TDAP-GS: SCALE-ADAPTIVE TILE-BASED DENSITY ACCUMULATION AND PRUNING FOR HIGHLY EFFICIENT 3D GAUSSIAN SPLATTING

Anonymous authors
Paper under double-blind review

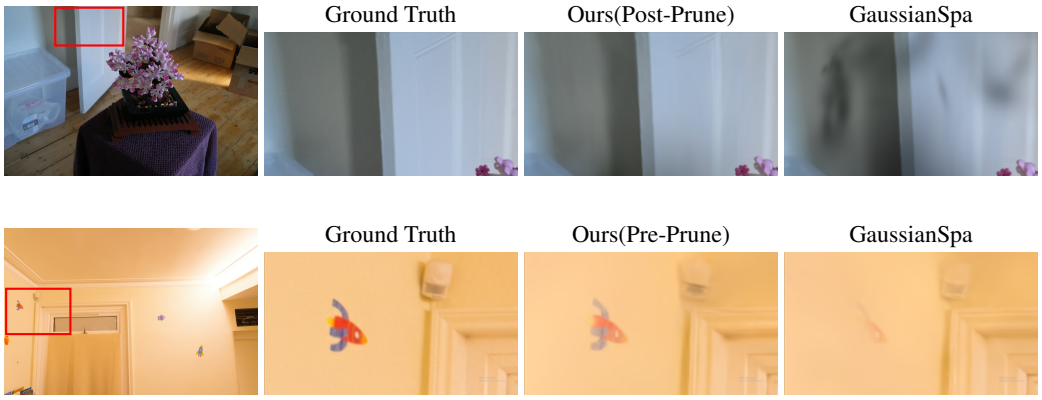


Figure 1: **We propose TDAP-GS**, with compare to exists state-of-the-art method GaussianSpa. Our model can keep details at the background and accurately prune the floaters.

ABSTRACT

3D Gaussian Splatting (3DGS) achieves high-quality novel-view synthesis with real-time rendering, but densification can produce redundant Gaussians that increase memory use and rendering cost. Existing compression methods may retain primitives in deeply occluded regions or high-frequency floaters, while hard pruning can degrade visual quality and require extensive post-pruning refinement. We propose **Tile-Based Density Accumulation and Pruning** (TDAP), which uses the tile-based rendering pipeline to estimate a scale-adaptive density for each Gaussian. TDAP then applies quantile-guided opacity modulation to progressively remove low-density candidates while preserving Gaussians in dense scene regions. It requires neither an auxiliary loss nor post-pruning optimization. Experiments show that TDAP reduces the number of Gaussians and rendering cost while maintaining, and in some cases improving, rendering quality.

1 INTRODUCTION

3D Gaussian Splatting (3DGS) (Kerbl et al., 2023) represents a scene as a set of anisotropic 3D Gaussian primitives. Each primitive has a position, covariance, opacity, and view-dependent appearance. A differentiable rasterizer projects these primitives into the image plane, sorts their contributions within screen-space tiles, and composites them to render novel views. This explicit representation enables high-quality reconstruction and interactive rendering, making 3DGS a useful foundation for applications such as dynamic scene modeling, large-scale reconstruction, and mixed reality. The efficiency of rendering, however, does not eliminate the cost of storing and optimizing the representation. During training, adaptive density control clones or splits Gaussians in regions associated with large view-space positional gradients. This process improves coverage of insufficiently represented regions, but its decisions are driven by local optimization signals rather than by

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

a direct measure of whether a primitive remains necessary to render the scene. Consequently, the number of Gaussians can grow substantially, increasing parameter storage, optimizer-state memory, and the work required for projection, sorting, and compositing. Reducing the final model size alone does not address these costs if redundant primitives persist throughout training.

Pruning during training presents a different difficulty. A Gaussian may appear redundant under an instantaneous criterion yet still support the current reconstruction. Removing many such primitives in a single step can create visible errors, after which the remaining representation must adapt through further optimization. The central challenge is therefore to identify primitives with limited rendering utility early enough to reduce training cost, while changing the representation gradually enough to preserve image quality. This calls for both a redundancy measure that can be evaluated during rendering and a pruning mechanism that avoids abrupt removal.

We propose **Tile-based Density Accumulation and Pruning** (TDAP), an in-training method built around these two requirements. First, TDAP derives a depth-ordered, scale-weighted density measure from the tile-level information used by the 3DGS rasterizer. By accounting for where Gaussians project, how they overlap within tiles, and their relative depth and scale, the measure ranks primitives according to their local configuration in the rendered views. Gaussians repeatedly associated with low-priority configurations become candidates for removal. The measure is computed from rendering information rather than from an auxiliary learned network or an additional supervision signal. Second, TDAP uses *quantile-guided opacity modulation* to act on these rankings. Instead of deleting all candidates immediately, it assigns opacity adjustments according to density quantiles. Candidate Gaussians are progressively attenuated, allowing their rendering contributions to decline before removal; Gaussians assigned to the retained core receive the corresponding opacity treatment defined by the modulation rule. Because these adjustments are made during training, the remaining Gaussians can adapt as the representation changes. TDAP introduces no additional loss term and retains the underlying 3DGS reconstruction objective. We evaluate TDAP along four complementary dimensions: the number of Gaussians retained, GPU memory use during training, rendering cost, and novel-view reconstruction quality. In particular, measuring training memory distinguishes an in-training intervention from methods that only compress the final model. We also examine the contributions of the tile-based density measure and opacity modulation separately to determine whether candidate selection and gradual removal each improve the memory–quality trade-off.

Our main contributions are:

- **A tile-based density measure.** We introduce a depth-ordered, scale-weighted statistic derived from the rasterizer's tile-level information to rank Gaussian primitives for in-training pruning.
- **Quantile-guided opacity modulation.** We replace immediate removal of selected candidates with progressive opacity adjustment, allowing the representation to adapt before pruning.
- **An evaluation of training-time efficiency.** We assess the effects of TDAP on primitive count, GPU memory, rendering cost, and novel-view quality, including the separate effects of its density measure and modulation strategy.

2 Related Work

2.1 Post-Refine Methods

LightGaussian(Fan et al. (2024)) spearheaded efforts to minimize final file sizes by identifying that Spherical Harmonics (SH) coefficients account for the vast majority of storage space. By employing SH knowledge distillation(Hinton et al. (2015), Sanh et al. (2020), Wang & Yoon (2022)) alongside Vector Quantization(Niedermayr et al. (2024)), it significantly reduces the storage footprint while preserving visual quality, and further introduces a geometry-based significance score to cull redundant primitives after training. Taking a more mathematically rigorous approach to primitive elimination, **PUP-3DGS**(Hanson et al. (2025)) frames Gaussian pruning as a sensitivity analysis problem. Rather than relying on simple geometric heuristics, PUP-3DGS approximates the Fisher Information Matrix using the second-order derivatives of the L_2 rendering loss. This formulation quantifies the exact structural destruction caused by removing any specific primitive, allowing the algorithm to surgically eliminate Gaussians that have the least impact on the final render. Furthermore, this work

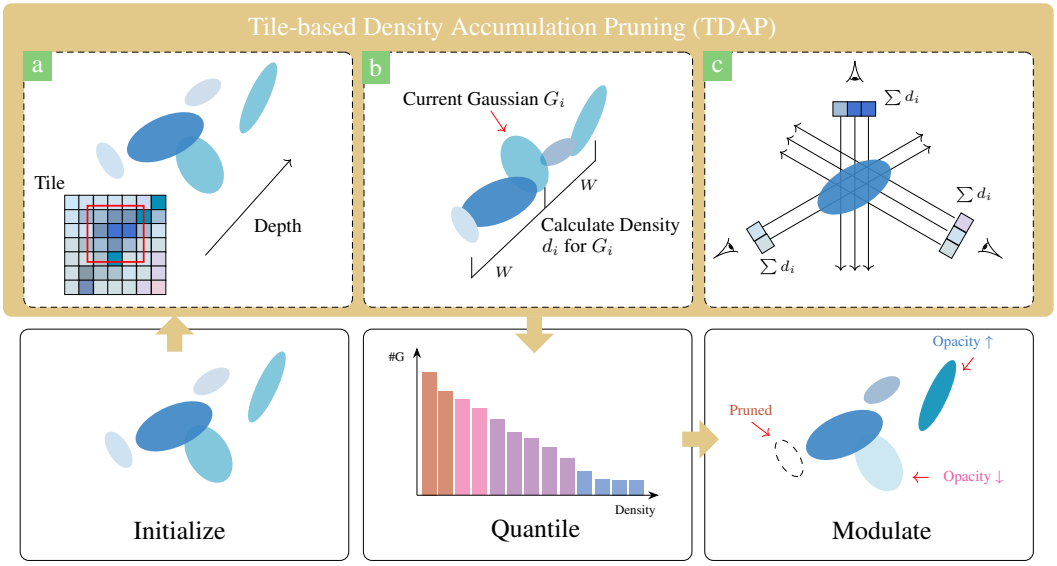


Figure 2: Overview of Tile-based Density Accumulation and Pruning (TDAP-GS). **(a)** The standard 3DGS rasterizer assigns Gaussians to image tiles and sorts them by depth. **(b)** TDAP uses neighboring entries in each sorted tile list to compute local density scores. **(c)** Scores are accumulated across training views over a 500-iteration interval. At each pruning event, density quantiles determine which Gaussians are removed, which receive an opacity penalty, and which receive an opacity boost. TDAP adds no auxiliary loss.

systematically demonstrates that a staged pruning schedule yields significantly higher fidelity than attempting a massive single-step reduction.

While post-refine methods reduce disk storage, they do not alleviate the massive VRAM overhead encountered during the training and densification phase. Consequently, recent literature (Chen & Wang (2026), Bao et al. (2024)) has shifted toward in-training regularization.

2.2 In-Training Densification Methods

To constrain primitive proliferation and artifact generation, recent approaches have introduced explicit regularization and spatial consolidation mechanisms. For instance, **Taming-3DGS**(Mallick et al. (2024)) incorporates auxiliary regularization terms directly into the optimization objective. While this effectively penalizes redundancy and promotes a cleaner spatial distribution, relying on auxiliary loss functions demands meticulous hyperparameter tuning to balance the primary photometric loss. Suboptimal weighting frequently results in either over-smoothing or the degradation of high-frequency texture details. Addressing this unconstrained expansion bottleneck from a structural perspective, **Mini-Splatting**(Fang & Wang (2024), Fang & Wang (2026)) incorporates explicit geometric constraints coupled with a primitive aggregation mechanism. By systematically consolidating over-split Gaussians within locally dense regions, it effectively limits unbounded population growth and improves overall scene compactness without relying solely on heuristic opacity thresholds (Deng et al. (2026), Lee et al. (2024)).

More recently, methodologies have shifted toward utility-based filtering and mathematically constrained sparsification frameworks. **MaskGaussian**(Liu et al. (2025)) tackles the proliferation of transparent floaters by integrating opacity- and mask-guided constraints(Guédon & Lepetit (2023), Ye et al. (2024)) into the training pipeline. By explicitly evaluating the contribution of individual primitives to the rendered view, it dynamically suppresses redundant geometries based on their active rendering utility, thereby maintaining a compact representation. Concurrently, **GaussianSpa**(Zhang et al. (2025)) formulates the simplification objective as a constrained optimization problem with a sparsity constraint on Gaussian opacity. Rather than utilizing abrupt heuristic pruning, it introduces absolute sparse auxiliary variables coupled with an augmented Lagrangian penalty. This mechanism coerces redundant Gaussians to gradually fade into transparency prior to removal, successfully pre-

Table 1: Quantitative comparison of TDAP-GS with pruning baselines. The upper section reports methods evaluated with an 80% target reduction in Gaussian count; an asterisk (*) indicates training extended to 40K iterations for refinement. The lower section reports methods whose final Gaussian counts are approximately 80% below the unpruned baseline. Arrows indicate whether higher (↑) or lower (↓) values are better. The best and second-best results are shown in **bold** and underline.

Method	Mip-Nerf360			Deep Blending			Tank and Temples		
	SSIM↑	PSNR↑	LPIPS↓	SSIM↑	PSNR↑	LPIPS↓	SSIM↑	PSNR↑	LPIPS↓
LightGaussian*	0.804	27.206	0.233	0.896	29.206	0.260	0.835	23.421	**0.198**
PUP-3DGS	0.809	27.229	0.222	**0.903**	**29.311**	**0.246**	0.825	22.786	**0.198**
Ours(Post-Refine)	**0.819**	**27.459**	**0.212**	0.893	29.095	0.268	**0.836**	**23.562**	0.202
Mini-Splatting	0.822	27.296	0.220	0.907	**30.014**	0.253	0.836	23.157	0.203
Taming-3DGS	0.803	27.429	0.248	0.908	30.000	0.252	0.838	**23.958**	0.206
MaskGaussian	0.811	27.369	0.227	0.909	29.793	0.248	0.848	23.676	0.183
GaussianSpa	0.822	27.461	0.219	**0.914**	30.000	**0.238**	**0.853**	23.506	**0.161**
Ours(Pre-Prune)	0.821	27.417	0.220	0.904	29.615	0.257	0.831	23.355	0.213
Ours(Post-Prune)	**0.823**	**27.493**	**0.211**	0.897	29.268	0.265	0.837	23.530	0.200

serving rendering stability throughout the sparsification process. However, similar to auxiliary-based methods, this paradigm necessitates complex supplementary loss formulations that substantially increase the computational overhead and memory footprint during the backward pass.

Our proposed Tile-based Density Accumulation Pruning (TDAP), progressively accumulate geometry information and executes pruning and opacity modulation through overall quantization without Voxel-like or spatial structure(Lu et al. (2023), Lin et al. (2024), Ren et al. (2024)) for computing. Achieving loss-free topological transition with zero additional computational cost.

3 METHOD

3.1 PRELIMINARIES: 3D GAUSSIAN SPLATTING

3D Gaussian Splatting (3DGS) (Kerbl et al., 2023) represents a scene as an explicit set of anisotropic 3D Gaussians. Each Gaussian G_i has a mean $\mu_i \in \mathbb{R}^3$ and covariance $\Sigma_i \in \mathbb{R}^{3\times3}$. The covariance is parameterized using a rotation R_i and a diagonal scale matrix S_i:

$$\Sigma_i = R_i S_i S_i^\top R_i^\top. \tag{1}$$

This parameterization keeps Σ_i positive semidefinite during optimization. Each Gaussian also has an opacity parameter $o_i \in [0, 1]$ and spherical harmonic coefficients that determine its view-dependent color. For a given camera, the Gaussians are projected into the image plane. Using a local affine approximation of the projection, the screen-space covariance of G_i is

$$\Sigma_i^{\mathrm{2D}} = J_i W \Sigma_i W^\top J_i^\top, \tag{2}$$

where W transforms world coordinates to camera coordinates and J_i is the Jacobian of the perspective projection evaluated at the transformed mean of G_i. The rasterizer assigns projected Gaussians to the image tiles they overlap. In the original implementation, tiles contain 16×16 pixels. Within each tile, Gaussian instances are sorted by depth for compositing.

For a pixel p, let $\mathcal{N}(p)$ denote the Gaussians that contribute to that pixel, ordered from front to back. Its rendered color is

$$C(p) = \sum_{i\in\mathcal{N}(p)} c_i(p)\,\alpha_i(p) \prod_{\substack{j\in\mathcal{N}(p)\\ j<i}} \big(1-\alpha_j(p)\big), \tag{3}$$

where $c_i(p)$ is the view-dependent color obtained from the spherical harmonic coefficients, and $\alpha_i(p)$ combines the Gaussian's opacity parameter with its screen-space contribution at p. Thus, $\alpha_i(p)$ is generally different from o_i.

3DGS interleaves parameter optimization with adaptive density control (ADC). ADC accumulates the magnitudes of view-space positional gradients and selects Gaussians whose average gradient

exceeds a threshold for densification. Selected Gaussians are cloned or split according to their scale: cloning adds capacity around smaller primitives, whereas splitting subdivides larger ones. The original method also periodically caps Gaussian opacities to encourage subsequent optimization to reassess their contributions. Although densification improves reconstruction, gradient-based selection does not directly measure whether a newly created Gaussian provides a distinct contribution across training views. As a result, the primitive count and associated training memory can grow substantially. This motivates TDAP-GS: an in-training approach that evaluates candidate redundancy using tile-level rendering information and controls primitive growth through progressive pruning.

3.2 Tile-based Density Evaluation

TDAP estimates how strongly each Gaussian is supported by nearby primitives in the views used for training. It reuses the depth-sorted Gaussian lists produced by tile-based rasterization, avoiding a separate global search over all pairs of primitives. The resulting score is a local support measure: Gaussians that repeatedly appear near other Gaussians in both projected tile assignment and 3D position receive higher scores.

Let $\mathcal{L}_{m,t}$ denote the depth-sorted list of Gaussian instances assigned to tile t when rendering training view m. Because a Gaussian may overlap several tiles, it can occur in more than one list for the same view. If G_i occurs at position p in $\mathcal{L}_{m,t}$, let $\mathcal{N}^W_{m,t}(i)$ be the entries at positions $p-W,\dots,p-1,p+1,\dots,p+W$, excluding positions outside the list. We use $W=2$. For this occurrence, we compute

$$\ell_{i,m,t} = \frac{1}{|\mathcal{N}^W_{m,t}(i)|} \sum_{j\in\mathcal{N}^W_{m,t}(i)} \exp\left[-\frac{\|\mu_i-\mu_j\|_2^2}{2\beta^2(\hat{s}_i^2+\hat{s}_j^2)+\epsilon}\right], \tag{4}$$

where $\mu_i \in \mathbb{R}^3$ is the Gaussian mean, $\hat{s}_i = (s_{i,x}+s_{i,y}+s_{i,z})/3$ is its mean scale, $\beta>0$ sets the distance bandwidth, and $\epsilon>0$ provides numerical stability. We define $\ell_{i,m,t}=0$ if the neighborhood is empty. Using the number of available neighbors in the denominator keeps the score comparable for entries near either end of a tile list.

Depth sorting determines *which* Gaussians are compared; the 3D distance and scale determine the strength of each comparison. In particular, adjacent entries in a depth-sorted list do not necessarily lie close together in 3D, which is why the distance term is needed. The positive, squared scale bandwidth also avoids the undefined or negative denominators that arise from using a difference of Gaussian scales.

We next aggregate repeated tile occurrences within each view. Let $\mathcal{T}_{i,m}$ be the tiles whose sorted lists contain G_i. Its support in view m is

$$\ell^{\text{view}}_{i,m} = \frac{1}{|\mathcal{T}_{i,m}|} \sum_{t\in\mathcal{T}_{i,m}} \ell_{i,m,t}. \tag{5}$$

This average prevents a Gaussian from receiving a higher score solely because its projected footprint intersects more tiles. Let $\mathcal{M}_i$ be the set of distinct training views in which G_i has been evaluated, and let $H_i=|\mathcal{M}_i|$. We average support across these views:

$$\bar{\ell}_i = \frac{1}{H_i} \sum_{m\in\mathcal{M}_i} \ell^{\text{view}}_{i,m}. \tag{6}$$

Finally, the density score used for ranking is

$$d_i = o_i\,\bar{\ell}_i\left(1-e^{-H_i/h_0}\right), \tag{7}$$

where $o_i\in[0,1]$ is the Gaussian's opacity parameter and $h_0>0$ controls how quickly the observation factor saturates. We set $d_i=0$ before the Gaussian has an eligible observation. The opacity term lowers the score of primitives with small current rendering weight, whereas the observation factor prevents support estimated from very few views from being treated as equally reliable as support observed repeatedly. Unlike inverse-sigmoid weighting, this factor increases, rather than decreases, with H_i.

The score is updated at the density-evaluation events defined below. Only running sums and counts are needed to maintain Eqs. equation 6–equation 7; previous tile lists need not be stored. Because

the neighborhood is restricted to $2W$ list entries, the additional comparisons scale linearly with the number of processed Gaussian–tile occurrences for fixed W. The actual runtime overhead is reported in Sec. 4.

3.3 Quantile-guided Opacity Modulation

The density score identifies candidates, but immediate removal can cause abrupt changes in the rendered scene. TDAP therefore adjusts their opacity before applying the scheduled pruning operation. At pruning event k, let $\mathcal{E}_k$ be the set of eligible Gaussians and let $q_p^{(k)}$ be the p-th percentile of $\{d_i : i \in \mathcal{E}_k\}$. We define a penalty weight for low-density Gaussians,

$$a_i^{(k)} = \mathrm{clip}\left(\frac{d_i - q_{25}^{(k)}}{q_{50}^{(k)} - q_{25}^{(k)} + \epsilon}, 0, 1\right), \tag{8}$$

and a boost weight for high-density Gaussians,

$$b_i^{(k)} = \mathrm{clip}\left(\frac{d_i - q_{85}^{(k)}}{q_{100}^{(k)} - q_{85}^{(k)} + \epsilon}, 0, 1\right). \tag{9}$$

Opacity is updated by

$$o_i \leftarrow \begin{cases} o_i\, a_i^{(k)}, & d_i < q_{50}^{(k)}, \\ o_i + (1 - o_i) b_i^{(k)}, & d_i > q_{85}^{(k)}, \\ o_i, & \text{otherwise.} \end{cases} \tag{10}$$

The penalty increases continuously from its strongest value below $q_{25}^{(k)}$ to no penalty at $q_{50}^{(k)}$. The boost increases from zero at $q_{85}^{(k)}$ toward the highest density scores. Eq. (10) preserves the interval $o_i \in [0, 1]$. Opacity modulation changes a Gaussian's rendered contribution but does not itself release its parameter or optimizer-state memory. Physical deletion is performed by the stepwise pruning rule in Sec. 3.4. No auxiliary training loss is introduced.

3.4 Progressive Stepwise Pruning

TDAP divides pruning into events separated by $\Delta = 500$ optimization iterations. For a pruning phase spanning iterations t_{start} to t_{end}, the number of events is

$$N = \frac{t_{\mathrm{end}} - t_{\mathrm{start}}}{\Delta}, \tag{11}$$

when the phase length is divisible by Δ. An event occurs at $t_{\mathrm{start}} + \Delta, \ldots, t_{\mathrm{end}}$. Suppose the intended retention fraction over a phase is $R \in (0, 1]$. For a fixed population and no intervening densification, a constant per-event retention fraction K satisfies

$$K^N = R, \qquad K = R^{1/N}. \tag{12}$$

For example, $R = 0.1$ means retaining 10% of the eligible population over the full phase; it does not mean pruning 10%. Because ADC may introduce new Gaussians between events, the population is not generally fixed. We therefore apply K to the eligible population present at each event. If n_k^{pre} is its size immediately before event k, the target count immediately after that event is

$$n_k^{\mathrm{post}} = \lceil K n_k^{\mathrm{pre}} \rceil. \tag{13}$$

Gaussians are ranked by d_i, and the lowest-ranked candidates are removed until the target count is reached. Thus, Eq. equation 12 specifies the nominal retention schedule, while Eq. equation 13 defines its operation when densification changes the population. The final count need not equal R times the phase's initial count if new primitives are created during the phase.

4 Experiments

4.1 Experimental Setup

Datasets. We evaluate TDAP-GS on three real-world novel-view synthesis benchmarks: Mip-NeRF 360 (Barron et al., 2022), Tanks and Temples (Knapitsch et al., 2017), and Deep Blending

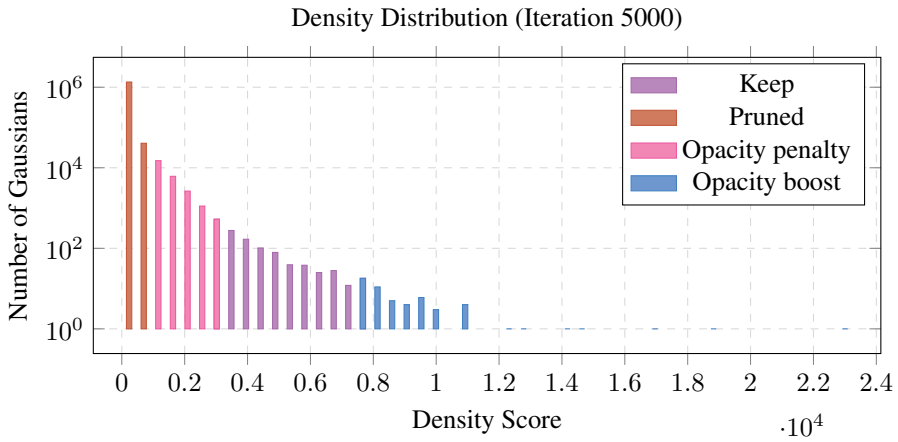


Figure 3: Histogram of Gaussian density scores at iteration 5K. Most primitives have low scores, while a smaller group has high scores. Colors indicate the action assigned to each density range: Pruned for removal, Penalty Tier for opacity attenuation, Boost Tier for opacity enhancement, and Keep for no change.

(Hedman et al., 2018). Mip-NeRF 360 contains nine scenes, comprising five outdoor and four indoor scenes. For Tanks and Temples and Deep Blending, we use the scenes specified in our evaluation protocol. All methods are trained and evaluated using the same scene selections, train–test splits, and image resolutions.

Evaluation metrics. We measure novel-view quality on held-out images using peak signal-to-noise ratio (PSNR), structural similarity (SSIM), and learned perceptual image patch similarity (LPIPS). Higher PSNR and SSIM and lower LPIPS indicate better reconstruction quality. To assess efficiency, we report the final number of Gaussian primitives, the primitive count throughout training, and peak allocated GPU memory during training. Peak memory is measured over the complete training run, including densification, pruning, and refinement. We additionally report training time and rendering throughput to distinguish memory savings from changes in computational cost. All efficiency measurements use the same hardware and measurement procedure for TDAP-GS and the baselines.

Implementation details. We implement TDAP-GS on the Mip-Splatting codebase (Yu et al., 2023) and incorporate the tile-based density computation into its CUDA rasterizer. Unless otherwise stated, TDAP-GS and the corresponding Mip-Splatting baseline share the same initialization, reconstruction loss, densification settings, image resolution, and training schedule. Pruning events occur every 500 iterations. We apply early pruning during iterations 0–15,000 and late pruning during iterations 15,000–30,000, followed by refinement without TDAP pruning during iterations 30,000–40,000. Experiments are run on a single NVIDIA RTX 3090 GPU with 24 GB of VRAM and a system with 32 GB of RAM.

4.2 Results

Effect of early pruning. We first examine whether TDAP-GS controls primitive growth during the initial densification phase. Figure 4 plots Gaussian count and allocated GPU memory against training iteration for TDAP-GS and the baseline. These trajectories show when the methods diverge and whether a lower final primitive count also corresponds to lower *peak training memory*. We report peak values over the full training run, since memory saved early in training is not sufficient if a higher peak occurs during a later phase.

We evaluate sensitivity to the early-phase pruning fraction $P \in \{0.15, 0.25, 0.35, 0.45\}$. Here, P denotes the fraction targeted for removal over the *entire early-pruning phase*, rather than the fraction removed at each 500-iteration event. For each setting, we compare the primitive-count trajectory,

peak training memory, and final PSNR, SSIM, and LPIPS. This analysis identifies whether stronger early pruning reduces resource use at the cost of novel-view quality and motivates the default setting used in subsequent experiments.

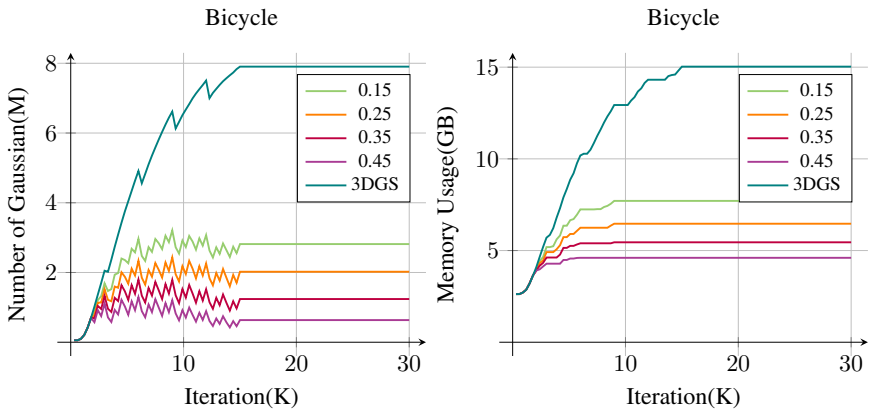


Figure 4: (Left) Number of Gaussian through Iteration. As display, applying higher pruning ratios effectively suppresses the unconstrained proliferation of primitives early in the training cycle. (Right) reports the maximum GPU VRAM consumption recorded during the active training phase.

With different threshold P leads to different final pruning ratio. For $P = 0.15$, the final number of primitives is around 40% of the baseline, which is 60% pruned. For $P = 0.25$, retain 30%, for $P = 0.35$, retain 40%, and for $P = 0.45$, retain 35%. We also do not recommend setting $P > 0.5$ because the pre-pruning stage heavily rely on ADC to regrow the gaussians back. Setting P more than 0.5 could lead to unrecoverable destruction after pruning for several times and the remaining gaussians may not able to optimize to construct complete scene.

Post Prune: Traditional two-stage hard-pruning strategies induce severe rendering degradation, causing a sudden drop in rendering quality immediately following primitive removal. By replacing two-stage deletion with our Quantile-Guided Opacity Modulation and Progressive Stepwise Pruning schedule, TDAP-GS completely circumvents this degradation. Penalizing the opacity of redundant candidates while solidifying high-density core structures. We also illustrate the density distribution while training, please refer to Figure 3. However, we observe that TDAP-GS is comparatively less competitive on the Deep Blending and Tanks and Temples datasets compared to the baselines. See Section 4.2 for a detailed limitation analysis

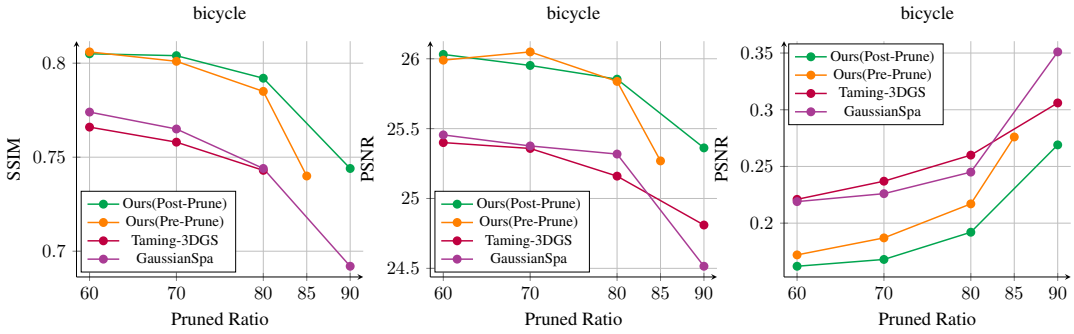


Figure 5: Post-Prune result compare to exists methods. (Left)SSIM score in scene bicycle (Middle) PSNR score in scene bicycle (Right) LPIPS score in scene bicycle. Notice that pre-prune is hardly achieve 90% pruning. For the detail setting of other models, please refer to Appendix A.

Post Refine: We evaluate the final converged models after the fine-tuning period, targeting an aggressive 80% overall pruning ratio. As shown in Table 1, we primarily compare TDAP-GS against state-of-the-art reduction methodologies, specifically LightGaussian and PUP-3DGS. As demonstrated in our quantitative results, our framework comprehensively outperforms both baselines across

all rendering metrics (PSNR, SSIM, and LPIPS) on the Mip-NeRF 360 dataset. As for Deep Blending and Tank and Temples, we thoroughly discuss the causes of this generalization bottleneck in Section 4.2.

Limitation: While achieving state-of-the-art results on Mip-NeRF 360, its performance gains are less pronounced on Deep Blending and Tanks and Temples. Our empirical ablation (Table 2) reveal that simply expanding the window size (e.g., from $W = 2$ to $W = 60$) yields marginal performance improvements. This indicates that the bottleneck is not the size of the local context, but rather the fundamental instability of screen-space depth sorting under extreme trajectory variations.

Table 2: Different window size effects on Deep Blending and Tank and Temples.

Method	Deep Blending			Tank and Temples		
	SSIM↑	PSNR↑	LPIPS↓	SSIM↑	PSNR↑	LPIPS↓
W=5	0.896	29.224	0.266	0.837	23.578	0.201
W=15	0.895	29.164	0.265	0.834	23.395	0.202
W=30	0.895	29.254	0.266	0.832	23.457	0.201
W=60	0.895	29.211	0.265	0.833	23.376	0.203

4.3 Ablation Study

To rigorously validate the necessity and individual contribution of each component within our TDAP-GS framework, we perform an extensive ablation analysis. We isolate three key design choices: (1) the Tile-based Density Evaluation, (2) the Quantile-Guided Opacity Modulation, and (3) the Progressive Stepwise Pruning schedule.

Table 3: Qualitative ablation of TDAP-GS. Removing quantile-guided opacity modulation (**w/o OM**) leaves only direct pruning and produces visible holes in textured regions. Omitting the multi-view observation factor from the density score (**w/o HC**) changes candidate selection. Replacing progressive stepwise pruning with a single pruning event (**w/o PSP**) produces blur and incomplete surface recovery.

			Mip-Nerf360			Deep Blending			Tank and Temples		
om	hc	psp	SSIM↑	PSNR↑	LPIPS↓	SSIM↑	PSNR↑	LPIPS↓	SSIM↑	PSNR↑	LPIPS↓
			0.820	27.340	0.212	0.895	29.113	0.268	0.834	23.370	0.202
✔			0.821	27.361	0.212	0.896	29.178	0.266	0.836	23.521	0.202
	✔		0.821	27.331	0.212	0.896	29.129	0.268	0.833	23.355	0.204
		✔	0.822	27.464	0.213	0.897	**29.279**	0.265	0.836	**23.575**	0.204
✔	✔	✔	**0.823**	**27.493**	**0.211**	**0.897**	29.268	**0.265**	**0.837**	23.530	**0.200**

5 Conclusions

In this paper, we introduced Tile-based Density Accumulation Pruning (TDAP-GS), a highly efficient lightweighted framework designed to resolve the memory bottlenecks and optimization instabilities inherent in 3D Gaussian Splatting. By leveraging the native depth-sorting mechanisms of the CUDA rasterizer, our method continuously accumulates density for each primitive with near-zero computational overhead. This temporal density serves as a foundation for our Quantile-Guided Opacity Modulation, allowing us to accurately identify and penalize redundant primitives while solidifying essential structures. TDAP-GS successfully pruning nearly 80% of the total Gaussian population and drastically reducing peak VRAM consumption during training. Providing a highly practical and scale-adaptive density window represents a key direction for future 3D scene research.

Despite establishing a new optimal balance between memory efficiency and rendering quality, our current formulation exhibits limitations on datasets characterized by highly irregular camera trajectories. Overcoming this limitation by decoupling density estimation from the 2D screen space—favoring explicit 3D geometric evaluations while maintaining computational efficiency—remains an important avenue for future work.

AI USE STATEMENT

In this work, we used generative AI tools for english writing polishment using Gemini Pro. We have not used generative AI tools for architecture and code generation,

We have reviewed all AI-assisted work. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Yanqi Bao, Tianyu Ding, Jing Huo, Yaoli Liu, Yuxin Li, Wenbin Li, Yang Gao, and Jiebo Luo. 3d gaussian splatting: Survey, technologies, challenges, and opportunities, 2024. URL https://arxiv.org/abs/2407.17418.

Jonathan T. Barron, Ben Mildenhall, Dor Verbin, Pratul P. Srinivasan, and Peter Hedman. Mip-nerf 360: Unbounded anti-aliased neural radiance fields, 2022. URL https://arxiv.org/abs/2111.12077.

Guikun Chen and Wenguan Wang. A survey on 3d gaussian splatting, 2026. URL https://arxiv.org/abs/2401.03890.

Tianchen Deng, Chang Nie, Shuhong Liu, Wenhua Wu, Jianfei Yang, Shenghai Yuan, Jiuming Liu, Danwei Wang, and Hesheng Wang. Compact 3d gaussian splatting for dense visual slam, 2026. URL https://arxiv.org/abs/2403.11247.

Zhiwen Fan, Kevin Wang, Kairun Wen, Zehao Zhu, Dejia Xu, and Zhangyang Wang. Lightgaussian: Unbounded 3d gaussian compression with 15x reduction and 200+ fps, 2024. URL https://arxiv.org/abs/2311.17245.

Guangchi Fang and Bing Wang. Mini-splatting: Representing scenes with a constrained number of gaussians, 2024. URL https://arxiv.org/abs/2403.14166.

Guangchi Fang and Bing Wang. Efficient scene modeling via structure-aware and region-prioritized 3d gaussians, 2026. URL https://arxiv.org/abs/2411.12788.

Antoine Guédon and Vincent Lepetit. Sugar: Surface-aligned gaussian splatting for efficient 3d mesh reconstruction and high-quality mesh rendering, 2023. URL https://arxiv.org/abs/2311.12775.

Alex Hanson, Allen Tu, Vasu Singla, Mayuka Jayawardhana, Matthias Zwicker, and Tom Goldstein. Pup 3d-gs: Principled uncertainty pruning for 3d gaussian splatting, 2025. URL https://arxiv.org/abs/2406.10219.

Peter Hedman, Julien Philip, True Price, Jan-Michael Frahm, George Drettakis, and Gabriel Brostow. Deep blending for free-viewpoint image-based rendering. *ACM Trans. Graph.*, 37(6), December 2018. ISSN 0730-0301. doi: 10.1145/3272127.3275084. URL https://doi.org/10.1145/3272127.3275084.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network, 2015. URL https://arxiv.org/abs/1503.02531.

Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering, 2023. URL https://arxiv.org/abs/2308.04079.

Arno Knapitsch, Jaesik Park, Qian-Yi Zhou, and Vladlen Koltun. Tanks and temples: benchmarking large-scale scene reconstruction. *ACM Trans. Graph.*, 36(4), July 2017. ISSN 0730-0301. doi: 10.1145/3072959.3073599. URL https://doi.org/10.1145/3072959.3073599.

Joo Chan Lee, Daniel Rho, Xiangyu Sun, Jong Hwan Ko, and Eunbyung Park. Compact 3d gaussian representation for radiance field, 2024. URL https://arxiv.org/abs/2311.13681.

Jiaqi Lin, Zhihao Li, Xiao Tang, Jianzhuang Liu, Shiyong Liu, Jiayue Liu, Yangdi Lu, Xiaofei Wu, Songcen Xu, Youliang Yan, and Wenming Yang. Vastgaussian: Vast 3d gaussians for large scene reconstruction, 2024. URL https://arxiv.org/abs/2402.17427.

Yifei Liu, Zhihang Zhong, Yifan Zhan, Sheng Xu, and Xiao Sun. Maskgaussian: Adaptive 3d gaussian representation from probabilistic masks, 2025. URL https://arxiv.org/abs/2412.20522.

Tao Lu, Mulin Yu, Linning Xu, Yuanbo Xiangli, Limin Wang, Dahua Lin, and Bo Dai. Scaffold-gs: Structured 3d gaussians for view-adaptive rendering, 2023.

Saswat Subhajyoti Mallick, Rahul Goel, Bernhard Kerbl, Francisco Vicente Carrasco, Markus Steinberger, and Fernando De La Torre. Taming 3dgs: High-quality radiance fields with limited resources, 2024. URL https://arxiv.org/abs/2406.15643.

Simon Niedermayr, Josef Stumpfegger, and Rüdiger Westermann. Compressed 3d gaussian splatting for accelerated novel view synthesis, 2024. URL https://arxiv.org/abs/2401.02436.

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, 2024. URL https://arxiv.org/abs/2403.17898.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter, 2020. URL https://arxiv.org/abs/1910.01108.

Lin Wang and Kuk-Jin Yoon. Knowledge distillation and student-teacher learning for visual intelligence: A review and new outlooks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(6):3048–3068, 2022. ISSN 1939-3539. doi: 10.1109/tpami.2021.3055564. URL http://dx.doi.org/10.1109/TPAMI.2021.3055564.

Mingqiao Ye, Martin Danelljan, Fisher Yu, and Lei Ke. Gaussian grouping: Segment and edit anything in 3d scenes, 2024. URL https://arxiv.org/abs/2312.00732.

Zehao Yu, Anpei Chen, Binbin Huang, Torsten Sattler, and Andreas Geiger. Mip-splatting: Alias-free 3d gaussian splatting, 2023. URL https://arxiv.org/abs/2311.16493.

Yangming Zhang, Wenqi Jia, Wei Niu, and Miao Yin. Gaussianspa: An "optimizing-sparsifying" simplification framework for compact and high-quality 3d gaussian splatting, 2025. URL https://arxiv.org/abs/2411.06019.