AdpSplit: Error-Driven Adaptive Splitting for Faster Geometry Discovery in 3D Gaussian Splatting

Yongjae Lee
Arizona State University
Tempe, AZ, 85281
ylee298@asu.edu

Jingxing Li
Arizona State University
Tempe, AZ, 85281
jingxing@asu.edu

Abhay Kumar Yadav
Johns Hopkins University
Baltimore, MD, 21218
ayadav13@jh.edu

Rama Chellappa
Johns Hopkins University
Baltimore, MD, 21218
rchella4@jhu.edu

Deliang Fan*
Arizona State University
Tempe, AZ, 85281
dfan@asu.edu

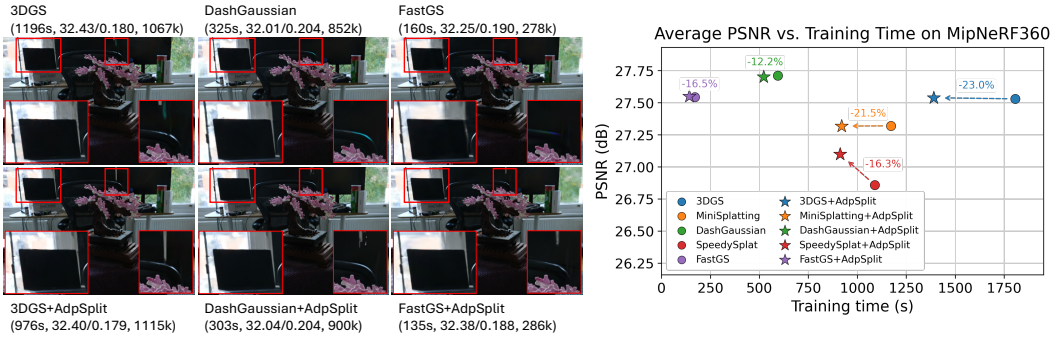

Figure 1: AdpSplit is a drop-in split operator for efficient 3D Gaussian Splatting (3DGS) training. Across multiple accelerated 3DGS baselines, its error-driven adaptive split improves reconstruction of under-modeled regions while reducing training time. Values reported for the `Bonsai` scene are ordered as (training time, PSNR/LPIPS, number of Gaussians). Percentages in the graph indicate reductions in training time.

Abstract

Adaptive density control in 3D Gaussian Splatting (3DGS) repeatedly grows the Gaussian population through fixed-cardinality random splitting to discover useful scene structure. However, in vanilla 3DGS, its binary split operator requires many densification rounds to expose fine details, making it a bottleneck for efficient training schedules with fewer iterations. We introduce **AdpSplit**, an error-driven adaptive split operator that determines the number of split children and initializes the child parameters from L1-pixel-error region statistics, enabling fewer densification iterations, thus reduced training time, while preserving the rendering quality of full-schedule training. Across the MipNeRF360, Deep-Blending, and Tanks&Temples datasets, AdpSplit reduces the training time of multiple accelerated 3DGS pipelines by 9.2%–22.3% as a simple drop-in replacement for the standard split operator. With FastGS, AdpSplit matches the full-schedule PSNR on MipNeRF360 while reducing training time by 16.4%, achieving a 12.6× speedup over vanilla 3DGS.

*Corresponding author

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

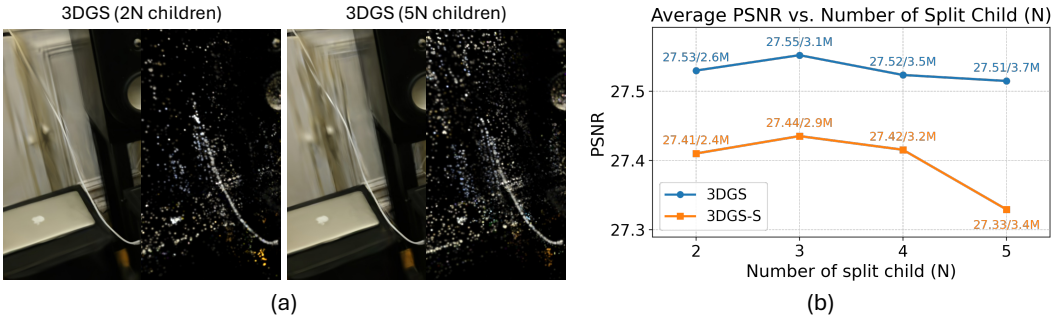

Figure 2: Using more randomly placed children can increase the risk of amplifying noise (a). Increasing the number of split children does not necessarily improve reconstruction quality (b). Values indicate PSNR/the number of Gaussians, and 3DGS-S is trained with the densification schedule shortened from 15k to 9k iterations.

1 Introduction

Novel view synthesis (NVS) is a core computer vision task that aims to generate photorealistic images from novel viewpoints given a set of observed images. Neural radiance fields (NeRF) [Mildenhall et al., 2021] showed that a scene can be represented as a continuous neural function, enabling high-fidelity view synthesis. However, NeRF relies on dense volumetric sampling and repeated forward passes through the neural networks, making both optimization and rendering computationally expensive. In contrast, 3D Gaussian Splatting (3DGS) [Kerbl et al., 2023] introduced an explicit scene representation based on learnable anisotropic Gaussian primitives and a differentiable rasterizer. With fast optimization, real-time rendering, and competitive visual quality, 3DGS has become a practical foundation for downstream applications such as SLAM [Yan et al., 2024], dynamic scene modeling [Wu et al., 2024], text-to-3D generation [Tang et al., 2024], and digital humans [Qian et al., 2024].

The quality and efficiency of 3DGS depend heavily on how the Gaussian population is built during training. Starting from a sparse structure-from-motion [Snavely et al., 2006] point cloud, the 3DGS pipeline applies adaptive density control (ADC), growing the Gaussian population to fill missing scene geometry. In this sense, ADC is fundamentally a scene-structure search mechanism that discovers a useful scene structure through repeated densification rounds. Naive propagation, however, often requires many such rounds, thereby slowing optimization and producing redundant Gaussians.

To accelerate optimization, one line of work reduces the number of Gaussians through pruning, compression, or structured representations while leaving the standard ADC process untouched [Girish et al., 2025, Lee et al., 2024, Zhang et al., 2024, Lu et al., 2024, Hanson et al., 2025a]. Another line of work addresses Gaussian propagation more directly through refined split candidate selection, optimization-based splitting, or reinitializing the scene [Fang and Wang, 2024, Cheng et al., 2024, Wang et al., 2025, Ren et al., 2026]. These advances are important, and they mainly answer which Gaussians should be split or removed, or where the propagated Gaussians should be placed. However, how many split children should be produced for each split candidate, and what shapes those children should take, remain unexplored.

The standard ADC split operator produces a fixed number of children, typically two, for each split candidate and places them at randomly sampled locations. This fixed-cardinality random splitting is a fundamental bottleneck. If a parent Gaussian (i.e., a split candidate) covers a structure that eventually requires G distinct Gaussians, a binary split needs at least $\log_2 G$ densification rounds to expose that structure. A naive alternative is to split each parent into more than two children, but additional children alone do not guarantee a better scene representation. Randomly initialized children may redundantly fit local observations, amplify noisy details, or miss the regions that actually cause view loss. Figure 2a visualizes this problem: the five-child split allocates additional Gaussians to under-reconstructed regions, but places them poorly for reducing perceptual error. This is especially problematic in shortened training schedules, where fewer densification steps are available. In our pilot study (Figure 2b), a densification schedule shortened from 15k to 9k iterations with five children

per parent produces more Gaussians than a longer binary-split schedule, yet achieves lower PSNR (27.33 PSNR with 3.4M Gaussians for 3DGS-S (5N) vs. 27.53 PSNR with 2.6M Gaussians for 3DGS (2N)). Thus, the core issue is not only how many Gaussians are added, but how those Gaussians are initialized.

We propose **AdpSplit**, an error-driven adaptive split method that replaces fixed-cardinality random child placement with view-loss-aware child generation. During ADC, AdpSplit renders sampled training views, identifies high-error pixels for which the candidate is a dominant contributor, and partitions these pixels into connected error regions. Each region proposes one child, so the number of children is determined by the observed error structure rather than a global constant. AdpSplit then initializes each child by back-projecting the region centroid to the depth that best agrees with the parent Gaussian, estimating its covariance from the region's statistics, and initializing its color from the ground-truth observation. Finally, cross-view child merging removes redundant children generated from different views of the same portion of a Gaussian. As shown in Figure 1, AdpSplit can be used as a drop-in replacement for the standard ADC split operator in existing 3DGS training accelerators, further reducing training time while maintaining rendering quality. Our contributions are summarized as follows:

- We identify fixed-cardinality random splitting as a hidden bottleneck in fast 3DGS training and show that naively increasing the number of children can increase the Gaussian count while leaving reconstruction quality unchanged or even impairing it.
- We introduce AdpSplit, a new ADC split operator that adaptively determines the number of children, estimates their parameters from attributable L1-pixel-error region statistics, and merges redundant child proposals across views.
- We demonstrate that AdpSplit can be integrated with existing 3DGS acceleration pipelines, achieving comparable rendering quality to the baselines with a shortened training schedule.

2 Related Works

We review prior work on accelerating 3DGS training, focusing on scene compaction and adaptive density control.

Gaussian compaction. One common strategy is to reduce the Gaussian count. These methods learn pruning masks, estimate uncertainty, or analyze importance statistics to prune low-utility Gaussians that have little impact on reconstruction quality [Fan et al., 2024, Zhang et al., 2024, Hanson et al., 2025b, Morgenstern et al., 2025, Lee et al., 2024, Niedermayr et al., 2024, Niemeyer et al., 2025, Hanson et al., 2025a]. Other methods focus on parameter-level information compression via entropy encoding or quantization, or on Gaussian-level compression through structured representations [Girish et al., 2025, Lu et al., 2024, Ren et al., 2025, Chen et al., 2025a,b]. These approaches are complementary to AdpSplit. They mainly compress, encode, or sparsify the representation after Gaussians have been created, whereas our method targets the earlier scene-building decision of how split children are generated.

Adaptive density control. Adaptive density control [Kerbl et al., 2023] grows the 3DGS scene representation from sparse initialization by splitting and cloning Gaussians selected using image-space gradient statistics. Later work has shown that this heuristic can be inefficient and unstable, motivating more principled growth rules, budgeted selection, depth reinitialization, optimization-based splitting, and error-guided insertion [Rota Bulò et al., 2025, Fang and Wang, 2024, Cheng et al., 2024, Kheradmand et al., 2024, Wang et al., 2025, Ren et al., 2026, Chen et al., 2025c, Baranowski et al., 2026, Gao et al., 2025]. These works reduce redundant growth, but the standard split operation commonly remains fixed-cardinality, and the initialization of new Gaussians is not directly integrated with the splitting and cloning operations.

Positioning of AdpSplit. To the best of our knowledge, no existing work has shown that an effective split operation can expedite the discovery of useful scene structure. The Gaussian initialization strategies in MiniSplatting [Fang and Wang, 2024] and ConeGS [Baranowski et al., 2026] can improve the spatial distribution of Gaussians, but they can substantially increase the number of Gaussians that must be optimized or require external geometric priors. FastGS [Ren et al., 2026]

and DashGaussian [Chen et al., 2025c] are designed for fast schedules, but they primarily regulate candidate selection or Gaussian budgets rather than estimating the number and parameters of split children from local error structure. SteepGS [Wang et al., 2025] is particularly relevant because it asks in which direction the children should be deployed; however, it does not consider how many children to create or where exactly they should be placed. Our approach is orthogonal to existing works, as supported by the experiments in section 5.

3 Preliminaries: 3D Gaussian Splatting

3DGS represents a scene as a set of G Gaussians $\mathcal{G} = \{\mathcal{G}_i\}_{i=1}^{G}$, each parameterized by a mean $\boldsymbol{\mu}_i \in \mathbb{R}^3$, a 3D covariance $\boldsymbol{\Sigma}_i = \mathbf{R}_i \mathbf{S}_i \mathbf{S}_i^\top \mathbf{R}_i^\top$ with rotation matrix $\mathbf{R}_i$ and diagonal scale matrix $\mathbf{S}_i = \mathrm{diag}(\mathbf{s}_i)$ with $\mathbf{s}_i \in \mathbb{R}^3$, an opacity $o_i \in [0, 1]$, and spherical harmonic (SH) color coefficients $\mathbf{c}_i = \{\mathbf{c}_i^{\mathrm{dc}}, \mathbf{c}_i^{\mathrm{rest}}\}$, consisting of a DC component and higher-order coefficients. Gaussians are projected to the image plane and α-blended in depth order with a differentiable rasterizer. For a pixel u in view v, the rendered color is

$$\hat{\mathbf{I}}_v(u) = \sum_{i \in \mathcal{O}_v(u)} T_{i,v}(u)\, \alpha_{i,v}(u)\, \mathbf{c}_i(\mathbf{d}_v), \quad T_{i,v}(u) = \prod_{j<i} \big(1 - \alpha_{j,v}(u)\big), \tag{1}$$

where $\mathcal{O}_v(u)$ is the set of Gaussians overlapping pixel u in view v in front-to-back depth order, $\alpha_{i,v}(u)$ combines the learned opacity with the projected 2D Gaussian footprint, $T_{i,v}(u)$ is the accumulated transmittance, and $\mathbf{c}_i(\mathbf{d}_v)$ is the view-dependent color decoded from the SH coefficients along viewing direction $\mathbf{d}_v$.

The ADC accumulates image-space gradients for visible Gaussians. At each densification interval, it first computes the average gradient statistic g_i for each Gaussian. Among high-gradient Gaussians, those with small spatial scale are cloned, while those with large spatial scale are split. The split candidate set $\mathcal{S}$ is defined as

$$i \in \mathcal{S} \quad \Longleftrightarrow \quad g_i \geq \tau_g \ \wedge\ \max(\mathbf{s}_i) > \tau_s, \tag{2}$$

where τ_g is the densification gradient threshold and τ_s is the scale threshold.

For each selected parent $\mathcal{G}_i$, the vanilla split operator generates a fixed number N of children, with $N = 2$ by default. Each child n is placed near the parent by sampling an offset from the parent's Gaussian distribution:

$$\boldsymbol{\delta}_{i,n} \sim \mathcal{N}(\mathbf{0}, \mathrm{diag}(\mathbf{s}_i^2)), \quad \boldsymbol{\mu}'_{i,n} = \boldsymbol{\mu}_i + \mathbf{R}_i \boldsymbol{\delta}_{i,n}. \tag{3}$$

The remaining child parameters are initialized by copying the parent's rotation, opacity, and SH coefficients, while uniformly shrinking the parent scale with shrink factor η:

$$\mathbf{s}'_{i,n} = \frac{\mathbf{s}_i}{\eta N}, \quad \mathbf{R}'_{i,n} = \mathbf{R}_i, \quad o'_{i,n} = o_i, \quad \mathbf{c}'_{i,n} = \mathbf{c}_i. \tag{4}$$

After appending the children to the Gaussian set, the selected parents are removed.

4 Method

Figure 3 gives a high-level overview of AdpSplit. At each densification round, we use the standard 3DGS gradient-and-size rule (Equation 2) to obtain the split candidate set $\mathcal{S}$, while small high-gradient Gaussians are still handled by the original clone operation. For each candidate in $\mathcal{S}$, AdpSplit proceeds in three stages: **(1) error region partitioning**, which attributes high-error pixels to the candidate and groups them into error regions (Section 4.1); **(2) child initialization**, which initializes one or more children from the statistics of the error regions (Section 4.2); and **(3) cross-view child merging**, which removes redundant children across views before insertion into the Gaussian set (Section 4.3).

4.1 Error Region Partitioning

We randomly sample V training cameras and render them. During rendering, we also collect a dominant Gaussian map D_v by selecting, for each pixel, the Gaussian with the largest alpha-weighted contribution:

$$D_v(u) = \arg \max_{i \in \mathcal{O}_v(u)} T_{i,v}(u)\, \alpha_{i,v}(u), \tag{5}$$

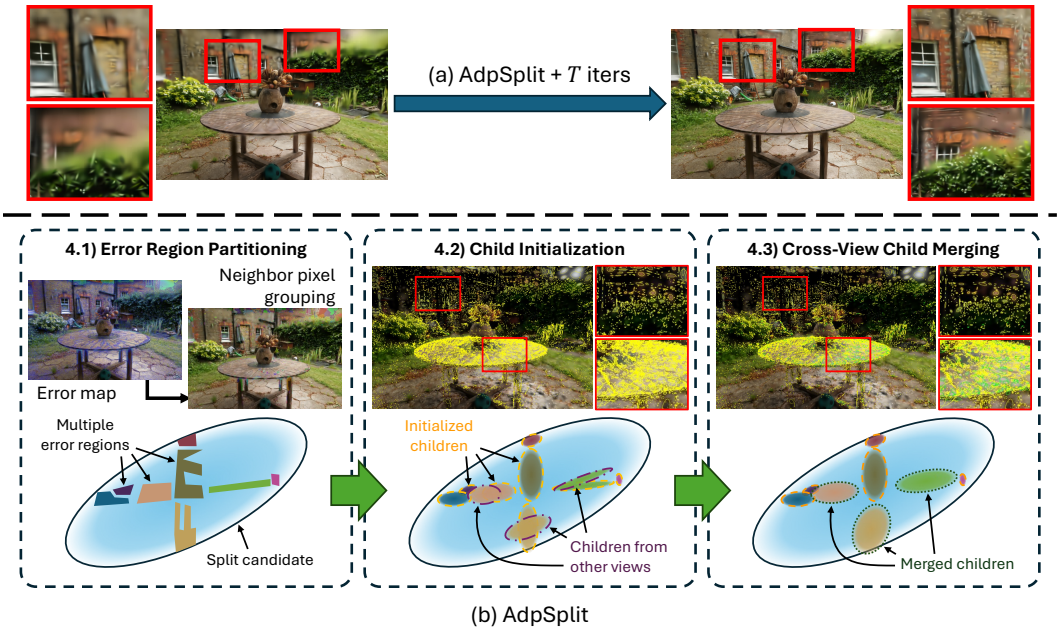

Figure 3: Overview of AdpSplit. Our method generates more children for a Gaussian when scattered error regions are detected, enabling fine scene structures to be discovered in earlier training iterations (a). Neighboring high-error pixels attributed to the same split candidate are grouped as an error region, whose statistics initialize a child Gaussian. To eliminate redundant children from the same error portion, children from different views with similar positions and colors are merged (b).

Then, we compute a min-max-normalized per-pixel L1 error map:

$$\mathcal{E}_v(u) = \frac{|\hat{\mathbf{I}}_v(u) - \mathbf{I}_v^{\mathrm{gt}}(u)|_1 - \mathcal{E}_{\min}}{\mathcal{E}_{\max} - \mathcal{E}_{\min}}. \tag{6}$$

where $\mathcal{E}_{\min}$ and $\mathcal{E}_{\max}$ denote the minimum and maximum values of $|\hat{\mathbf{I}}_v(u) - \mathbf{I}_v^{\mathrm{gt}}(u)|_1$ over all pixels in view v. A binary metric map $\mathcal{M}_v(u) = \mathbf{1}[\mathcal{E}_v(u) > \tau_{\mathrm{l1}}]$ marks high-error pixels, where τ_{l1} is the normalized L1-error threshold. Optionally, an erosion filter with length r_{erode} removes isolated noisy pixels before error-region grouping.

We stratify the normalized error values into L error-level bands over $[\tau_{\mathrm{l1}}, 1]$, and denote the resulting band index by $B_v(u)$. For every split candidate $\mathcal{G}_i$ with $i \in \mathcal{S}$ and sampled view v, distinct error regions are defined as maximal groups of neighboring high-error pixels that are dominated by the same candidate and share the same error band. Formally, the q-th error region for candidate i in view v is

$$\mathcal{R}_v^q = \{u \mid \mathcal{M}_v(u) = 1,\ D_v(u) = i,\ u \sim_8 u^q,\ B_v(u) = B_v(u^q)\}, \tag{7}$$

where $u \sim_8 u^q$ means that u can be reached from the seed pixel u^q through a chain of 8-neighbor pixels satisfying the same conditions, and u^q satisfies $\mathcal{M}_v(u^q) = 1$ and $D_v(u^q) = i$. The region area is defined as $\mathcal{A}_v^q = |\mathcal{R}_v^q|$, and only regions satisfying

$$\mathcal{A}_v^q \geq m_{\min} \tag{8}$$

are retained.

For every retained region $\mathcal{R}_v^q$ of candidate $\mathcal{G}_i$, we compute region statistics: centroid $(\bar{x}_v^q, \bar{y}_v^q)$, area $\mathcal{A}_v^q$, dominant PCA direction $\mathbf{e}_{1,v}^q = (e_{1x,v}^q, e_{1y,v}^q)$, orthogonal direction $\mathbf{e}_{2,v}^q = (-e_{1y,v}^q, e_{1x,v}^q)$, and standard deviations $(\sigma_{1,v}^q, \sigma_{2,v}^q)$ of $\mathcal{R}_v^q$, as well as the ground-truth RGB value $\mathbf{r}_v^q$ at the centroid. These statistics are the sufficient information used to initialize 3D children.

4.2 Child Initialization

Each attributed error region proposes one child Gaussian. The child is initialized from the region's image-space position, shape, and color, while the parent supplies a depth prior.

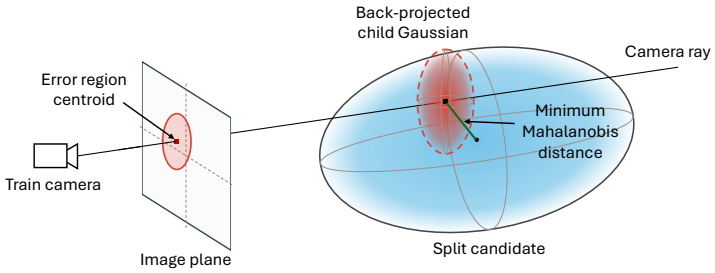

Figure 4: Child initialization from an attributed 2D error region. The region centroid defines a camera ray, and the child center is placed at the closest point to the parent center in the parent's Mahalanobis metric. The region covariance determines the child's rotations and scales.

To compute the child center $\boldsymbol{\mu}_v^q$, we back-project the centroid $(\bar{x}_v^q, \bar{y}_v^q)$ along its camera ray

$$\boldsymbol{\ell}_v^q(t) = \mathbf{o}_v + t\hat{\mathbf{d}}_v^q, \tag{9}$$

where $\mathbf{o}_v$ is the camera center and $\hat{\mathbf{d}}_v^q$ is the unit ray direction in world coordinates. We place the child at the point on this ray that minimizes the Mahalanobis distance to the parent center, so that the placement respects the parent's anisotropic Gaussian shape (Figure 4):

$$t_v^* = \arg\min_t \left(\boldsymbol{\ell}_v^q(t) - \boldsymbol{\mu}_i\right)^\top \boldsymbol{\Sigma}_i^{-1} \left(\boldsymbol{\ell}_v^q(t) - \boldsymbol{\mu}_i\right) = \frac{(\boldsymbol{\mu}_i - \mathbf{o}_v)^\top \boldsymbol{\Sigma}_i^{-1} \hat{\mathbf{d}}_v^q}{(\hat{\mathbf{d}}_v^q)^\top \boldsymbol{\Sigma}_i^{-1} \hat{\mathbf{d}}_v^q + \epsilon}, \tag{10}$$

where ϵ is a small constant for numerical stability. We then initialize the child parameters as

$$\begin{aligned} &\boldsymbol{\mu}_v^q = \boldsymbol{\ell}_v^q(t_v^*), \quad \mathbf{R}_v^q = [\hat{\mathbf{u}}_1 \mid \hat{\mathbf{u}}_2 \mid \mathbf{z}_v], \\ &\mathbf{s}_v^q = \left(\|\mathbf{a}_{1,v}^q\|, \|\mathbf{a}_{2,v}^q\|, \|\mathbf{a}_{2,v}^q\|\right), \quad \mathbf{c}_v^q = \mathbf{r}_v^q, \quad o_v^q = o_i, \end{aligned} \tag{11}$$

where $\hat{\mathbf{u}}_1$ and $\hat{\mathbf{u}}_2$ are orthonormal in-plane axes obtained from the unprojected PCA axes $\mathbf{a}_{1,v}^q$ and $\mathbf{a}_{2,v}^q$, $\mathbf{z}_v$ is the camera forward direction, and the opacity is inherited from the parent. To conservatively initialize the child, we set its thickness along the camera forward direction to the minimum scale in image-view space, i.e., $\|\mathbf{a}_{2,v}^q\|$. Appendix A gives the full initialization details.

4.3 Cross-View Child Merging

The same underfit portion of a Gaussian can be visible in several sampled views, so the previous stage may produce redundant child proposals for one parent. We merge compatible proposals before inserting them into the Gaussian set $\mathcal{G}$.

For each parent independently, two child proposals $\mathcal{G}^{(i)}$ and $\mathcal{G}^{(j)}$ are connected if their symmetric Mahalanobis distance is below the distance threshold γ_d and their maximum RGB-channel difference is below the color threshold γ_c:

$$\sqrt{\boldsymbol{\delta}^\top \boldsymbol{\Sigma}_i^{-1} \boldsymbol{\delta}} + \sqrt{\boldsymbol{\delta}^\top \boldsymbol{\Sigma}_j^{-1} \boldsymbol{\delta}} \le \gamma_d \quad \text{and} \quad \|\mathbf{c}^{(i)} - \mathbf{c}^{(j)}\|_\infty \le \gamma_c, \tag{12}$$

where $\boldsymbol{\delta} = \boldsymbol{\mu}^{(j)} - \boldsymbol{\mu}^{(i)}$.

We group connected proposals in the merge graph, so merge eligibility is transitive. For a merged group $\mathcal{M}$ with $M = |\mathcal{M}|$ proposals, the center and color are the member averages. The covariance is chosen to enclose all member ellipsoids. Let $\mathbf{E}$ be the eigenvectors of the mean covariance $\bar{\boldsymbol{\Sigma}}_\mathcal{M} = \frac{1}{M}\sum_{m\in\mathcal{M}} \boldsymbol{\Sigma}^{(m)}$. Along each axis $\mathbf{e}_r$ of $\mathbf{E}$, we set the merged variance by the maximum member reach:

$$\begin{aligned} &\boldsymbol{\mu}_\mathcal{M} = \frac{1}{M}\sum_{m\in\mathcal{M}} \boldsymbol{\mu}^{(m)}, \quad \boldsymbol{\Sigma}_\mathcal{M} = \mathbf{E}\,\mathrm{diag}(\boldsymbol{\lambda}_\mathcal{M})\,\mathbf{E}^\top, \quad \mathbf{c}_\mathcal{M} = \frac{1}{M}\sum_{m\in\mathcal{M}} \mathbf{c}^{(m)}, \\ &\lambda_{\mathcal{M},r} = \max_{m\in\mathcal{M}} \left(|\mathbf{e}_r^\top (\boldsymbol{\mu}^{(m)} - \boldsymbol{\mu}_\mathcal{M})| + \sigma_{m,r}\right)^2, \quad r \in \{1,2,3\}. \end{aligned} \tag{13}$$

where $\sigma_{m,r}$ is the 1σ extent of member m along $\mathbf{e}_r$.

After merging, each split candidate has a final set of child proposals. To avoid letting a single parent introduce excessive children, we sort its children by covariance extent and keep the largest $N_{\max}$ proposals.

5 Experiments

5.1 Setup

We integrate AdpSplit into 3DGS, various 3DGS training accelerators, and methods designed for challenging scenarios, and evaluate its impact on training time and reconstruction quality.

Metrics. We report novel view synthesis quality using PSNR (dB), SSIM, and LPIPS on the test views. To measure training efficiency, we also report wall-clock training time in seconds, rendering FPS, and the final number of Gaussians.

Datasets. We evaluate 3DGS and its training accelerators on the standard real-world benchmarks: MipNeRF360 [Barron et al., 2022], Tanks&Temples [Knapitsch et al., 2017], and Deep-Blending [Hedman et al., 2018]. Following the 3DGS protocol [Kerbl et al., 2023], images are resized and split into training and test views. We use NeRF-DS [Yan et al., 2023], NeRF on-the-go [Ren et al., 2024], and RobustNeRF [Sabour et al., 2023] to evaluate methods for challenging scenarios.

Baselines. We select 3DGS [Kerbl et al., 2023], MiniSplatting [Fang and Wang, 2024], Dash-Gaussian [Chen et al., 2025c], SpeedySplat [Hanson et al., 2025a], FastGS [Ren et al., 2026], Deformable3DGS [Yang et al., 2024], and RobustSplat [Fu et al., 2025] as our baselines. We train each using its original training schedule. We also train each with a shortened densification schedule, denoted by -S (densification stops at 9k iterations, followed by continued optimization until 24k iterations). We apply AdpSplit on top of the shortened densification variants, and denote the variants by -AdpSplit. For the shortened schedules of DashGaussian, Deformable3DGS, and RobustSplat, we use 19k, 9k, and 14k densification iterations followed by 5k, 5k, and 10k fine-tuning iterations, respectively, with the same -S and -AdpSplit suffixes.

Implementation. AdpSplit is invoked only inside the densification step and leaves the baseline rendering loss and optimizer unchanged. Following the baselines' settings, the densification interval is $T = 100$ for all methods except FastGS, for which it is $T = 500$. For MiniSplatting-S and MiniSplatting-AdpSplit, we use `reinit_interval 3000`, `simp_iteration1 9000`, and `simp_iteration2 14000` to adapt the original settings to the shortened densification schedule.

For AdpSplit, we set the L1-error threshold $\tau_{\text{l1}} = 0.1$, erosion length $r_{\text{erode}} = 2$, minimum error-region area $m_{\min} = 5$, number of error-level bands $L = 3$, maximum number of children per parent $N_{\max} = 19$, and number of sampled views $V = 20$ by default. For outdoor scenes from MipNeRF360, the `Playroom` scene from Deep-Blending, and scenes from NeRF on-the-go, we reduce the child cap and sampled-view count to $N_{\max} = 9$ and $V = 10$, respectively. When merging cross-view children, we use $\gamma_d = 2.0$ and $\gamma_c = 0.15$. The split thresholds τ_s and τ_g, along with all other hyperparameters, follow the corresponding baseline defaults.

Our method generates an adaptive set of N_i split children with proper initial parameter estimates, according to the available error evidence for split candidates. When appending the estimated children to the Gaussian set, we additionally include a parent copy with opacity reduced by a factor of $N_i + 1$ and scale unchanged to cover error-free regions. If no sampled view attributes error pixels to the candidate (when the Gaussian has never been dominant in the sampled views), or no valid error region survives after pixel erosion filtering and minimum error-region area filtering, the candidate falls back to the vanilla split with $N = 2$ children. In the latter case, when error pixels were attributed, we additionally retain the unchanged copy of the parent with its Adam moments reset to zero.

We modified the CUDA rasterizer of 3DGS [Kerbl et al., 2023] to use the compact-box method [Ren et al., 2026] for rasterizing Gaussians and to additionally output dominant Gaussian maps for sampled views. We implemented a CUDA extension for projecting split candidates and collecting high-error pixels inside each candidate footprint, and utilized the `cuCIM` library to group neighboring error

Table 1: Average reconstruction quality and efficiency on the three benchmarks. Time denotes training time in seconds, Δt denotes the percentage change in training time from the baseline of each triple, and #G. denotes the number of Gaussians in millions. Bold indicates the best value within each baseline triple.

Method	MipNeRF360						Deep-Blending						Tanks&Temples					
	PSNR	SSIM	LPIPS	Time	Δt	#G.	PSNR	SSIM	LPIPS	Time	Δt	#G.	PSNR	SSIM	LPIPS	Time	Δt	#G.
3DGS	27.53	**0.813**	**0.221**	1807	–	2.65	29.68	**0.907**	**0.239**	1676	–	2.48	23.72	**0.852**	**0.169**	967	–	1.58
3DGS-S	27.41	0.808	0.228	**1318**	**-27.1**	**2.41**	**29.87**	0.906	0.243	**1152**	**-31.3**	**2.21**	23.69	0.850	0.174	**707**	**-26.9**	**1.47**
3DGS-AdpSplit	**27.54**	0.810	0.225	1391	-23.0	2.55	29.80	**0.907**	0.240	1197	-28.6	2.25	**23.83**	0.850	0.172	748	-22.6	1.51
MiniSplatting	**27.32**	**0.822**	**0.217**	1172	–	0.49	29.97	0.910	**0.241**	1050	–	0.55	**23.40**	**0.846**	**0.181**	723	–	0.30
MiniSplatting-S	27.27	0.821	0.219	**898**	**-23.4**	**0.48**	**30.07**	**0.911**	0.242	**816**	**-22.3**	**0.53**	**23.40**	0.843	0.186	**527**	**-27.1**	**0.28**
MiniSplatting-AdpSplit	**27.32**	0.820	0.220	920	-21.5	**0.48**	29.97	0.910	0.242	828	-21.1	**0.53**	23.27	0.838	0.192	562	-22.3	**0.28**
DashGaussian	**27.71**	**0.820**	**0.213**	594	–	2.43	**29.79**	**0.905**	**0.246**	358	–	1.95	24.00	**0.853**	**0.178**	349	–	1.21
DashGaussian-S	27.63	0.819	0.214	**499**	**-16.0**	2.40	29.72	0.904	0.248	**314**	**-12.3**	**1.82**	23.94	0.850	0.181	**292**	**-16.3**	1.17
DashGaussian-AdpSplit	27.70	**0.820**	**0.213**	522	-12.1	**2.36**	29.71	0.904	0.248	325	-9.2	1.85	**24.12**	0.851	0.179	304	-12.9	**1.15**
SpeedySplat	26.86	0.781	0.295	1090	–	**0.30**	29.63	0.903	0.268	841	–	**0.25**	**23.51**	0.820	0.240	500	–	**0.18**
SpeedySplat-S	27.03	0.787	0.283	**868**	**-20.4**	0.34	**29.64**	**0.904**	**0.266**	**649**	**-22.8**	0.27	23.46	0.823	0.234	**395**	**-20.9**	0.20
SpeedySplat-AdpSplit	**27.10**	**0.790**	**0.279**	912	-16.3	0.35	29.62	0.903	**0.266**	677	-19.5	0.27	23.49	**0.824**	**0.233**	422	-15.5	0.20
FastGS	27.54	**0.798**	0.261	171	–	0.40	29.90	**0.905**	**0.267**	105	–	0.22	**24.26**	**0.842**	**0.208**	108	–	0.24
FastGS-S	27.37	0.790	0.272	**127**	**-25.7**	**0.31**	29.80	0.901	0.274	**78**	**-25.7**	**0.17**	23.99	0.837	0.214	**80**	**-25.9**	**0.21**
FastGS-AdpSplit	**27.55**	0.797	**0.260**	143	-16.4	0.40	**29.95**	0.904	0.269	83	-21.0	0.19	24.17	0.840	0.209	86	-20.4	0.23

pixels into error regions. The rest of AdpSplit is implemented in PyTorch. Our code is available at: `https://github.com/ASU-ESIC-FAN-Lab/AdpSplit`.

5.2 Results

In this section, we empirically demonstrate the effectiveness of AdpSplit by comparing baselines with AdpSplit on top of the baselines, and examine the effect of each stage by progressively ablating them. Experiments are conducted using an NVIDIA RTX 5000 Ada GPU in Section 5.2.1 and Section 5.2.3, and an NVIDIA A100 GPU in Section 5.2.2. For more visualizations and hyperparameter sensitivity analysis, please refer to Appendix B.

5.2.1 Improving existing efficient training methods

Table 1 shows that shortened densification schedules consistently reduce training time, but often at the cost of reconstruction quality. This trade-off is especially visible for accelerated training pipelines. On MipNeRF360, FastGS-S reduces training time by 25.7% but also decreases PSNR from 27.54 to 27.37. Similar quality drops appear for FastGS on Deep-Blending and Tanks&Temples, indicating that simply removing densification rounds can leave scene structure under-modeled even when split candidates are carefully selected.

AdpSplit recovers much of this lost quality while preserving most of the speedup from the shortened schedules. For 3DGS on MipNeRF360, the shortened schedule reduces PSNR by 0.12 dB, whereas 3DGS-AdpSplit restores the PSNR to 27.54 while still reducing training time by 23.0% relative to the full 3DGS baseline. The same trend holds for FastGS: FastGS-AdpSplit matches the full FastGS PSNR on MipNeRF360, improves LPIPS from 0.261 to 0.260, and trains in only 143 seconds, corresponding to a 16.4% time reduction. Across Deep-Blending, AdpSplit maintains comparable SSIM for most baselines while reducing training time by up to 28.6%. Remarkably, FastGS-AdpSplit trains a scene from Deep-Blending in only 83 seconds. On Tanks&Temples, AdpSplit similarly keeps quality metrics close to the full baselines while reducing training time by 12.9% to 22.6% depending on the underlying method.

The final number of Gaussians usually increases slightly relative to the shortened schedule, but remains below the corresponding full densification baseline in most cases. This suggests that AdpSplit does not simply add uncontrolled Gaussians; instead, it allocates children to regions useful for recovering scene structure. One exception is DashGaussian-AdpSplit, which can take longer than DashGaussian-S despite using a comparable or smaller Gaussian count. We attribute this to an overly deep cross-view child-merging graph formed during low-resolution early training. Overall, these results support our claim that error-driven split child generation can recover scene structure otherwise missed when densification is shortened. Notably, our error-driven child initialization prioritizes the reconstruction of small details. As shown in Figure 5, AdpSplit deploys split children not randomly,

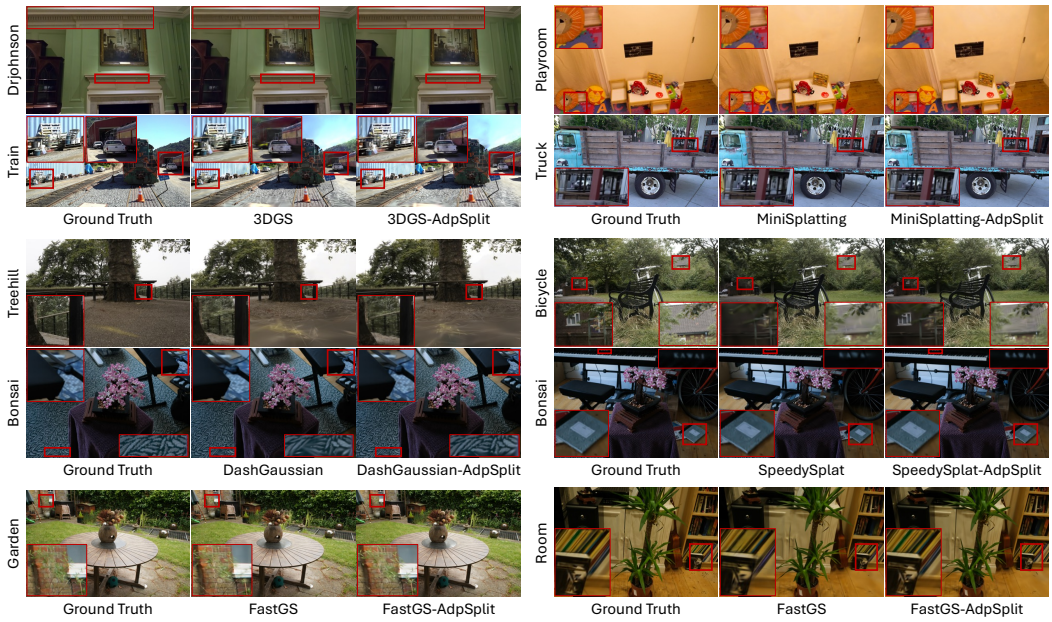

Figure 5: Qualitative results on the scenes from MipNeRF360 [Barron et al., 2022], Deep-Blending [Hedman et al., 2018], and Tanks&Temples [Knapitsch et al., 2017]. Our error-driven child initialization better reconstructs small details that fixed-cardinality random splitting fails to capture effectively, as highlighted by the red boxes.

Table 2: Average reconstruction quality and efficiency on NeRF-DS

Method	PSNR	SSIM	LPIPS	Time	#G.	FPS
Deformable3DGS	23.84	0.852	0.179	662	0.13	75.16
Deformable3DGS-S	23.86	0.850	0.186	410	0.13	78.52
Deformable3DGS-AdpSplit	23.97	0.852	0.186	446	0.14	72.73

but in regions where they effectively reduce view loss, such as the patterns on the book and carpet in the `Bonsai` scene and the railings in the `Treehill` scene.

5.2.2 Improving existing methods designed for challenging datasets

We check generalizability by applying AdpSplit to methods for challenging scenarios. For dynamic scenes, we integrate AdpSplit into Deformable3DGS, which represents Gaussians in a canonical space and learns a time-dependent deformation field. As shown in Table 2, AdpSplit improves PSNR over both the full and shortened variants on NeRF-DS, reaching 23.97 dB while reducing training time from 662 to 446 seconds relative to the full baseline.

For scenes captured in the wild, we integrate AdpSplit into RobustSplat. In Table 3, AdpSplit improves PSNR, SSIM, and LPIPS over the shortened variant on both NeRF on-the-go and RobustNeRF, with a modest increase in training time. Compared with the full baseline, training time decreases from 681 to 470 seconds on NeRF on-the-go and from 723 to 562 seconds on RobustNeRF. PSNR and SSIM improve on both datasets, while LPIPS improves on RobustNeRF but is slightly worse on NeRF on-the-go.

Together, these results suggest that AdpSplit, alongside appropriate base models and mechanisms that address challenges inherent in scenes, still accelerates training.

5.2.3 Ablation study

Table 4 ablates the components of AdpSplit on top of the shortened 3DGS schedule. Randomly placing an adaptive number of children, derived by counting partitioned error regions, recovers part of the lost reconstruction quality, suggesting that a fixed $N = 2$ split cannot effectively search for useful

Table 3: Average reconstruction quality and efficiency on NeRF on-the-go and RobustNeRF

Method	NeRF on-the-go						RobustNeRF					
	PSNR	SSIM	LPIPS	Time	#G.	FPS	PSNR	SSIM	LPIPS	Time	#G.	FPS
RobustSplat	22.13	0.796	0.170	681	0.94	266.79	28.23	0.886	0.145	723	0.54	410.18
RobustSplat-S	22.17	0.796	0.174	462	0.67	341.00	28.18	0.884	0.146	543	0.43	439.21
RobustSplat-AdpSplit	22.22	0.797	0.172	470	0.68	335.20	28.30	0.888	0.144	562	0.46	438.53

Table 4: Ablation results averaged over MipNeRF360 scenes

Method	PSNR	SSIM	LPIPS	Time	#G.	FPS
3DGS	27.53	0.813	0.221	1807	2.65	99.50
3DGS-S	27.41	0.808	0.228	1318	2.41	109.98
+ Adaptive N	27.47	0.810	0.224	1398	2.58	105.48
+ Child initialization	27.49	0.810	0.224	1436	2.63	102.44
+ Child merging (=AdpSplit)	27.54	0.810	0.225	1391	2.55	105.13

scene structure. Child initialization further improves PSNR, but adds more Gaussians to the scene. Child merging obtains the highest PSNR, successfully recovers the PSNR of 3DGS, and removes some redundant Gaussians while preserving the training-time reduction from the shortened schedule.

6 Conclusion

We presented AdpSplit, an error-driven adaptive split operator for accelerating 3D Gaussian Splatting training. Instead of assigning each split candidate a fixed number of randomly placed children, AdpSplit uses attributable L1-pixel-error regions to initialize a dynamic number of children and their parameters, then merges redundant cross-view proposals before insertion. Because AdpSplit only replaces the split operator, it can be integrated into existing 3DGS acceleration pipelines as a drop-in replacement. Our experiments show that AdpSplit recovers much of the quality lost by shortened densification schedules while retaining most of their speedup. Across 3DGS, MiniSplatting, DashGaussian, SpeedySplat, and FastGS on three benchmarks, AdpSplit consistently maintains or improves the quality-efficiency trade-off and reconstructs small details often missed by random child placement. These results suggest that adaptive, error-driven child generation can reduce the densification rounds needed for 3DGS scene discovery.

Limitations. AdpSplit is effective when densification is limited to 9k iterations, but more elaborate splitting strategies may further reduce the required number of densification steps. Since our error-driven child initialization starts from the first densification round, overly large early Gaussians can cover broad scene regions and may generate floating children from unreliable error regions. Future work could add early-stage safeguards or extend adaptive splitting to more various dynamic and sparse-view reconstruction settings.

Acknowledgments and Disclosure of Funding

Supported by the Intelligence Advanced Research Projects Activity (IARPA) via Department of Interior/ Interior Business Center (DOI/IBC) contract number 140D0423C0076. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. Disclaimer: The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of IARPA, DOI/IBC, or the U.S. Government.

References

Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: representing scenes as neural radiance fields for view synthesis. *Commun. ACM*, 65(1):99–106, December 2021. ISSN 0001-0782. doi: 10.1145/3503250.

Bernhard Kerbl, Georgios Kopanas, Thomas Leimkuehler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Trans. Graph.*, 42(4), July 2023. ISSN 0730-0301. doi: 10.1145/3592433.

Chi Yan, Delin Qu, Dan Xu, Bin Zhao, Zhigang Wang, Dong Wang, and Xuelong Li. Gs-slam: Dense visual slam with 3d gaussian splatting. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 19595–19604, 2024. doi: 10.1109/CVPR52733.2024.01853.

Guanjun Wu, Taoran Yi, Jiemin Fang, Lingxi Xie, Xiaopeng Zhang, Wei Wei, Wenyu Liu, Qi Tian, and Xinggang Wang. 4d gaussian splatting for real-time dynamic scene rendering. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 20310–20320, 2024. doi: 10.1109/CVPR52733.2024.01920.

Jiaxiang Tang, Jiawei Ren, Hang Zhou, Ziwei Liu, and Gang Zeng. Dreamgaussian: Generative gaussian splatting for efficient 3d content creation. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun, editors, *International Conference on Learning Representations*, volume 2024, pages 33879–33896, 2024. URL `https://proceedings.iclr.cc/paper_files/paper/2024/file/905202e21386913d8eac637c2b50f590-Paper-Conference.pdf`.

Shenhan Qian, Tobias Kirschstein, Liam Schoneveld, Davide Davoli, Simon Giebenhain, and Matthias Nießner. Gaussianavatars: Photorealistic head avatars with rigged 3d gaussians. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 20299–20309, 2024. doi: 10.1109/CVPR52733.2024.01919.

Noah Snavely, Steven M. Seitz, and Richard Szeliski. Photo tourism: exploring photo collections in 3d. In *ACM SIGGRAPH 2006 Papers*, SIGGRAPH '06, page 835–846, New York, NY, USA, 2006. Association for Computing Machinery. ISBN 1595933646. doi: 10.1145/1179352.1141964.

Sharath Girish, Kamal Gupta, and Abhinav Shrivastava. Eagles: Efficient accelerated 3d gaussians with lightweight encodings. In Aleš Leonardis, Elisa Ricci, Stefan Roth, Olga Russakovsky, Torsten Sattler, and Gül Varol, editors, *Computer Vision – ECCV 2024*, pages 54–71, Cham, 2025. Springer Nature Switzerland. ISBN 978-3-031-73036-8. doi: 10.1007/978-3-031-73036-8_4.

Joo Chan Lee, Daniel Rho, Xiangyu Sun, Jong Hwan Ko, and Eunbyung Park. Compact 3d gaussian representation for radiance field. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 21719–21728, 2024. doi: 10.1109/CVPR52733.2024.02052.

Zhaoliang Zhang, Tianchen Song, Yongjae Lee, Li Yang, Cheng Peng, Rama Chellappa, and Deliang Fan. Lp-3dgs: Learning to prune 3d gaussian splatting. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, *Advances in Neural Information Processing Systems*, volume 37, pages 122434–122457. Curran Associates, Inc., 2024. doi: 10.52202/079017-3891.

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 *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 20654–20664, 2024. doi: 10.1109/CVPR52733.2024.01952.

Alex Hanson, Allen Tu, Geng Lin, Vasu Singla, Matthias Zwicker, and Tom Goldstein. Speedy-splat: Fast 3d gaussian splatting with sparse pixels and sparse primitives. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 21537–21546, 2025a. doi: 10.1109/CVPR52734.2025.02006.

Guangchi Fang and Bing Wang. Mini-splatting: Representing scenes with a constrained number of gaussians. In Aleš Leonardis, Elisa Ricci, Stefan Roth, Olga Russakovsky, Torsten Sattler, and Gül Varol, editors, *Computer Vision – ECCV 2024*, pages 165–181, Cham, 2024. Springer Nature Switzerland. ISBN 978-3-031-72980-5. doi: 10.1007/978-3-031-72980-5_10.

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 Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp, editors, *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 8123–8140. PMLR, 21–27 Jul 2024. URL `https://proceedings.mlr.press/v235/cheng24f.html`.

Peihao Wang, Yuehao Wang, Dilin Wang, Sreyas Mohan, Zhiwen Fan, Lemeng Wu, Ruisi Cai, Yu-Ying Yeh, Zhangyang Wang, Qiang Liu, and Rakesh Ranjan. Steepest descent density control for compact 3d gaussian splatting. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 26663–26672, 2025. doi: 10.1109/CVPR52734.2025.02483.

Shiwei Ren, Tianci Wen, Yongchun Fang, and Biao Lu. Fastgs: Training 3d gaussian splatting in 100 seconds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 26094–26103, June 2026.

Zhiwen Fan, Kevin Wang, Kairun Wen, Zehao Zhu, Dejia Xu, and Zhangyang Wang. Lightgaussian: Unbounded 3d gaussian compression with 15x reduction and 200+ fps. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, *Advances in Neural Information Processing Systems*, volume 37, pages 140138–140158. Curran Associates, Inc., 2024. doi: 10.52202/079017-4447.

Alex Hanson, Allen Tu, Vasu Singla, Mayuka Jayawardhana, Matthias Zwicker, and Tom Goldstein. Pup 3d-gs: Principled uncertainty pruning for 3d gaussian splatting. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5949–5958, 2025b. doi: 10.1109/CVPR52734.2025.00558.

Wieland Morgenstern, Florian Barthel, Anna Hilsmann, and Peter Eisert. Compact 3d scene representation via self-organizing gaussian grids. In Aleš Leonardis, Elisa Ricci, Stefan Roth, Olga Russakovsky, Torsten Sattler, and Gül Varol, editors, *Computer Vision – ECCV 2024*, pages 18–34, Cham, 2025. Springer Nature Switzerland. ISBN 978-3-031-73013-9. doi: 10.1007/978-3-031-73013-9_2.

Simon Niedermayr, Josef Stumpfegger, and Rüdiger Westermann. Compressed 3d gaussian splatting for accelerated novel view synthesis. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10349–10358, 2024. doi: 10.1109/CVPR52733.2024.00985.

Michael Niemeyer, Fabian Manhardt, Marie-Julie Rakotosaona, Michael Oechsle, Daniel Duckworth, Rama Gosula, Keisuke Tateno, John Bates, Dominik Kaeser, and Federico Tombari. Radsplat: Radiance field-informed gaussian splatting for robust real- time rendering with 900+ fps. In *2025 International Conference on 3D Vision (3DV)*, pages 134–144, 2025. doi: 10.1109/3DV66043.2025.00018.

Kerui Ren, Lihan Jiang, Tao Lu, Mulin Yu, Linning Xu, Zhangkai Ni, and Bo Dai. Octree-gs: Towards consistent real-time rendering with lod-structured 3d gaussians. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–15, 2025. doi: 10.1109/TPAMI.2025.3568201.

Yihang Chen, Qianyi Wu, Weiyao Lin, Mehrtash Harandi, and Jianfei Cai. Hac: Hash-grid assisted context for 3d gaussian splatting compression. In Aleš Leonardis, Elisa Ricci, Stefan Roth, Olga Russakovsky, Torsten Sattler, and Gül Varol, editors, *Computer Vision – ECCV 2024*, pages 422–438, Cham, 2025a. Springer Nature Switzerland. ISBN 978-3-031-72667-5. doi: 10.1007/978-3-031-72667-5_24.

Yihang Chen, Qianyi Wu, Weiyao Lin, Mehrtash Harandi, and Jianfei Cai. Hac++: Towards 100x compression of 3d gaussian splatting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 47(11):10210–10226, 2025b. doi: 10.1109/TPAMI.2025.3594066.

Samuel Rota Bulò, Lorenzo Porzi, and Peter Kontschieder. Revising densification in gaussian splatting. In Aleš Leonardis, Elisa Ricci, Stefan Roth, Olga Russakovsky, Torsten Sattler, and Gül Varol, editors, *Computer Vision – ECCV 2024*, pages 347–362, Cham, 2025. Springer Nature Switzerland. ISBN 978-3-031-73036-8. doi: 10.1007/978-3-031-73036-8_20.

Shakiba Kheradmand, Daniel Rebain, Gopal Sharma, Weiwei Sun, Yang-Che Tseng, Hossam Isack, Abhishek Kar, Andrea Tagliasacchi, and Kwang Moo Yi. 3d gaussian splatting as markov chain monte carlo. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, *Advances in Neural Information Processing Systems*, volume 37, pages 80965–80986. Curran Associates, Inc., 2024. doi: 10.52202/079017-2573.

Youyu Chen, Junjun Jiang, Kui Jiang, Xiao Tang, Zhihao Li, Xianming Liu, and Yinyu Nie. Dashgaussian: Optimizing 3d gaussian splatting in 200 seconds. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11146–11155, 2025c. doi: 10.1109/CVPR52734.2025.01041.

Bartlomiej Baranowski, Stefano Esposito, Patricia Gschoßmann, Anpei Chen, and Andreas Geiger. Conegs: Error-guided densification using pixel cones for improved reconstruction with fewer primitives. In *2026 International Conference on 3D Vision (3DV)*, pages 989–999, 2026. doi: 10.1109/3DV69130.2026.00099.

Ao Gao, Luosong Guo, Tao Chen, Zhao Wang, Ying Tai, Jian Yang, and Zhenyu Zhang. Easysplat: View-adaptive learning makes 3d gaussian splatting easy. In *2025 IEEE International Conference on Multimedia and Expo (ICME)*, pages 1–6, 2025. doi: 10.1109/ICME59968.2025.11209122.

Jonathan T. Barron, Ben Mildenhall, Dor Verbin, Pratul P. Srinivasan, and Peter Hedman. Mip-nerf 360: Unbounded anti-aliased neural radiance fields. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5460–5469, 2022. doi: 10.1109/CVPR52688.2022.00539.

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.

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.

Zhiwen Yan, Chen Li, and Gim Hee Lee. Nerf-ds: Neural radiance fields for dynamic specular objects. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8285–8295, June 2023. doi: 10.1109/CVPR52729.2023.00801.

Weining Ren, Zihan Zhu, Boyang Sun, Jiaqi Chen, Marc Pollefeys, and Songyou Peng. Nerf on-the-go: Exploiting uncertainty for distractor-free nerfs in the wild. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8931–8940, June 2024. doi: 10.1109/CVPR52733.2024.00853.

Sara Sabour, Suhani Vora, Daniel Duckworth, Ivan Krasin, David J. Fleet, and Andrea Tagliasacchi. Robustnerf: Ignoring distractors with robust losses. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 20626–20636, June 2023. doi: 10.1109/CVPR52729.2023.01976.

Ziyi Yang, Xinyu Gao, Wen Zhou, Shaohui Jiao, Yuqing Zhang, and Xiaogang Jin. Deformable 3d gaussians for high-fidelity monocular dynamic scene reconstruction. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 20331–20341, June 2024. doi: 10.1109/CVPR52733.2024.01922.

Chuanyu Fu, Yuqi Zhang, Kunbin Yao, Guanying Chen, Yuan Xiong, Chuan Huang, Shuguang Cui, and Xiaochun Cao. Robustsplat: Decoupling densification and dynamics for transient-free 3dgs. In *2025 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 27126–27136, Oct 2025. doi: 10.1109/ICCV51701.2025.02518.

A Child Initialization Details

For each retained error region $\mathcal{R}_v^q$ of parent Gaussian $\mathcal{G}_i$, we initialize one child Gaussian from the region's centroid, covariance, and color. Let the camera intrinsics be (f_x, f_y, p_x, p_y), where (f_x, f_y) are the focal lengths and (p_x, p_y) is the image center. The centroid $(\bar{x}_v^q, \bar{y}_v^q)$ defines the camera-space ray direction

$$\mathbf{d}_{\mathrm{cam},v}^q = \begin{bmatrix} (\bar{x}_v^q - p_x)/f_x \\ (\bar{y}_v^q - p_y)/f_y \\ 1 \end{bmatrix}. \tag{14}$$

Given the camera-to-world rotation $\mathbf{R}_{c2w}$, the corresponding unit ray direction in world coordinates is

$$\hat{\mathbf{d}}_v^q = \frac{\mathbf{R}_{c2w}\mathbf{d}_{\mathrm{cam},v}^q}{\|\mathbf{R}_{c2w}\mathbf{d}_{\mathrm{cam},v}^q\|}. \tag{15}$$

The back-projected camera ray is

$$\boldsymbol{\ell}_v^q(t) = \mathbf{o}_v + t\hat{\mathbf{d}}_v^q, \tag{16}$$

where $\mathbf{o}_v$ is the camera center in world coordinates.

We choose the point on this ray that minimizes the Mahalanobis distance to the parent center $\boldsymbol{\mu}_i$:

$$t_v^* = \arg\min_t \left(\boldsymbol{\ell}_v^q(t) - \boldsymbol{\mu}_i\right)^\top \boldsymbol{\Sigma}_i^{-1} \left(\boldsymbol{\ell}_v^q(t) - \boldsymbol{\mu}_i\right). \tag{17}$$

Let $\mathbf{A} = \boldsymbol{\Sigma}_i^{-1}$ and $\mathbf{b} = \boldsymbol{\mu}_i - \mathbf{o}_v$. Substituting the ray equation gives

$$f(t) = \left(t\hat{\mathbf{d}}_v^q - \mathbf{b}\right)^\top \mathbf{A} \left(t\hat{\mathbf{d}}_v^q - \mathbf{b}\right) \tag{18}$$

$$= t^2(\hat{\mathbf{d}}_v^q)^\top \mathbf{A}\hat{\mathbf{d}}_v^q - 2t\mathbf{b}^\top \mathbf{A}\hat{\mathbf{d}}_v^q + \mathbf{b}^\top \mathbf{A}\mathbf{b}. \tag{19}$$

Taking the derivative with respect to t and setting it to zero yields

$$\frac{\partial f}{\partial t} = 2t\,(\hat{\mathbf{d}}_v^q)^\top \mathbf{A}\hat{\mathbf{d}}_v^q - 2\mathbf{b}^\top \mathbf{A}\hat{\mathbf{d}}_v^q = 0. \tag{20}$$

Therefore,

$$t_v^* = \frac{\mathbf{b}^\top \mathbf{A}\hat{\mathbf{d}}_v^q}{(\hat{\mathbf{d}}_v^q)^\top \mathbf{A}\hat{\mathbf{d}}_v^q} = \frac{(\boldsymbol{\mu}_i - \mathbf{o}_v)^\top \boldsymbol{\Sigma}_i^{-1}\hat{\mathbf{d}}_v^q}{(\hat{\mathbf{d}}_v^q)^\top \boldsymbol{\Sigma}_i^{-1}\hat{\mathbf{d}}_v^q}. \tag{21}$$

In implementation, we add a small ϵ to the denominator for numerical stability, giving the expression in Equation 10. The child center is

$$\boldsymbol{\mu}_v^q = \boldsymbol{\ell}_v^q(t_v^*). \tag{22}$$

Next, we convert the image-space region covariance into child scale and orientation. Because $\mathbf{d}^q_{\mathrm{cam},v}$ is normalized when forming $\hat{\mathbf{d}}^q_v$, the camera-space depth associated with t^*_v is

$$t^*_{z,v} = \frac{t^*_v}{\|\mathbf{d}^q_{\mathrm{cam},v}\|}. \tag{23}$$

Let $\mathbf{u}_v$ and $\mathbf{w}_v$ be the camera right and down directions in world coordinates, and let $s^p_{\max} = \max(\mathbf{s}_i)$ be the parent's maximum scale. We clamp the unprojected axis components with $\mathrm{clip}(x, a) = \mathrm{sign}(x)\min(|x|, a)$:

$$w^q_{1x,v} = \mathrm{clip}\left(e^q_{1,x,v}\sigma^q_{1,v}\frac{t^*_{z,v}}{f_x}, s^p_{\max}|e^q_{1,x,v}|\right), \tag{24}$$

$$w^q_{1y,v} = \mathrm{clip}\left(e^q_{1,y,v}\sigma^q_{1,v}\frac{t^*_{z,v}}{f_y}, s^p_{\max}|e^q_{1,y,v}|\right), \tag{25}$$

$$w^q_{2x,v} = \mathrm{clip}\left(e^q_{2,x,v}\sigma^q_{2,v}\frac{t^*_{z,v}}{f_x}, s^p_{\max}|e^q_{2,x,v}|\right), \tag{26}$$

$$w^q_{2y,v} = \mathrm{clip}\left(e^q_{2,y,v}\sigma^q_{2,v}\frac{t^*_{z,v}}{f_y}, s^p_{\max}|e^q_{2,y,v}|\right). \tag{27}$$

The two PCA directions are then unprojected to world-space vectors:

$$\mathbf{a}^q_{1,v} = w^q_{1x,v}\mathbf{u}_v + w^q_{1y,v}\mathbf{w}_v, \tag{28}$$

$$\mathbf{a}^q_{2,v} = w^q_{2x,v}\mathbf{u}_v + w^q_{2y,v}\mathbf{w}_v. \tag{29}$$

The child scales are

$$s^q_{1,v} = \|\mathbf{a}^q_{1,v}\|, \quad s^q_{2,v} = \|\mathbf{a}^q_{2,v}\|, \quad s^q_{3,v} = s^q_{2,v}. \tag{30}$$

Although the PCA directions are orthogonal in image space, their unprojected vectors can lose exact orthogonality after perspective scaling and clamping. We therefore obtain orthonormal in-plane axes by Gram-Schmidt before forming the child rotation matrix:

$$\hat{\mathbf{u}}_1 = \frac{\mathbf{a}^q_{1,v}}{\|\mathbf{a}^q_{1,v}\|}, \quad \hat{\mathbf{u}}_2 = \frac{\mathbf{a}^q_{2,v} - ((\mathbf{a}^q_{2,v})^\top\hat{\mathbf{u}}_1)\hat{\mathbf{u}}_1}{\left\|\mathbf{a}^q_{2,v} - ((\mathbf{a}^q_{2,v})^\top\hat{\mathbf{u}}_1)\hat{\mathbf{u}}_1\right\|}. \tag{31}$$

With camera forward direction $\mathbf{z}_v$, the child covariance is

$$\mathbf{R}^q_v = \left[\hat{\mathbf{u}}_1 \mid \hat{\mathbf{u}}_2 \mid \mathbf{z}_v\right], \quad \mathbf{\Sigma}^q_v = \mathbf{R}^q_v\,\mathrm{diag}((s^q_{1,v})^2, (s^q_{2,v})^2, (s^q_{3,v})^2)\mathbf{R}^{q\top}_v. \tag{32}$$

Finally, we initialize the child color from the ground-truth RGB value $\mathbf{r}^q_v$ at the region centroid, convert it to the DC spherical-harmonic coefficient, and set all higher-order SH coefficients to zero. The child opacity inherits the parent opacity, matching the vanilla split initialization.

B Additional Results

B.1 Peak Memory and More Visualizations

Table 5 reports the peak GPU memory values corresponding to the main results in Table 1. Figure 6–Figure 10 provide an extended visual comparison of Figure 5, including -S variants (i.e., baseline with a shortened densification schedule). Figure 11–Figure 16 present additional visualizations, where the first column shows the baseline method, the second column shows the baseline with a shortened densification schedule, and the last column shows the corresponding AdpSplit variant.

Table 5: Average peak GPU memory (GB) on MipNeRF360, Deep-Blending, and Tanks&Temples

Method	MipNeRF360	Deep-Blending	Tanks&Temples
3DGS	9.93	8.14	4.75
3DGS-S	**9.52**	**7.57**	**4.55**
3DGS-AdpSplit	10.00	7.80	4.77
MiniSplatting	6.33	6.20	4.65
MiniSplatting-S	**6.32**	**6.07**	**4.56**
MiniSplatting-AdpSplit	6.57	6.33	4.83
DashGaussian	8.67	7.75	4.49
DashGaussian-S	**8.61**	7.60	4.44
DashGaussian-AdpSplit	8.65	**7.51**	**4.36**
SpeedySplat	9.01	6.85	**4.50**
SpeedySplat-S	**8.87**	**6.83**	**4.50**
SpeedySplat-AdpSplit	9.50	7.11	4.74
FastGS	7.06	4.98	2.71
FastGS-S	**5.08**	**3.69**	**2.22**
FastGS-AdpSplit	5.83	3.94	2.39



Figure 6: Qualitative results of 3DGS and variants on `Drjohnson` from Deep-Blending and `Train` from Tanks&Temples.

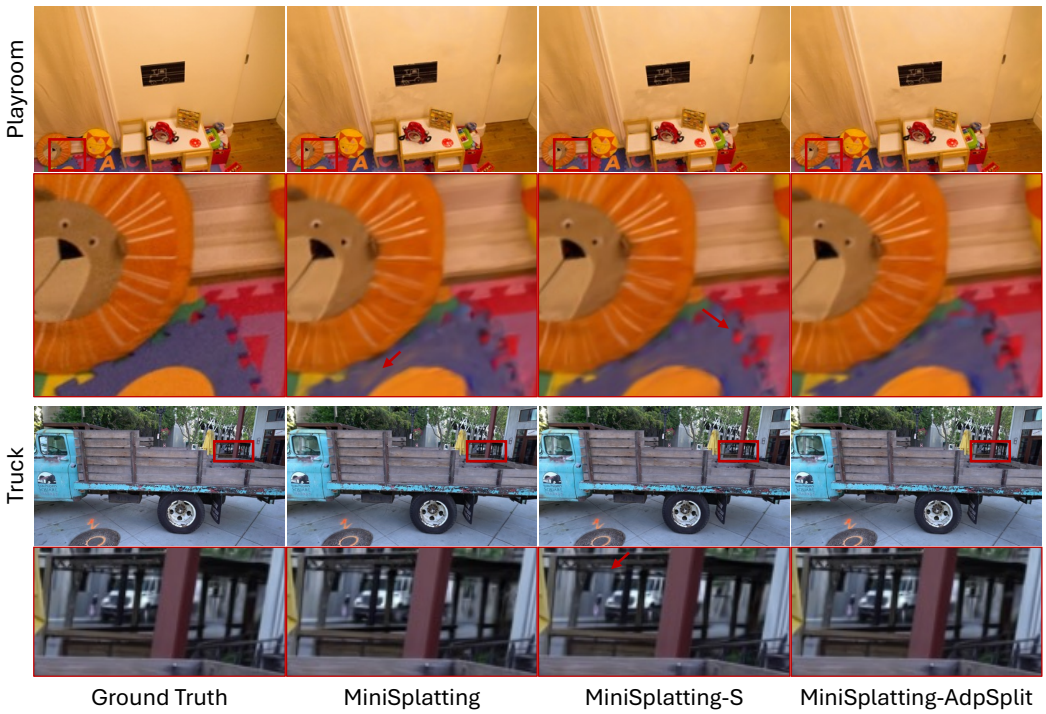

Figure 7: Qualitative results of MiniSplatting and variants on `Playroom` from Deep-Blending and `Truck` from Tanks&Temples.

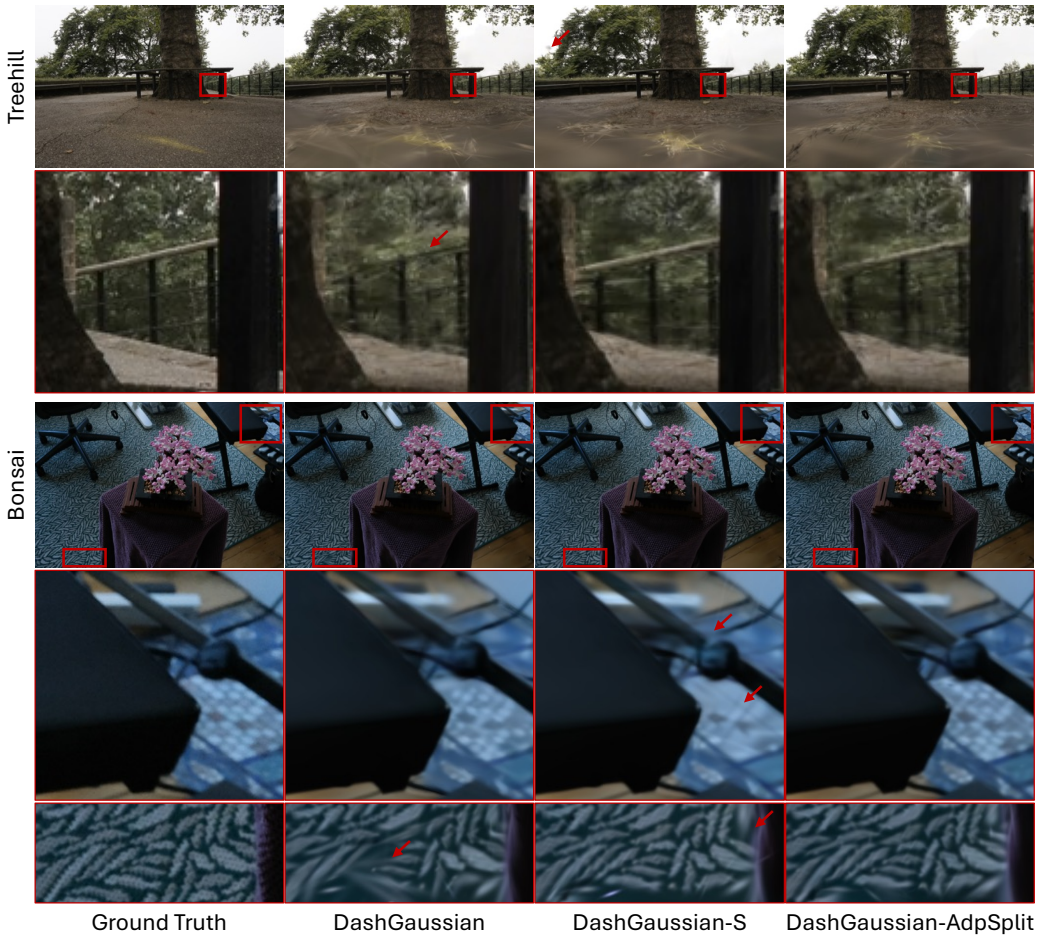

Figure 8: Qualitative results of DashGaussian and variants on `Treehill` and `Bonsai` from MipNeRF360.



Figure 9: Qualitative results of SpeedySplat and variants on `Bicycle` and `Bonsai` from MipNeRF360.

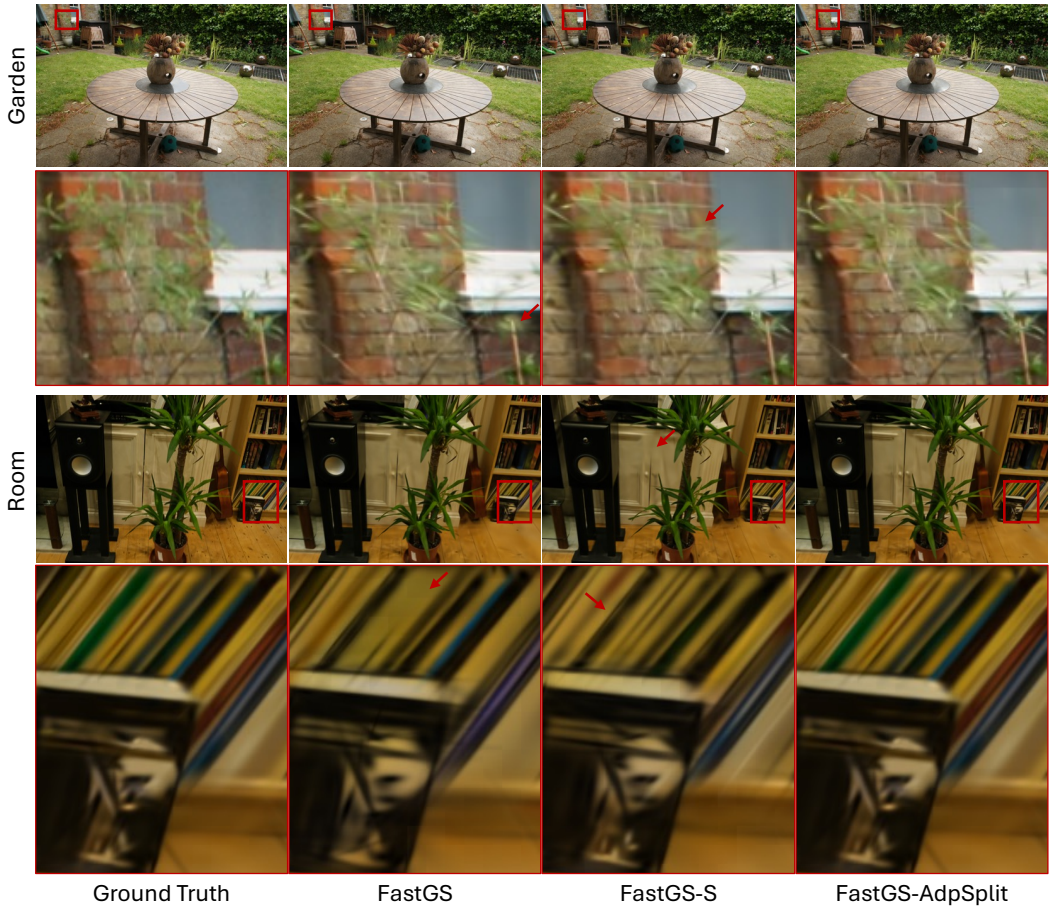

Figure 10: Qualitative results of FastGS and variants on `Garden` and `Room` from MipNeRF360

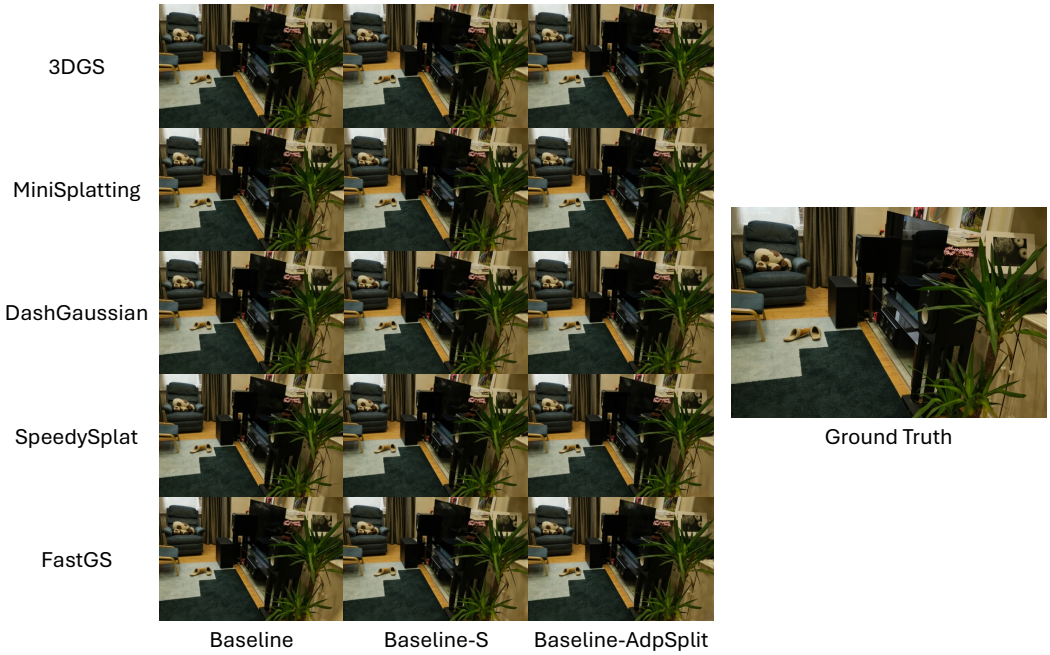

Figure 11: Qualitative results on `Room` from MipNeRF360

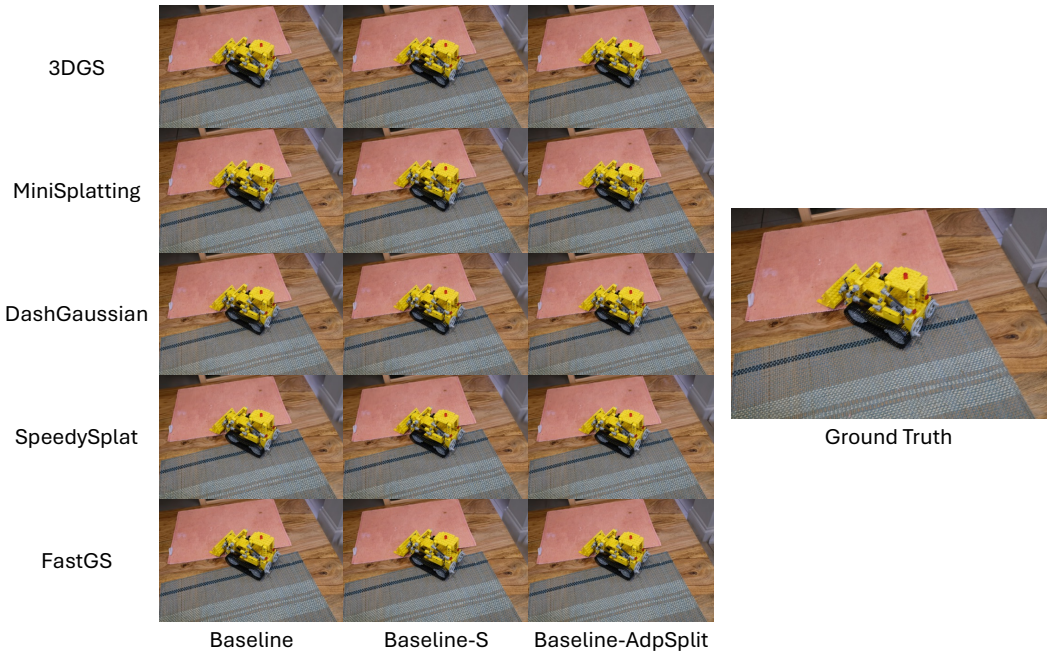

Figure 12: Qualitative results on `Kitchen` from MipNeRF360

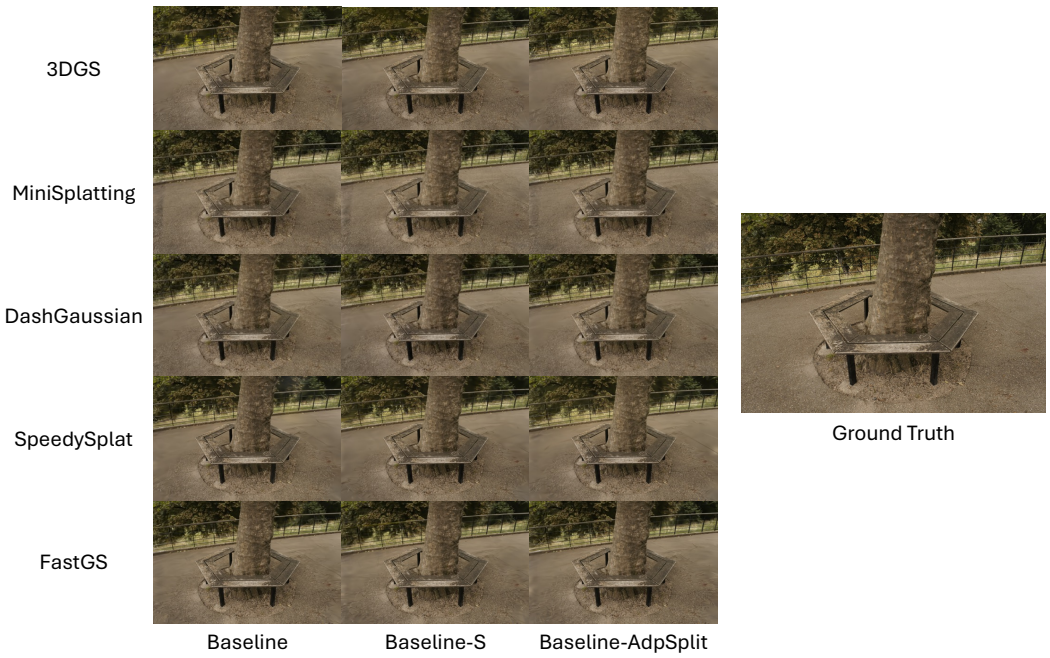

Figure 13: Qualitative results on `Treehill` from MipNeRF360

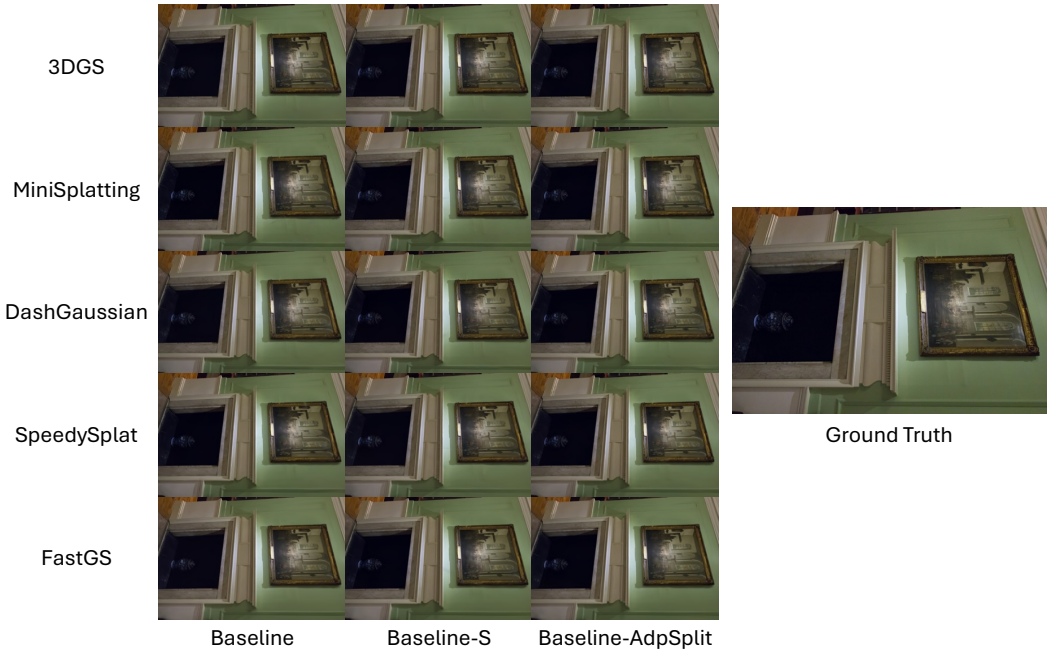

Figure 14: Qualitative results on `Drjohnson` from Deep-Blending

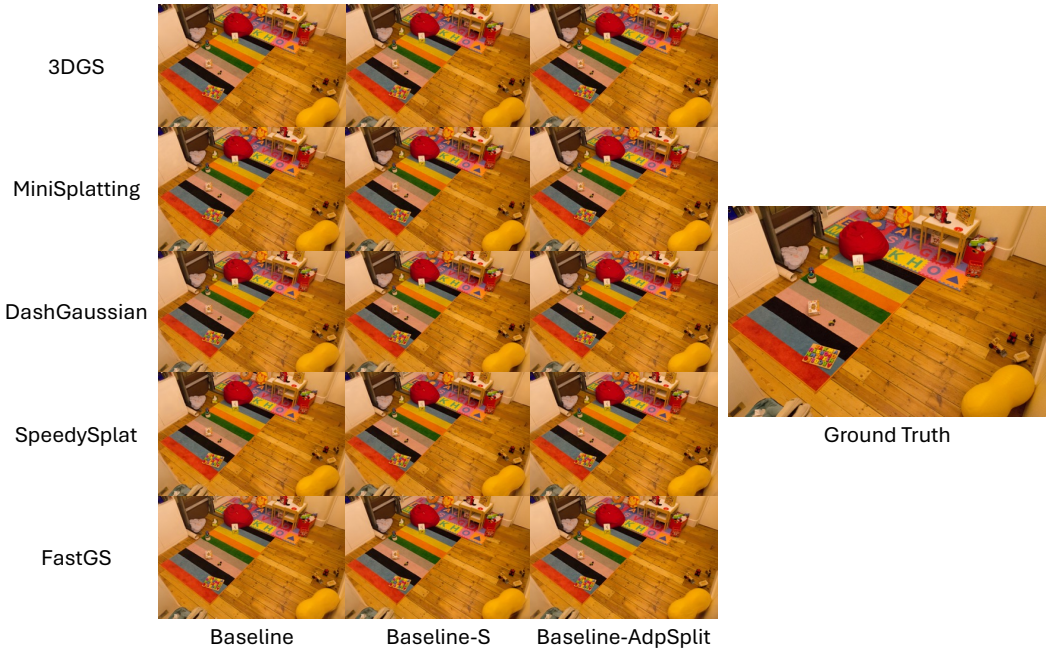

Figure 15: Qualitative results on `Playroom` from Deep-Blending

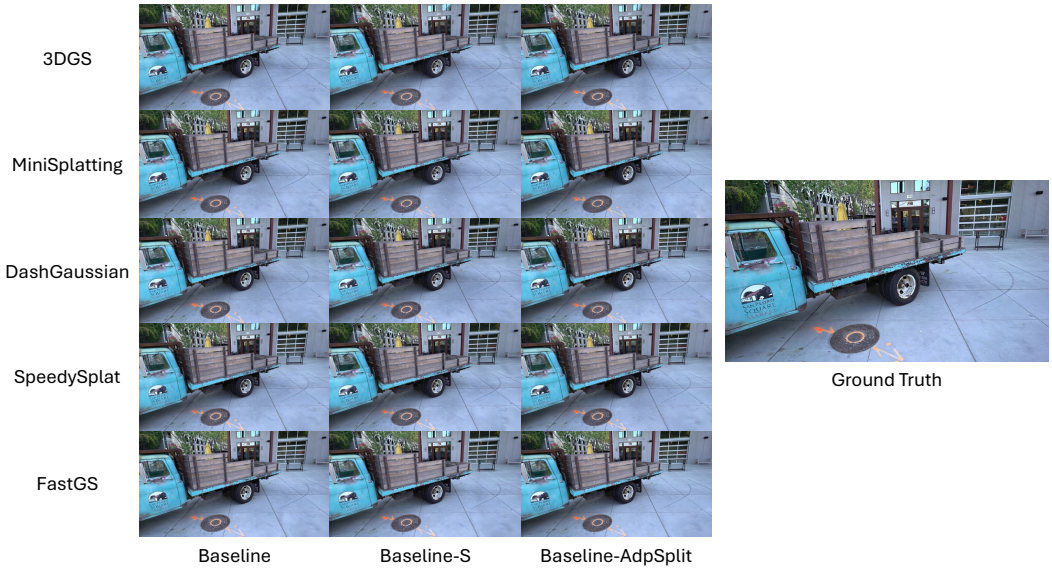

Figure 16: Qualitative results on `Truck` from Tanks&Temples

Table 6: Tuning results for L1-error threshold τ_{l1}, averaged over nine MipNeRF360 scenes

τ_{l1}	**PSNR**	**SSIM**	**LPIPS**	**Time**	**#G.**	**FPS**	**PM**
0.05	27.50	0.811	0.223	1111	2.60	122.70	10.65
0.10	27.54	0.810	0.225	1107	2.55	125.36	10.00
0.15	27.51	0.810	0.226	1109	2.51	124.44	9.94
0.20	27.48	0.809	0.227	1095	2.48	125.12	9.90

Table 7: Tuning results for erosion length r_{erode}, averaged over nine MipNeRF360 scenes

r_{erode}	**PSNR**	**SSIM**	**LPIPS**	**Time**	**#G.**	**FPS**	**PM**
0	27.48	0.811	0.221	1100	2.69	126.17	16.88
1	27.47	0.810	0.224	1109	2.63	125.06	11.58
2	27.54	0.810	0.225	1107	2.55	125.36	10.00
3	27.53	0.810	0.226	1110	2.49	125.89	9.92

B.2 Hyperparameter Sensitivity

Table 6–Table 14 summarize the average effect of AdpSplit hyperparameters. The first four tables average over all nine MipNeRF360 scenes. For $N_{\max}$ and V, we separately report outdoor and indoor averages. Time is reported in seconds, #G. in millions, and PM denotes peak GPU memory in GB. All experiments in this section were run on a single NVIDIA A100 80GB GPU.

Table 8: Tuning results for the number of error-level bands L, averaged over nine MipNeRF360 scenes.

L	PSNR	SSIM	LPIPS	Time	#G.	FPS	PM
1	27.46	0.810	0.225	1111	2.53	125.39	9.99
2	27.52	0.810	0.225	1107	2.55	125.49	10.00
3	27.54	0.810	0.225	1107	2.55	125.36	10.00
4	27.49	0.810	0.225	1107	2.56	125.17	10.02

Table 9: Tuning results for the child cap $N_{\max}$, averaged separately over outdoor and indoor MipNeRF360 scenes.

$N_{\max}$	Outdoor							Indoor						
	PSNR	SSIM	LPIPS	Time	#G.	FPS	PM	PSNR	SSIM	LPIPS	Time	#G.	FPS	PM
4	24.58	0.714	0.271	1139	3.52	110.77	9.95	31.17	0.931	0.168	1062	1.26	145.48	9.96
9	24.58	0.714	0.270	1143	3.57	109.72	10.02	31.23	0.931	0.168	1062	1.26	145.41	9.97
19	24.53	0.714	0.270	1148	3.62	110.20	10.09	31.23	0.931	0.168	1062	1.26	144.90	9.97
29	24.56	0.713	0.270	1152	3.63	109.28	10.11	31.19	0.930	0.168	1062	1.26	145.22	9.98
all	24.54	0.713	0.270	1159	3.70	108.28	10.21	31.22	0.930	0.168	1067	1.27	144.08	9.98

Table 10: Tuning results for minimum error-region area $m_{\min}$, averaged over nine MipNeRF360 scenes.

$m_{\min}$	PSNR	SSIM	LPIPS	Time	#G.	FPS	PM
0	27.52	0.810	0.224	1114	2.62	124.76	11.73
5	27.54	0.810	0.225	1107	2.55	125.36	10.00
10	27.50	0.810	0.225	1107	2.52	125.77	9.96

Table 11: Tuning results for the number of sampled views V, averaged separately over outdoor and indoor MipNeRF360 scenes.

V	Outdoor							Indoor						
	PSNR	SSIM	LPIPS	Time	#G.	FPS	PM	PSNR	SSIM	LPIPS	Time	#G.	FPS	PM
5	24.57	0.713	0.272	1129	3.46	112.09	10.78	31.16	0.931	0.169	1034	1.23	148.06	9.62
10	24.58	0.714	0.270	1143	3.57	109.72	10.02	31.19	0.931	0.168	1059	1.25	145.31	9.75
20	24.58	0.714	0.269	1157	3.64	108.96	10.99	31.23	0.931	0.168	1062	1.26	144.90	9.97
30	24.57	0.714	0.269	1177	3.70	108.22	22.30	31.27	0.931	0.167	1073	1.27	144.40	10.18
50	24.60	0.714	0.268	1196	3.80	106.55	29.35	31.23	0.931	0.167	1097	1.30	142.15	11.05

Table 12: Tuning results for Mahalanobis distance threshold γ_d, averaged over nine MipNeRF360 scenes.

γ_d	PSNR	SSIM	LPIPS	Time	#G.	FPS	PM
1.0	27.52	0.810	0.225	1104	2.55	123.58	10.01
1.5	27.54	0.810	0.225	1111	2.54	123.52	10.00
2.0	27.54	0.810	0.225	1107	2.55	125.36	10.00
2.5	27.52	0.810	0.225	1104	2.55	124.11	10.01
3.0	27.52	0.810	0.225	1109	2.54	123.56	10.00

Table 13: Tuning results for color difference threshold γ_c, averaged over nine MipNeRF360 scenes

γ_c	PSNR	SSIM	LPIPS	Time	#G.	FPS	PM
0.10	27.51	0.810	0.225	1109	2.54	123.25	10.00
0.15	27.54	0.810	0.225	1107	2.55	125.36	10.00
0.20	27.52	0.810	0.225	1106	2.53	123.49	9.98

Table 14: Sweep results of densification schedule, averaged over nine MipNeRF360 scenes

Total Iter.	**Densify Until**	**Method**	**PSNR**	**SSIM**	**LPIPS**	**Time**	**#G.**
18k	3k	3DGS-S	26.95	0.775	0.278	652	1.01
18k	3k	3DGS-AdpSplit	27.01	0.779	0.269	712	1.19
21k	6k	3DGS-S	27.22	0.799	0.242	834	1.97
21k	6k	3DGS-AdpSplit	27.29	0.802	0.234	874	2.19
24k	9k	3DGS-S	27.41	0.808	0.228	1051	2.39
24k	9k	3DGS-AdpSplit	27.54	0.810	0.225	1107	2.55
27k	12k	3DGS-S	27.47	0.812	0.223	1247	2.58
27k	12k	3DGS-AdpSplit	27.57	0.814	0.217	1299	2.80
30k	15k	3DGS	27.53	0.813	0.221	1426	2.65

Table 15: Training results of random point cloud initialization, averaged over nine MipNeRF360 scenes.

Method	PSNR	SSIM	LPIPS	Time	#G.
3DGS (random init.)	23.795	0.7320	0.3002	1323	1.73
3DGS-S (random init.)	23.557	0.7164	0.3202	973	1.36
3DGS-AdpSplit (random init.)	24.484	0.7470	0.2837	1111	1.73

B.3 Effect of Point Cloud Initialization

By default, 3D Gaussian Splatting [Kerbl et al., 2023] initializes the scene using a sparse point cloud obtained through structure from motion [Snavely et al., 2006]. This sparse point cloud already provides adequate information about the scene's 3D structure, leaving limited room for AdpSplit's error-aware child generation to improve reconstruction quality over the baseline, as shown in Table 1. To evaluate the ability of our method discovering missing geometry, we train 3DGS models from randomly initialized point clouds. All experiments in this section were run on a single NVIDIA A100 80GB GPU.

Table 15 compares three methods initialized from random point clouds: 3DGS with the full training schedule, 3DGS with a shortened training schedule (3DGS-S), and 3DGS with AdpSplit (3DGS-AdpSplit). Under random initialization, 3DGS-AdpSplit increases PSNR by 0.927 dB and SSIM by 0.0306 and reduces LPIPS by 0.0365 relative to 3DGS-S. Compared with 3DGS, it increases PSNR by 0.689 dB and SSIM by 0.0150 and reduces LPIPS by 0.0165. These results suggest that AdpSplit's error-driven placement yields larger improvements in rendering quality when the initial point cloud provides limited geometric information and standard splitting and cloning operations struggle to recover missing geometry.

B.4 Comparison with ConeGS

In this section, we compare AdpSplit with ConeGS [Baranowski et al., 2026]. We would like to note that a key advantage of AdpSplit is its broad applicability. Our adaptive splitting strategy complements existing 3DGS acceleration techniques and can be readily integrated into accelerated pipelines, such as SpeedySplat and FastGS (Table 1), to further reduce training time while maintaining reconstruction quality. In contrast, ConeGS relies on cone-shaped Gaussian initialization, which may complicate integration with other acceleration strategies, particularly those that assume standard 3DGS splitting. Furthermore, the two-minute iNGP pretraining stage in ConeGS introduces substantial overhead when state-of-the-art 3DGS accelerators often require only two to three minutes for training.

For a fair comparison, both methods are initialized with an iNGP-generated 1M point cloud and use matched Gaussian budgets and training schedules, with densification ending at 9k iterations and training ending at 24k iterations. We conduct the experiments on an NVIDIA RTX 5000 Ada GPU and report results averaged over the scenes in each dataset. Reported training times include only Gaussian optimization, excluding iNGP training and point cloud extraction.

As shown in Table 16, AdpSplit achieves higher PSNR on all three datasets and higher SSIM on MipNeRF360 and Deep-Blending. ConeGS-S achieves shorter training times on Deep-Blending and Tanks&Temples, primarily because ConeGS maintains most Gaussians as small, nearly spherical primitives, reducing the number of Gaussian–ray intersection tests.

B.5 Operation Time Profiling

We report a timing breakdown on MipNeRF360 using an NVIDIA RTX 5000 Ada GPU, averaged over the nine scenes. **Total** is the total training time. **Optim.** is the total optimizer step time. **Densify Total** includes gradient-stat collection, sampled-view rendering and error-map computation, split/clone/prune operations, and opacity reset (i.e., gradient-stat collection + Error Map + Densify/Prune + opacity reset). **Error Map** includes sampled-view rendering, dominant-map extraction, and L1 error-map computation. **Densify/Prune** includes error-region grouping (**Grouping**), child initialization (**Child Init.**), cross-view merging (**Merge**), and the actual split/clone/prune tensor updates.

Table 16: Comparison of AdpSplit and ConeGS-S on MipNeRF360, Deep-Blending, and Tanks&Temples.

Dataset	Method	PSNR	SSIM	LPIPS	Time	#G.	FPS	PM
MipNeRF360	3DGS-AdpSplit	27.846	0.8286	0.1873	1814	3.19	84.52	11.00
MipNeRF360	ConeGS-S	27.757	0.8216	0.1859	1871	3.19	95.33	8.81
Deep-Blending	3DGS-AdpSplit	30.288	0.9088	0.2248	1650	3.03	95.27	8.94
Deep-Blending	ConeGS-S	29.741	0.9015	0.2270	1446	3.03	124.79	7.45
Tanks&Temples	3DGS-AdpSplit	23.755	0.8543	0.1489	1012	1.94	122.55	5.46
Tanks&Temples	ConeGS-S	23.715	0.8568	0.1440	983	1.94	141.87	4.46

Table 17: Training time analysis

Method	#G.	Total	Optim.	Densify Total	Error Map	Densify/Prune	Grouping	Child Init.	Merge
3DGS-S	2.41	1345.86	370.06	3.97	0.00	2.37	0.00	0.00	0.00
3DGS-AdpSplit	2.55	1402.95	394.23	11.60	3.28	6.66	3.02	0.39	0.37

AdpSplit adds 7.63 seconds of densification overhead relative to 3DGS-S on average, which is less than 1% of the total training time. The main additional costs are sampled-view rendering/error-map computation (3.28s) and connected-component grouping (3.02s). The child-parameter estimation and cross-view merging costs are comparatively small (0.39s and 0.37s). The 57.09s gap in total time is mainly attributable to rendering and optimizing the increased number of Gaussians in -AdpSplit. Thus, the overhead of error analysis and child parameter estimation is minor, and the main speedup in the paper still comes from shortening the densification schedule, while AdpSplit recovers the quality that would otherwise be lost under that shortened schedule.

B.6 Impact of Input Image Resolution

We analyze how input image resolution affects AdpSplit's image-space error identification. We tested MipNeRF360 with 400px and 800px image widths using an NVIDIA A100 GPU (cf., the main-paper results use an image width of 1600px). At both lower resolutions, AdpSplit recovers most of the quality lost by the shortened schedule while preserving most of the training-time reduction. In the 400px setting, the metric gain from 3DGS-S to 3DGS-AdpSplit is +0.091 PSNR and -0.0007 LPIPS. In the 800px setting, the gain is +0.128 PSNR and -0.002 LPIPS. The time reduction from 3DGS to 3DGS-AdpSplit is 19% and 21% in the 400px and 800px settings, respectively. Because lower image resolutions require fewer propagated Gaussians, peak memory is substantially reduced compared with the full-resolution setting (see Table 5), from 10GB to 4.84GB at 800px and 2.62GB at 400px. Compared with 3DGS-S, 3DGS-AdpSplit slightly increases peak memory in both the 400px and 800px settings; this is mainly due to the slightly larger number of Gaussians and the error-driven child initialization logic. In conclusion, AdpSplit reduces training time while recovering the quality metrics with only a minor increase in peak memory at lower image resolutions, similar to the trend observed in Table 1 of the main paper.

B.7 Results Across Multiple Random Seeds

To verify the statistical significance of the values reported in Table 1 in the main paper, we ran each shortened -S baseline and its -AdpSplit variant with five random seeds (0–4) on MipNeRF360, Deep-Blending, and Tanks&Temples using an NVIDIA RTX 5000 Ada GPU and report the per-scene mean metrics in Table 19–Table 21. Here, Δ denotes -AdpSplit $-$ -S for each metric. Across the 39 scene-baseline pairs, AdpSplit improves mean PSNR in 36 cases, SSIM in 37 cases, and LPIPS in 37 cases, with one LPIPS tie.

Table 18: Training results on images downsampled by factors of 2 (800px) and 4 (400px)

Method	2× downsampling (800px)							4× downsampling (400px)						
	PSNR	SSIM	LPIPS	Time	#G.	FPS	PM	PSNR	SSIM	LPIPS	Time	#G.	FPS	PM
3DGS	28.638	0.8612	0.1382	696.88	2.25	187.78	4.87	29.887	0.9064	0.0853	458.06	1.50	224.88	2.63
3DGS-S	28.468	0.8586	0.1421	525.95	2.15	196.62	4.69	29.787	0.9056	0.0870	355.73	1.48	223.85	2.58
3DGS-AdpSplit	28.596	0.8593	0.1401	548.73	2.20	193.09	4.84	29.878	0.9058	0.0863	370.86	1.49	226.01	2.62

Table 19: Per-scene results for the 3DGS baseline, averaged over five random seeds. Δ denotes the difference between the -AdpSplit and -S variants.

Scene	Model	PSNR	SSIM	LPIPS
MipNeRF360 `Bicycle`	3DGS-S	24.965	0.7332	0.2632
	3DGS-AdpSplit	25.040	0.7363	0.2584
	Δ	+0.075	+0.0031	-0.0048
MipNeRF360 `Flowers`	3DGS-S	21.284	0.5776	0.3717
	3DGS-AdpSplit	21.315	0.5818	0.3657
	Δ	+0.031	+0.0042	-0.0060
MipNeRF360 `Garden`	3DGS-S	27.239	0.8546	0.1258
	3DGS-AdpSplit	27.326	0.8559	0.1236
	Δ	+0.087	+0.0013	-0.0022
MipNeRF360 `Stump`	3DGS-S	26.498	0.7617	0.2546
	3DGS-AdpSplit	26.583	0.7644	0.2477
	Δ	+0.085	+0.0027	-0.0069
MipNeRF360 `Treehill`	3DGS-S	22.570	0.6299	0.3598
	3DGS-AdpSplit	22.692	0.6314	0.3549
	Δ	+0.122	+0.0015	-0.0049
MipNeRF360 `Room`	3DGS-S	31.606	0.9266	0.1968
	3DGS-AdpSplit	31.807	0.9275	0.1948
	Δ	+0.201	+0.0009	-0.0020
MipNeRF360 `Counter`	3DGS-S	29.022	0.9141	0.1846
	3DGS-AdpSplit	29.108	0.9149	0.1827
	Δ	+0.086	+0.0008	-0.0019
MipNeRF360 `Kitchen`	3DGS-S	31.168	0.9320	0.1170
	3DGS-AdpSplit	31.606	0.9329	0.1156
	Δ	+0.438	+0.0009	-0.0014
MipNeRF360 `Bonsai`	3DGS-S	32.295	0.9463	0.1805
	3DGS-AdpSplit	32.354	0.9473	0.1786
	Δ	+0.059	+0.0010	-0.0019
Deep-Blending `Drjohnson`	3DGS-S	29.389	0.9038	0.2413
	3DGS-AdpSplit	29.399	0.9035	0.2406
	Δ	+0.010	-0.0003	-0.0007
Deep-Blending `Playroom`	3DGS-S	30.257	0.9091	0.2430
	3DGS-AdpSplit	30.261	0.9097	0.2414
	Δ	+0.004	+0.0006	-0.0016
Tanks&Temples `Train`	3DGS-S	22.130	0.8157	0.2037
	3DGS-AdpSplit	22.315	0.8176	0.2008
	Δ	+0.185	+0.0019	-0.0029
Tanks&Temples `Truck`	3DGS-S	25.273	0.8832	0.1442
	3DGS-AdpSplit	25.385	0.8837	0.1430
	Δ	+0.112	+0.0005	-0.0012

Table 20: Per-scene results for the SpeedySplat baseline, averaged over five random seeds. Δ denotes the difference between the -AdpSplit and -S variants.

Scene	Model	PSNR	SSIM	LPIPS
MipNeRF360 `Bicycle`	SpeedySplat-S	24.726	0.7023	0.3329
	SpeedySplat-AdpSplit	24.788	0.7073	0.3267
	Δ	+0.062	+0.0050	-0.0062
MipNeRF360 `Flowers`	SpeedySplat-S	21.205	0.5621	0.4120
	SpeedySplat-AdpSplit	21.269	0.5672	0.4067
	Δ	+0.064	+0.0051	-0.0053
MipNeRF360 `Garden`	SpeedySplat-S	26.786	0.8245	0.1943
	SpeedySplat-AdpSplit	26.884	0.8265	0.1921
	Δ	+0.098	+0.0020	-0.0022
MipNeRF360 `Stump`	SpeedySplat-S	26.499	0.7611	0.2885
	SpeedySplat-AdpSplit	26.619	0.7653	0.2828
	Δ	+0.120	+0.0042	-0.0057
MipNeRF360 `Treehill`	SpeedySplat-S	22.575	0.5981	0.4527
	SpeedySplat-AdpSplit	22.600	0.6043	0.4433
	Δ	+0.025	+0.0062	-0.0094
MipNeRF360 `Room`	SpeedySplat-S	31.090	0.9108	0.2409
	SpeedySplat-AdpSplit	31.220	0.9118	0.2383
	Δ	+0.130	+0.0010	-0.0026
MipNeRF360 `Counter`	SpeedySplat-S	28.418	0.8869	0.2396
	SpeedySplat-AdpSplit	28.481	0.8880	0.2381
	Δ	+0.063	+0.0011	-0.0015
MipNeRF360 `Kitchen`	SpeedySplat-S	30.508	0.9072	0.1728
	SpeedySplat-AdpSplit	30.635	0.9080	0.1711
	Δ	+0.127	+0.0008	-0.0017
MipNeRF360 `Bonsai`	SpeedySplat-S	31.313	0.9302	0.2171
	SpeedySplat-AdpSplit	31.399	0.9312	0.2161
	Δ	+0.086	+0.0010	-0.0010
Deep-Blending `Drjohnson`	SpeedySplat-S	29.142	0.9005	0.2655
	SpeedySplat-AdpSplit	29.132	0.9000	0.2655
	Δ	-0.010	-0.0005	+0.0000
Deep-Blending `Playroom`	SpeedySplat-S	30.174	0.9072	0.2661
	SpeedySplat-AdpSplit	30.110	0.9073	0.2655
	Δ	-0.064	+0.0001	-0.0006
Tanks&Temples `Train`	SpeedySplat-S	21.694	0.7760	0.2849
	SpeedySplat-AdpSplit	21.822	0.7789	0.2817
	Δ	+0.128	+0.0029	-0.0032
Tanks&Temples `Truck`	SpeedySplat-S	25.129	0.8698	0.1823
	SpeedySplat-AdpSplit	25.218	0.8699	0.1824
	Δ	+0.089	+0.0001	+0.0001

Table 21: Per-scene results for the FastGS baseline, averaged over five random seeds. Δ denotes the difference between the -AdpSplit and -S variants.

Scene	Model	PSNR	SSIM	LPIPS
MipNeRF360 `Bicycle`	FastGS-S	24.489	0.6922	0.3343
	FastGS-AdpSplit	24.801	0.7151	0.3081
	Δ	+0.312	+0.0229	-0.0262
MipNeRF360 `Flowers`	FastGS-S	20.986	0.5426	0.4258
	FastGS-AdpSplit	21.114	0.5530	0.4112
	Δ	+0.128	+0.0104	-0.0146
MipNeRF360 `Garden`	FastGS-S	27.118	0.8327	0.1800
	FastGS-AdpSplit	27.290	0.8424	0.1608
	Δ	+0.172	+0.0097	-0.0192
MipNeRF360 `Stump`	FastGS-S	26.340	0.7421	0.3159
	FastGS-AdpSplit	26.502	0.7511	0.3006
	Δ	+0.162	+0.0090	-0.0153
MipNeRF360 `Treehill`	FastGS-S	22.758	0.5981	0.4473
	FastGS-AdpSplit	22.763	0.6010	0.4374
	Δ	+0.005	+0.0029	-0.0099
MipNeRF360 `Room`	FastGS-S	31.758	0.9197	0.2188
	FastGS-AdpSplit	31.925	0.9226	0.2129
	Δ	+0.167	+0.0029	-0.0059
MipNeRF360 `Counter`	FastGS-S	29.059	0.9070	0.2045
	FastGS-AdpSplit	29.173	0.9088	0.2005
	Δ	+0.114	+0.0018	-0.0040
MipNeRF360 `Kitchen`	FastGS-S	31.678	0.9280	0.1294
	FastGS-AdpSplit	31.944	0.9305	0.1240
	Δ	+0.266	+0.0025	-0.0054
MipNeRF360 `Bonsai`	FastGS-S	32.168	0.9426	0.1921
	FastGS-AdpSplit	32.375	0.9444	0.1876
	Δ	+0.207	+0.0018	-0.0045
Deep-Blending `Drjohnson`	FastGS-S	29.305	0.8979	0.2793
	FastGS-AdpSplit	29.335	0.8981	0.2763
	Δ	+0.030	+0.0002	-0.0030
Deep-Blending `Playroom`	FastGS-S	30.567	0.9086	0.2681
	FastGS-AdpSplit	30.450	0.9093	0.2624
	Δ	-0.117	+0.0007	-0.0057
Tanks&Temples `Train`	FastGS-S	22.372	0.8011	0.2481
	FastGS-AdpSplit	22.396	0.8034	0.2439
	Δ	+0.024	+0.0023	-0.0042
Tanks&Temples `Truck`	FastGS-S	25.696	0.8750	0.1786
	FastGS-AdpSplit	25.882	0.8781	0.1726
	Δ	+0.186	+0.0031	-0.0060

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: AdpSplit is able to recover scene reconstruction quality lost due to a shortened densification schedule by adaptively adding new Gaussians in high-error regions that indicate under-reconstruction, as demonstrated in Table 1.

   Guidelines:

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

2. **Limitations**

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

   Answer: [Yes]

   Justification: We discuss the limitations in Conclusion section.

   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 doesn't include theoretical results.

Guidelines:

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

4. **Experimental result reproducibility**

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

   Answer: [Yes]

   Justification: The paper provides detailed information in section 5.1 to reproduce results reported in section 5.2 and Appendix B.

   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: [Yes]

Justification: The datasets are all publicly accessible in the internet. We don't provide our code at the time of submission. At the point of camera-ready submission, we open the source code to the public, thus we change the answer from No to Yes.

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 detailed experiment settings are listed in Section 5.1.

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: [N/A]

Justification: This paper follows the existing work about 3DGS and lists the essential detailed quantitative results in Section 5.2 and Appendix B. But it does not list error bars.

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: This paper lists the hardware used in Section 5.2 and Appendix B. This paper shows the required graphic memory in Appendix B.

   Guidelines:

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

9. **Code of ethics**

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

   Answer: [Yes]

   Justification: The research conducted in this paper confirm the NeurIPS Code of Conducts.

   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: [N/A]

    Justification: This paper primarily focuses on improving model training efficiency. It is not expected to have any societal impact.

    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: This paper does not pose such risk.

    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: This paper follows the applicable licenses and terms of usage.

    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: [No]

    Justification: This paper provides implementation codes as new assets. The code and documentation will be open after paper is accepted.

    Guidelines:

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

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

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

    Answer: [N/A]

    Justification: This paper does not involve crowdsourcing nor 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: This paper does not involve crowdsourcing nor research with human subjects.

    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: LLMs are used solely for grammar improvement.

    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.