting system in virtual reality. In: SIGGRAPH (2024)
24. Kerbl, B., Kopanas, G., Leimkühler, T., Drettakis, G.: 3d gaussian splatting for real-time radiance field rendering. In: SIGGRAPH (2023)
25. Kerbl, B., Meuleman, A., Kopanas, G., Wimmer, M., Lanvin, A., Drettakis, G.: A hierarchical 3d gaussian representation for real-time rendering of very large datasets. ACM Transactions On Graphics (TOG) **43**(4), 1–15 (2024)
26. Khronos 3D Commerce Working Group: Real-time Asset Creation Guidelines. `https://github.com/KhronosGroup/3DC-Asset-Creation` (2025), accessed: 2026-05-05
27. Khronos 3D Formats Working Group: glTF Extension: KHR_gaussian_splatting (2024), accessed: 2026-05-05
28. Knapitsch, A., Park, J., Zhou, Q.Y., Koltun, V.: Tanks and temples: Benchmarking large-scale scene reconstruction. ACM Transactions on Graphics (ToG) **36**(4), 1–13 (2017)
29. Lee, J.C., Ko, J.H., Park, E.: Optimized minimal 3d gaussian splatting (2025), `https://arxiv.org/abs/2503.16924`
30. Lee, J.C., Rho, D., Sun, X., Ko, J.H., Park, E.: Compact 3d gaussian representation for radiance field. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 21719–21728 (2024)
31. Liu, L., Chen, Z., Jiang, W., Wang, W., Xu, D.: Hemgs: A hybrid entropy model for 3d gaussian splatting data compression. arXiv preprint arXiv:2411.18473 (2024)
32. Liu, R., Gao, Z., Planche, B., Chen, M., Nguyen, V.N., Zheng, M., Choudhuri, A., Chen, T., Wang, Y., Feng, A., et al.: Universal beta splatting. arXiv preprint arXiv:2510.03312 (2025)
33. Liu, R., Sun, D., Chen, M., Wang, Y., Feng, A.: Deformable beta splatting. In: Proceedings of the Special Interest Group on Computer Graphics and Interactive Techniques Conference Conference Papers. pp. 1–11 (2025)
34. Mallick, S.S., Goel, R., Kerbl, B., Steinberger, M., Carrasco, F.V., De La Torre, F.: Taming 3dgs: High-quality radiance fields with limited resources. In: SIGGRAPH Asia 2024 Conference Papers. pp. 1–11 (2024)
35. Mescheder, L., Dong, W., Li, S., Bai, X., Santos, M., Hu, P., Lecouat, B., Zhen, M., Delaunoy, A., Fang, T., Tsin, Y., Richter, S.R., Koltun, V.: Sharp monocular view synthesis in less than a second. arXiv preprint arXiv:2512.10685 (2025)
36. Mildenhall, B., Srinivasan, P.P., Tancik, M., Barron, J.T., Ramamoorthi, R., Ng, R.: Nerf: Representing scenes as neural radiance fields for view synthesis. In: ECCV (2020)
37. Morgenstern, W., Barthel, F., Hilsmann, A., Eisert, P.: Compact 3d scene representation via self-organizing gaussian grids. In: European conference on computer vision. pp. 18–34. Springer (2024)

38. Müller, T., Evans, A., Schied, C., Keller, A.: Instant neural graphics primitives with a multiresolution hash encoding. In: SIGGRAPH (2022)
39. Niedermayr, S., Stumpfegger, J., Westermann, R.: 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)
40. Papantonakis, P., Kopanas, G., Kerbl, B., Lanvin, A., Drettakis, G.: Reducing the memory footprint of 3d gaussian splatting. Proceedings of the ACM on Computer Graphics and Interactive Techniques **7**(1) (May 2024)
41. PlayCanvas: Supersplat: 3d gaussian splat editor (2024), `https://github.com/playcanvas/supersplat`
42. Qin, M., Li, W., Zhou, J., Wang, H., Pfister, H.: Langsplat: 3d language gaussian splatting. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 20051–20060 (2024)
43. Runnalls, A.R.: Kullback-leibler approach to gaussian mixture reduction. IEEE Transactions on Aerospace and Electronic Systems **43**(3), 989–999 (2007)
44. Scott, W.R., Roth, G., Rivest, J.F.: View planning for automated three-dimensional object reconstruction and inspection. ACM Computing Surveys **35**(1), 64–96 (2003)
45. Tang, J., Ren, J., Zhou, H., Liu, Z., Zeng, G.: Dreamgaussian: Generative gaussian splatting for efficient 3d content creation. In: ICLR (2024)
46. Wang, C., Wolski, K., Kerbl, B., Serrano, A., Bemana, M., Seidel, H.P., Myszkowski, K., Leimkühler, T.: Cinematic gaussians: Real-time hdr radiance fields with depth of field. In: Computer Graphics Forum. p. e15214. Wiley Online Library (2024)
47. Wang, T., Li, M., Zeng, G., Meng, C., Zhang, Q.: Gaussian herding across pens: An optimal transport perspective on global gaussian reduction for 3dgs. In: Advances in Neural Information Processing Systems (NeurIPS 2025) (2025)
48. Wang, Y., Li, Z., Guo, L., Yang, W., Kot, A., Wen, B.: Contextgs: Compact 3d gaussian splatting with anchor level context model. Advances in neural information processing systems **37**, 51532–51551 (2024)
49. Wei, J., Leutenegger, S., Schaefer, S.: Gsfix3d: Diffusion-guided repair of novel views in gaussian splatting. arXiv preprint arXiv:2508.14717 (2025)
50. Wu, G., Yi, T., Fang, J., Xie, L., Zhang, X., Wei, W., Liu, W., Tian, Q., Wang, X.: 4d gaussian splatting for real-time dynamic scene rendering. In: CVPR (2024)
51. Xie, H., Chen, Z., Hong, F., Liu, Z.: Generative gaussian splatting for unbounded 3d city generation. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (2025)
52. Xiong, B., Liu, R., Xu, K., Chen, M., Feng, A.: Splat feature solver. In: ICLR (2026)
53. Yang, Z., Yang, H., Pan, Z., Zhang, L.: Real-time photorealistic dynamic scene representation and rendering with 4d gaussian splatting. In: ICLR (2024)
54. Yang, Z., Gao, X., Zhou, W., Jiao, S., Zhang, Y., Jin, X.: Deformable 3d gaussians for high-fidelity monocular dynamic scene reconstruction. In: CVPR (2024)
55. Yi, T., Fang, J., Wang, J., Wu, G., Xie, L., Zhang, X., Liu, W., Tian, Q., Wang, X.: Gaussiandreamer: Fast generation from text to 3d gaussians by bridging 2d and 3d diffusion models. In: CVPR (2024)
56. Zhang, Y., Jia, W., Niu, W., Yin, M.: Gaussianspa: An" optimizing-sparsifying" simplification framework for compact and high-quality 3d gaussian splatting. In: CVPR (2025)

57. Zhang, Z., Song, T., Lee, Y., Yang, L., Peng, C., Chellappa, R., Fan, D.: Lp-3dgs: Learning to prune 3d gaussian splatting. Advances in Neural Information Processing Systems **37**, 122434–122457 (2024)
58. Zhou, J., Fan, L., Chen, X., Huang, L., Liu, S., Li, H.: Gaussianpainter: Painting point cloud into 3d gaussians with normal guidance (2024), https://arxiv.org/abs/2412.17715
59. Zhou, S., Chang, H., Jiang, S., Fan, Z., Zhu, Z., Xu, D., Chari, P., You, S., Wang, Z., Kadambi, A.: Feature 3dgs: Supercharging 3d gaussian splatting to enable distilled feature fields. In: CVPR (2024)