

# Mini-Splatting2: Building 360 Scenes within Minutes via Aggressive Gaussian Densification

Guangchi Fang, Bing Wang

The Hong Kong Polytechnic University

guangchi.fang@gmail.com, bingwang@polyu.edu.hk



Figure 1. Through aggressive Gaussian densification, our proposed Mini-Splatting2 efficiently reconstructs dense 3D Gaussian representations in the early stages of training. With subsequent simplification and optimization, high-quality Gaussian models are produced in just a few minutes. The Gaussian centers of the *bicycle* model are visualized as dense point clouds (3K iterations, 5.1 million Gaussians) and high-quality point clouds (8K iterations, 0.7 million Gaussians). Furthermore, we present rendering results for both Mini-Splatting2 (3 minutes) and the latest official 3DGS implementation, 3DGS-accel [18, 31] (12 minutes), for comparison.

## Abstract

*In this study, we explore the essential challenge of fast scene optimization for Gaussian Splatting. Through a thorough analysis of the geometry modeling process, we reveal that dense point clouds can be effectively reconstructed early in optimization through Gaussian representations. This insight leads to our approach of aggressive Gaussian densification, which provides a more efficient alternative to conventional progressive densification methods. By significantly increasing the number of critical Gaussians, we enhance the model capacity to capture dense scene geometry at the early stage of optimization. This strategy is seamlessly integrated into the Mini-Splatting densification and simplification framework, enabling rapid convergence without compromising quality. Additionally, we introduce visibility culling within Gaussian Splatting, leveraging per-view Gaussian importance as precomputed visibility to accelerate the optimization process. Our Mini-Splatting2 achieves a balanced trade-off among optimization time, the number of Gaussians, and rendering quality, establishing a strong baseline for future Gaussian-Splatting-based works. Our work sets the stage for more efficient, high-quality 3D scene modeling in real-world applications, and the code will be made available no matter acceptance.*

## 1. Introduction

As a recent breakthrough in novel view synthesis, 3D Gaussian Splatting (3DGS) [18] has drawn significant attention for its fast optimization and real-time rendering capabilities, both essential for real-world applications such as object modeling and scene reconstruction. Compared to prior radiance field-based methods [2, 28, 29], 3DGS offers substantial improvements through its Gaussian representation [44] and a carefully designed rasterization pipeline. Despite the algorithmic and implementation advancements in 3DGS, most approaches still require approximately 10 minutes or more to recover a typical 360 scene (*e.g.*, the MipNeRF360 dataset [1]). In this regard, we recognize this ongoing need for further acceleration, and our study aims to advance Gaussian Splatting optimization while preserving high rendering quality.

Efficient scene geometry modeling is a critical step for accelerated Gaussian Splatting optimization. Using sparse points from Structure-from-Motion (SfM) [32] as an initialization, 3DGS [18] relies heavily on Gaussian densification to capture scene geometry. Many follow-up studies [3, 7, 9, 19, 24, 40, 43] have introduced specialized Gaussian densification techniques to aim at improving geometry and appearance modeling. However, these stud-

ies primarily focus on enhancing final rendering quality or reconstruction fidelity, often neglecting the efficiency needs for fast scene optimization. Some recent approaches employ learned data priors [4, 6, 11, 26, 35] or incorporate depth sensors [8, 17] to tackle the geometry modeling challenge, achieving promising results in specific use cases. In contrast, this work targets a more generalized setting, multi-view-based per-scene optimization. In this broader context, efficient geometry modeling for Gaussian representations remains a challenging yet crucial task.

In this paper, we first investigate the geometry modeling process of 3DGS by visualizing Gaussian centers throughout the densification process. While 3D Gaussians with limited optimization iterations show notable artifacts, we observe that it remains feasible to extract a dense point cloud from the corresponding scene representation. Building on this observation, we propose the concept of aggressive Gaussian densification, which dramatically increase the number of critical Gaussians, enabling dense Gaussian reconstruction within a significantly shortened period. These dense Gaussians can be further refined into a high-quality Gaussian model through subsequent optimization and simplification, as shown in Fig. 1. To further accelerate the optimization progress, we introduce visibility culling into Gaussian Splatting by leveraging Gaussian importance from each training view as precomputed visibility data. Combined within the Mini-Splatting framework, these techniques substantially reduce optimization time while maintaining a constrained number of Gaussians and high-quality rendering. Figure 1 also indicates the training progress of *bicycle* and the corresponding rendering performance of our Mini-Splatting2. Our main contributions are as follows:

- Through a comprehensive analysis of the 3DGS geometry modeling process, we identify the feasibility of extracting dense points from Gaussian representations and their potential for refinement into high-quality Gaussian models.
- We introduce aggressive Gaussian densification, which increases the number of critical Gaussians to speed up optimization convergence. To further accelerate the process, we propose visibility Gaussian culling, using per-view Gaussian importance as precomputed visibility.
- Extensive experimental results on multiple benchmarks and datasets demonstrate that our Mini-Splatting2 achieves a balanced trade-off among optimization time, number of Gaussians and rendering quality.

## 2. Related Work

**Gaussian Splatting.** 3D Gaussian Splatting (3DGS) [18] was initially developed for novel view synthesis, emphasizing real-world immersive rendering. Early work in this domain [28] employed multi-layer perceptrons to implicitly model scenes as neural radiance fields. More recent approaches have moved towards explicit representa-

tions, including voxel grids [25, 34], hash grids [29], and point clouds [22, 37], enhancing representation capacity and thereby improving training speed and rendering quality. Expanding on classical 3D representations, elliptical Gaussians [44] offer robust capabilities for modeling scene geometry and appearance while inherently supporting parallel rasterization [18]. This approach eliminates the need for ray marching required in previous volume rendering techniques [28], thereby accelerating both optimization and rendering.

With the strong scalability of Gaussian representations, 3DGS has demonstrated promising potential across various applications. Studies from diverse fields, including 3D reconstruction, computer graphics, and robotics, have applied this technique to downstream tasks such as surface reconstruction [16, 41], content generation [5, 36] and localization and mapping [27, 38]. In addition to the substantial academic literature, the broader community of startups and engineers is also actively exploring the commercial potential of 3DGS for real-world platforms, such as social media, online display, and computer-aided design. Most academic and industrial applications are increasingly demanding faster Gaussian Splatting optimization.

**Gaussian Densification.** Gaussian densification is a crucial step in recovering scene geometry. The vanilla 3DGS approach [18] initializes with sparse points from SfM [32] and employs an adaptive density control strategy that first selects Gaussians based on image-space gradients and then clones or splits them according to their scale. Building on this approach, recent studies [3, 20, 31, 40, 43] have focused on Gaussian selection strategies informed by image-space information and intrinsic Gaussian attributes. Other densification methods integrate multi-view constraints [7, 9, 24], leverage advanced optimization techniques [19], or perform point cloud analysis [12]. Intuitively, a well-designed densification strategy can improve optimization convergence, thereby accelerating the overall optimization process. However, most existing algorithms, with few exceptions such as [12, 20, 31], primarily target improved rendering quality, often increasing the optimization budget.

**Accelerated Gaussian Optimization.** Recent advancements have introduced various methods to accelerate Gaussian Splatting optimization. A notable trend [4, 6, 11, 26, 35] leverages learning-based techniques to directly predict scene geometry in the form of depth maps, dense point clouds, or Gaussian attributes. These algorithms have shown significant potential for 3D reconstruction, particularly in sparse-view scenarios and man-made environments.

In this work, we focus specifically on the general and classical multi-view settings. A practical approach in this context is to enhance the forward and backward rasterization pipeline [10, 31, 39]. For example, the codebase of the Taming-3DGS [31], referred to as 3DGS-accel, contains enhanced implementations including per-splat backward ren-

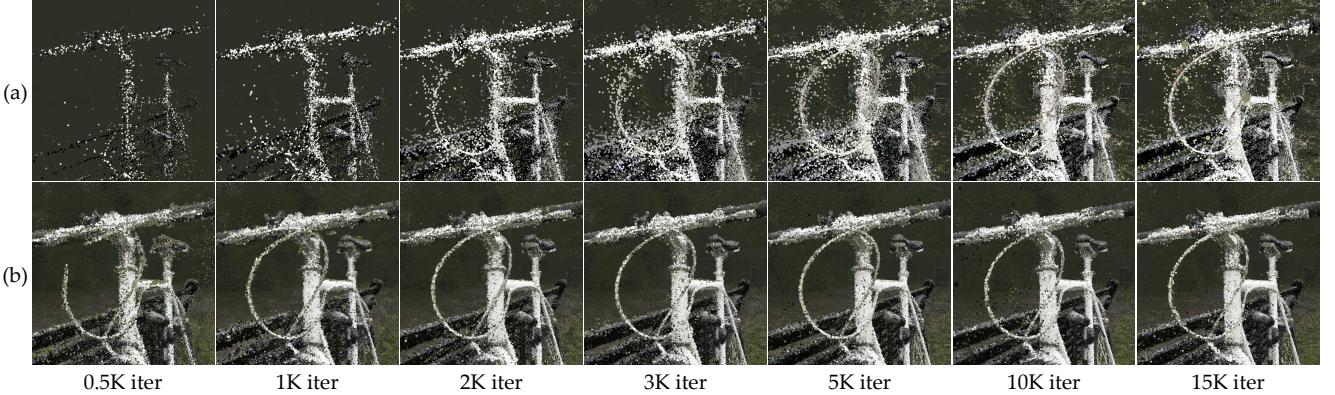


Figure 2. Visual analysis of (a) Gaussian centers and (b) corresponding depth points across 0.5K to 15K iterations.

dering, fused-SSIM, separation of spherical harmonics, and tile-based culling [30], achieving a  $2.7\times$  speedup in training time. We validated the effectiveness of each modification and recommend it as a new baseline implementation for 3DGS. Another recent work, 3DGS-LM [15], focuses on accelerating Gaussian Splatting optimization by developing a specialized 3DGS-oriented optimizer. In contrast, our study addresses the challenge from a point cloud reconstruction perspective.

### 3. Bridging 3DGS and Point Reconstruction

**Gaussian Representation.** 3D Gaussian Splatting (3DGS) [18] utilizes an extensive number of elliptical Gaussians  $\{\mathcal{G}_i|i = 1, \dots, N\}$  [44] to represent scenes. Each Gaussian  $\mathcal{G}_i$  includes a set of attributes: center  $\mathbf{p}_i \in \mathbb{R}^{3 \times 1}$ , opacity  $\alpha_i \in [0, 1]$ , and a covariance matrix  $\Sigma_i \in \mathbb{R}^{3 \times 3}$  which encodes the 3D scale and rotation, to collectively model the scene geometry. Additionally, the view-dependent color  $\mathbf{c}_i$ , represented by spherical harmonic (SH) coefficients, is associated with each 3D Gaussian  $\mathcal{G}_i$  to represent scene appearance. Through alpha-blending [28], 3DGS employs a well-designed differentiable Gaussian rasterizer to render images from this Gaussian representation as follows:

$$\mathbf{c}(\mathbf{x}) = \sum_{i=1}^N w_i \mathbf{c}_i, \quad \text{where } w_i = T_i \alpha_i \mathcal{G}_i^{2D}(\mathbf{x}). \quad (1)$$

Here, the accumulated transmittance  $T_i$  is defined as  $T_i = \prod_{j=1}^{i-1} (1 - \alpha_j \mathcal{G}_j^{2D}(\mathbf{x}))$ , and  $\mathcal{G}_i^{2D}$  denotes the 2D projection of 3D Gaussian  $\mathcal{G}_i$  onto the image plane at location  $\mathbf{x}$ .

**Geometry Modelling.** Starting from sparse SfM points as initial Gaussians, 3DGS [18] relies on Gaussian densification to capture scene geometry. Specifically, 3DGS identifies blurry regions by accumulating image-space gradients for each Gaussian, then iteratively selects Gaussians with high gradients, cloning or splitting them according to their 3D scale. While this approach enables adaptive density con-

trol, it often results in suboptimal geometry modeling, evident in the irregular spatial distribution of Gaussians.

Mini-Splatting [12] addresses these geometry modeling limitations, such as overlapping and under-reconstruction, by reorganizing the spatial distribution of Gaussians. Specifically, this approach enhances the 3DGS densification process and introduces an aggressive Gaussian simplification strategy, significantly reducing the number of Gaussians while simultaneously accelerating optimization. Despite these advances, the main geometry modeling phase (*i.e.*, densification before 15K iterations) remains a substantial portion of the optimization time. Thus, within the Mini-Splatting framework, the primary challenge in optimizing Gaussian Splatting lies in achieving efficient Gaussian densification. In other words, this study centers on whether dense Gaussian reconstruction can be accomplished within a reduced period or in the early optimization stages.

**Visual Analysis of Gaussians.** Given that Gaussian centers, similar to point clouds, exhibit a strong correlation with scene geometry [12], we analyze the geometry modeling progress of Gaussian Splatting by visualizing Gaussian centers as point clouds throughout densification. As illustrated in Fig. 2 (a), Gaussians are iteratively optimized and densified, beginning from an initial set of sparse points (0.5K iterations). As optimization and densification proceed, the number of Gaussians increases progressively, forming an approximate scene geometry with notable floaters from 0.5K to 5K iterations. Subsequently, Gaussians tend to converge toward the object surface until 15K iterations, with some Gaussians remaining as floaters to capture camera or view-dependent effects.

In general, the presence of incomplete structures with significant floaters, especially in reconstructed models limited to a small number of iterations (*i.e.*, 1K to 3K iterations), indicates suboptimal reconstruction quality. Consequently, directly leveraging Gaussian centers for point cloud reconstruction and further geometry modeling is impractical.

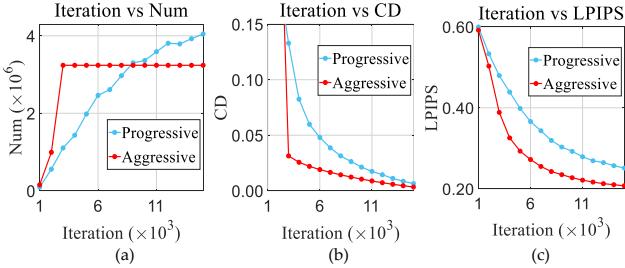


Figure 3. The relationship between the optimization progress of *bicycle* and (a) the number of Gaussians, (b) geometric similarity with the resulting Gaussians, and (c) rendering quality, as observed for both the progressive and our aggressive densification strategies.

cal at the early stages of optimization.

**Point Cloud Reconstruction via Gaussians.** The aforementioned challenge can also be framed as point cloud reconstruction through Gaussians, which can be further addressed by incorporating additional geometry-related Gaussian attributes within a depth estimation framework. Following the implementation of Mini-Splatting [12], we define the reconstructed depth map  $d$  as:

$$d(\mathbf{x}) = d_{i_{max}}^{mid}(\mathbf{x}), \quad \text{where } i_{max} = \arg \max_i w_i. \quad (2)$$

Here, Gaussian depth  $d^{mid}$  represents the depth calculated from the midpoint of ray intersections with Gaussian ellipsoids, and  $w_i$  denotes the blending weight in Eq. (1). Depth points can then be merged and reconstructed by directly re-projecting the depth map into world space.

A key observation is that dense point clouds can be efficiently reconstructed in the early stages of optimization with this approach. As illustrated in Fig. 2 (b), even Gaussian representations with imperfect spatial positions (*e.g.*, 1K to 3K iterations) are still capable of producing reasonable point reconstruction results, particularly for foreground objects. This finding underscores the potential for efficient dense Gaussian reconstruction by simply initializing depth points as Gaussians, providing empirical support for our aggressive Gaussian densification approach in Sec. 4.

## 4. Aggressive Gaussian Densification

In this section, we introduce aggressive Gaussian densification as an alternative to conventional progressive densification. We then propose critical Gaussian identification and aggressive Gaussian clone, which are integrated with depth reinitialization, to form the overall densification pipeline.

**Gaussian Densification: from Progressive to Aggressive.** A typical Gaussian Splatting optimization process can be roughly divided into two main phases: the densification phase (*i.e.*, optimization along with Gaussian densification before 15K iterations) and the refinement phase

(*i.e.*, pure optimization after 15K iterations). Most existing works design densification algorithms following the guidance of **Progressive Gaussian Densification** introduced in 3DGS [18], which incrementally increases the number of Gaussians in a restrained and iterative manner over an extended densification period, as shown in Fig. 3.

In contrast, we introduce the concept of aggressive approach induced from the analysis in Sec. 3. **Aggressive Gaussian Densification** aims to dramatically densify 3D Gaussians within a constrained number of optimization iterations, thereby effectively shortening the densification period and accelerating the overall optimization progress.

**Design Choice.** One straightforward implementation of aggressive densification strategy is to reconstruct dense Gaussians with a predefined number of sampled depth points according to Eq. (2), similar to the depth reinitialization approach [12] but applied at a limited optimization iteration (*e.g.*, 2K iterations). In practice, this naive approach can significantly accelerate the optimization process. However, simply setting a hyperparameter for the number of sampled points limits its scalability to more general scenes. Furthermore, this operation often results in a sudden increase in the number of Gaussians (*e.g.*, from 0.5 to 4 million for *bicycle*), which can disrupt the optimization process and lead to a decline in rendering quality. Retaining depth reinitialization as the key component of aggressive densification, we address these problems by incorporating critical Gaussian identification and aggressive Gaussian clone, resulting in our aggressive Gaussian densification method.

**Critical Gaussian Identification.** Assuming that the Gaussian representation with sparse initialization is in the early stages of optimization, all 3D Gaussians can be considered as being in an under-reconstruction state. In this phase, our goal is to identify as many critical Gaussians as possible, which are likely to represent the object surface, rather than directly cloning and splitting all Gaussians, which is typically a degraded case by reducing densification threshold. To achieve this, we derive critical Gaussians through an approximation of alpha blending as follows:

$$\mathbf{c}(\mathbf{x}) = \sum_{i=1}^N w_i \mathbf{c}_i \approx w_{i_{max}} \mathbf{c}_{i_{max}}, \quad (3)$$

where  $i_{max} = \arg \max_i w_i$ , and the critical Gaussians  $\mathcal{G}_{i_{max}}$  are those providing the maximum blending weight  $w_{i_{max}}$ . To ensure a minimum contribution of each selected Gaussian, we apply a predefined threshold via the quantile function [23] using the 0.99 quantile to filter out a few less significant Gaussians from  $\mathcal{G}_{i_{max}}$ , resulting in the final set  $\mathcal{G}^{crit}$ . This formulation is similar to intersection preserving in Mini-Splatting [12], yet follows a distinct underlying principle tailored to densification.

**Aggressive Gaussian Clone.** According to [3, 19], cloning or splitting Gaussians while preserving opacity introduces

a bias that implicitly amplifies the influence of densified Gaussians, thereby disrupting the optimization process. This issue becomes more significant in our aggressive densification approach (*e.g.*, approximately 5% or fewer Gaussians are selected at each densification iteration in the progressive strategy, compared to 30% in our case). To mitigate this problem, we smooth the densification operation in accordance with the state transition method in [19]. Specifically, we adopt Gaussian clone to densify all critical Gaussians, and set the number of cloned Gaussians to 2, which simplifies the computation of the Gaussian center  $\mathbf{p}^{\text{new}} = \mathbf{p}^{\text{old}}$ , opacity  $\alpha^{\text{new}} = 1 - \sqrt{1 - \alpha^{\text{old}}}$  and covariance matrix  $\Sigma^{\text{new}} = (\alpha^{\text{old}})^2 \left( 2\alpha^{\text{new}} - \frac{(\alpha^{\text{new}})^2}{\sqrt{2}} \right)^{-2} \Sigma^{\text{old}}$ .

**Overall Aggressive Densification Pipeline.** We further integrate the aforementioned modules into the densification stage and ensure compatibility with depth reinitialization [12]. Specifically, the original progressive densification is retained, with critical Gaussian identification and aggressive Gaussian clone applied every 250 iterations starting from 500 iterations. Depth reinitialization is conducted at 2K iterations, with the number of sampled depth points set to match the current number of Gaussians. We shorten the total densification period to 3K iterations, with the overall optimization limited to 18K steps. Figure 3 shows the optimization progress of *bicycle* for both progressive and aggressive densification strategies. The proposed aggressive densification significantly increases the number of Gaussians, thereby accelerating convergence, as reflected in both the geometric similarity of resulting Gaussians and rendering quality. Our aggressive densification pipeline is fully compatible with the Mini-Splatting framework, allowing Mini-Splatting simplification to be applied at 3K and 8K iterations, consistent with the original setup.

## 5. Visibility Gaussian Culling

To further accelerate the optimization process, we incorporate visibility culling into the Gaussian Splatting framework. The vanilla 3DGS [18] already applies a modified frustum culling technique, which preserves Gaussians with touched tiles during preprocessing. The effects of this culling on typical 3DGS and Mini-Splatting models for a specific view are shown in Fig. 4 (a). In this section, we first define Gaussian visibility and subsequently integrate visibility Gaussian culling into the optimization.

**Gaussian Visibility.** For the  $k$ th training view, the visibility  $V_i^k$  of Gaussian  $\mathcal{G}_i$  can be determined by its importance, derived from the blending weights in Eq. (1). Specifically, we first compute Gaussian importance  $I_i^k$  for the  $k$ th view by accumulating blending weights  $w_{ij}^k$  of Gaussian  $\mathcal{G}_i$  as  $I_i^k = \sum_{j=1}^J w_{ij}^k$ , where  $J$  represents the total number of rays intersecting  $\mathcal{G}_i$ . The visibility mask  $V_i^k$  is calculated:

$$V_i^k = \mathbb{I}(I_i^k > \tau), \quad (4)$$

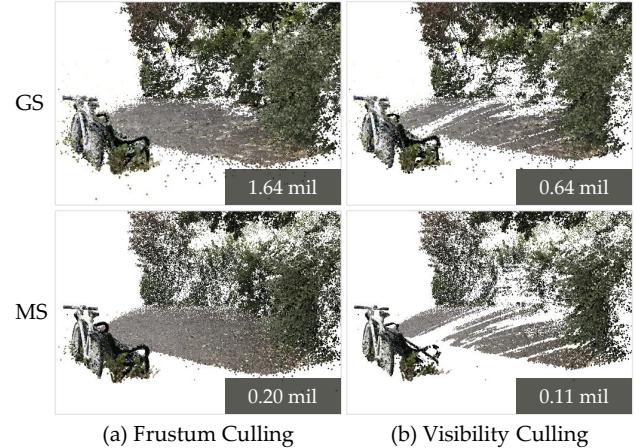


Figure 4. The visualization shows the distribution of Gaussians after applying two distinct culling techniques: (a) frustum culling and (b) visibility culling, for both Gaussian Splatting (GS) and Mini-Splatting (MS). Each image highlights the effect of these techniques in reducing unnecessary Gaussians, with the respective number of retained Gaussians clearly indicated.

where  $\mathbb{I}$  represents the indicator function, and  $\tau$  is a predefined threshold, obtained through the quantile function [23], with the keep quantile set at 0.99. As shown in Fig. 4 (b), visibility culling effectively masks more Gaussians than frustum culling, reducing computational overhead while preserving critical visual information.

**Precomputed Visibility for Gaussian Optimization.** The calculation of Gaussian visibility is seamlessly integrated with our densification and simplification processes, all anchored by Gaussian importance as a core metric. In practice, we compute and update visibility masks for each training view alongside the critical Gaussian identification and the simplification steps in Mini-Splatting. Visibility Gaussian culling is activated between iterations 500 and 13K, during which all non-visible Gaussians are excluded in the preprocessing phase of rasterization. This approach effectively reduces computational load by focusing only on Gaussians that contribute to the visible scene.

## 6. Experiments

We first present experimental setup, and then assess Mini-Splatting2 in terms of rendering quality and resource consumption. We further show the potential our algorithm for point cloud reconstruction, and conduct ablation on the proposed algorithm. Finally, we discuss the limitation and the corresponding improvement in future work.

### 6.1. Experimental Setup

**Datasets.** We conduct experiments on three real-world datasets: Mip-NeRF360 [1], Tanks&Temples [21], and

Table 1. Quantitative evaluation of our Mini-Splatting2 and other methods, categorizing algorithms into sparse and dense Gaussian models. Representative NeRF-based methods are also included as performance references.

Dataset Method   Metric	Mip-NeRF 360					Tanks&Temples					Deep Blending				
	SSIM ↑	PSNR ↑	LPIPS ↓	Train ↓	Num	SSIM ↑	PSNR ↑	LPIPS ↓	Train ↓	Num	SSIM ↑	PSNR ↑	LPIPS ↓	Train ↓	Num
mip-NeRF 360 [1]	0.792	27.69	0.237	-	-	0.759	22.22	0.257	-	-	0.901	29.40	0.245	-	-
Zip-NeRF [2]	0.828	28.54	0.189	-	-	-	-	-	-	-	-	-	-	-	-
3DGS [18]	0.815	27.47	0.216	27min6s	3.35	0.848	23.66	0.176	15min6s	1.84	0.904	29.54	0.244	24min51s	2.82
3DGS-accel [18, 31]	0.811	27.38	0.225	10min2s	2.39	0.849	23.61	0.175	6min53s	1.52	0.907	29.69	0.246	8min	2.31
Mini-Splatting-D [12]	0.831	27.51	0.176	32min50s	4.69	0.853	23.23	0.140	24min24s	4.28	0.906	29.88	0.211	28min26s	4.63
MSv2-D (Ours)	0.827	27.54	0.184	8min52s	3.59	0.851	23.38	0.151	5min55s	2.69	0.908	29.74	0.223	6min23s	4.11
Mini-Splatting [12]	0.822	27.32	0.217	20min21s	0.49	0.846	23.43	0.180	12min35s	0.30	0.910	29.98	0.241	17min16s	0.56
MSv2 (Ours)	0.821	27.33	0.215	3min34s	0.62	0.841	23.14	0.186	2min22s	0.35	0.912	30.08	0.240	2min45s	0.65



Figure 5. Qualitative results of our Mini-Splatting2 and other methods. LPIPS and optimization time are displayed as (LPIPS, Time).

Deep Blending [14]. Following the experimental protocol established by 3DGS [18], we apply consistent processing details across all datasets, including scene selection, train/test split, and image resolution. To ensure a fair evaluation, we assess rendering quality using the same implementations of key metrics: PSNR, SSIM, and LPIPS [42], as specified in the official 3DGS implementation [18].

**Implementation Variants and Details.** We implement our densification and culling in PyTorch, and modify the `preprocessCUDA` function within the rasterizer to support visibility culling. These modules are integrated into the Mini-Splatting framework to establish Mini-Splatting2, denoted as MSv2 (Ours). To construct the dense Gaussian model, we build Mini-Splatting2-D (MSv2-D) by replacing simplification in MSv2 with importance pruning, which preserves a relatively larger number of Gaussians. To manage memory consumption during MSv2 optimization, we disable SH coefficients during the densification stage, similar to Mini-Splatting, and initiate training with lower resolution (*i.e.*, 0.5× resolution) in this period.

Both of our Mini-Splatting2 variants, MSv2 and MSv2-D, incorporate improved implementations provided by Taming-3DGS [31]. Note that this enhanced codebase, re-

ferred to as 3DGS-accel, has recently been merged into the official 3DGS implementation, which we highly recommend as the new baseline for 3DGS. All experiments are conducted on a workstation equipped with an Intel Xeon Platinum 8383C CPU and a single RTX 3090 GPU.

## 6.2. Experimental Results

**Quantitative Results.** As shown in Table 1, we compare our Mini-Splatting2 (MSv2) and Mini-Splatting2-D (MSv2-D) against the baseline 3DGS, Mini-Splatting, and other NeRF-based algorithms in terms of training time and standard visual quality metrics. The table shows that our MSv2 achieves rendering quality comparable to the baseline 3DGS and 3DGS-accel, while significantly outperforming all other algorithms in optimization speed. Our dense Gaussian model, MSv2-D, which requires a longer but still reasonable training time, demonstrates superior rendering quality, with only a slight performance drop compared to its dense counterpart, Mini-Splatting-D. Overall, our MSv2 accelerates both the original 3DGS and the latest 3DGS-accel by factors of 7.6× and 2.8×, respectively, with similar rendering quality. Additionally, our MSv2-D provides a notable rendering quality improvement over 3DGS-accel without extending the optimization time budget.

Table 2. Resource consumption of our algorithms and other approaches on Mip-NeRF360, categorized into sparse and dense Gaussian models. Num and Peak represent the final and peak number of Gaussians (in millions) during optimization. Time indicates the total time required for optimization, and Memory indicates the peak GPU memory usage during optimization.

Method   Metric	Mip-NeRF360 (Outdoor)				Mip-NeRF360 (Indoor)			
	Num	Peak	Time	Memory	Num	Peak	Time	Memory
3DGS [18]	4.86	4.86	30min8s	7.45GB	1.46	1.46	24min41s	2.75GB
3DGS-accel [18, 31]	3.34	3.34	11min5s	5.03GB	1.29	1.29	10min50s	2.64GB
Mini-Splatting-D [12]	5.40	5.40	31min48s	7.45GB	3.80	3.80	40min13s	5.55GB
MSv2-D (Ours)	3.58	4.60	7min46s	6.60GB	3.57	4.86	11min	7.53GB
Mini-Splatting [12]	0.57	5.40	17min56s	2.61GB	0.40	3.80	27min2s	2.77GB
MSv2 (Ours)	0.65	4.61	3min20s	3.70GB	0.58	4.89	4min59s	4.55GB



Figure 6. Direct visualization of Mini-Splatting2 Gaussian centers and MVS points [33] in MeshLab.

**Qualitative Results.** The qualitative results of the *bicycle* and *truck* scenes are displayed as rendered images in Fig. 5. These results align well with the quantitative findings provided in Table 1. Notably, all Mini-Splatting-based algorithms, our MSv2, MSv2-D, Mini-Splatting, and Mini-Splatting-D, demonstrate improved visual performance over the baseline 3DGS-accel, particularly on the *bicycle* scan, where foreground regions display fewer blurry artifacts. This improvement stems from the foundational densification and simplification modules shared among these Mini-Splatting algorithms, such as depth reinitialization. The performance drop from MSv2-D to MSv2 mainly affects background regions, due to Mini-Splatting simplification that balances the number of Gaussians and rendering quality. Additional qualitative examples of background artifacts are provided in the appendix. In summary, compared to 3DGS-accel, MSv2 achieves similar qualitative results with substantial speed gains, while MSv2-D offers significant improvements in rendering outcomes without extending the optimization time. These findings underscore the effectiveness of our proposed Mini-Splatting2.

**Resource Consumption.** As shown in Table 2, we present the resource consumption results for our algorithm, including the final and peak number of Gaussians during optimization, training time, and peak GPU memory usage. Note that this experiment is specifically designed for limited-resource scenarios, thus source data is loaded into CPU memory, which slightly slows down training compared to Table 1.

For outdoor scenes, MSv2 incurs an increase in GPU memory consumption compared to Mini-Splatting due to the precomputed visibility. In indoor scenes, the peak number of Gaussians reveals that MSv2 tends to reconstruct more Gaussians than Mini-Splatting and 3DGS during optimization. This increase is due to our aggressive Gaussian densification strategy, which allocates more Gaussians to textureless regions, such as walls and floors, to enhancing rendering metrics. We note that the *kitchen* scan, which has the highest GPU memory demand, is reconstructed as 5.2 million Gaussians after densification with only 5.0GB peak memory consumption, owing to the SH disabling trick in Sec. 6.1. These results indicate that our algorithm remains accessible for most low-cost graphics cards.

Table 3. Ablation study on each component of Mini-Splatting2.

	SSIM $\uparrow$	PSNR $\uparrow$	LPIPS $\downarrow$	Train $\downarrow$	Num
3DGS-accel	0.811	27.38	0.225	10min2s	2.39
Mini-Splatting	0.822	27.32	0.217	20min21s	0.49
Mini-Splatting*	0.823	27.23	0.214	7min59s	0.56
+ Agg Dens	0.822	27.43	0.212	4min4s	0.64
+ Vis Culling	0.821	27.33	0.215	3min34s	0.62

From this experiment, we take a distinct perspective from Taming-3DGS [31]. Rather than constraining the peak number of Gaussians, we find that limiting SH coefficients (*i.e.*, 48 out of 59 optimizable Gaussian attributes) is a more practical and effective approach under limited resources. Furthermore, this experiment supports our aggressive densification and simplification approach over the non-aggressive one which simply constrains the Gaussian count throughout the optimization progress.

### 6.3. Point Cloud Reconstruction

To substantiate our assumption that Mini-Splatting2 supports dense Gaussian/point cloud reconstruction even in the early stages of optimization, we provide a direct and faithful visualization of Gaussian centers and MVS points [33] in MeshLab in Fig. 6. After a brief 3K-iteration densification and subsequent simplification (within 1 minute), Mini-Splatting2 effectively reconstructs the coarse scene geometry, albeit with some floaters. Continuing with an additional 5K-iteration optimization and further simplification (taking an additional 1 to 2 minutes), the algorithm produces a high-quality point cloud, approaching the fidelity of the dense MVS points, which typically require around 2 to 3 hours. As this study primarily targets fast optimization for novel view synthesis, we defer the enhancement and quantitative evaluation of geometry quality to future work, and the implementation of this part will also be released alongside Mini-Splatting2, aiming to inspire the community toward efficiency-driven reconstruction methods.

### 6.4. Ablation Studies

**Effectiveness of Components.** To isolate the impact of Mini-Splatting and 3DGS-accel on speed-up, we conduct an ablation study on the Mip-NeRF360 dataset, analyzing each component’s effectiveness in Table 3. Starting with the baseline, Mini-Splatting\*, which combines Mini-Splatting and 3DGS-accel, we incrementally add aggressive Gaussian densification (Agg Dens) and visibility Gaussian culling (Vis Culling). Comparison results show that our key component, aggressive Gaussian densification, reduces training time by half, while visibility Gaussian culling further accelerates optimization by an additional half-minute.

**Aggressive Gaussian Densification.** Table 4 presents the results of our ablation study on aggressive Gaussian densification. Starting with the MSv2 model, we compare sev-

Table 4. Ablation study of aggressive Gaussian densification.

	SSIM $\uparrow$	PSNR $\uparrow$	LPIPS $\downarrow$	Train $\downarrow$	Num
Mini-Splatting*	0.823	27.23	0.214	7min59s	0.56
MSv2 (Ours)	0.821	27.33	0.215	3min34s	0.62
w 3K Dens	0.776	26.49	0.285	2min23s	0.25
w Low Thres	0.799	26.58	0.237	3min43s	0.59
w Agg Dens-Depth	0.817	27.12	0.215	3min14s	0.62
w Agg Dens-Full	0.821	27.33	0.215	3min34s	0.62

eral densification variants: 3K Dens (a shortened progressive densification over 3K iterations), Low Thres (a lower gradient threshold for densification), Agg Dens-Depth (a predefined large number of points for depth initialization), and Agg Dens-Full (our complete proposed densification method). Both aggressive densification methods deliver substantial speed-ups over the Mini-Splatting\* baseline, with Agg Dens-Full providing adaptive density control and also enhancing rendering quality. This demonstrates the effectiveness of our aggressive densification approach in both scalability and quality. Additional ablation studies on critical Gaussian identification and aggressive Gaussian clone are provided in the appendix.

### 7. Limitation and Future Work

As discussed in Resource Consumption, Mini-Splatting2 prioritizes accelerated scene optimization without considering runtime memory constraints. Although the current implementation is manageable for most low-cost graphics cards, edge cases with limited resources present additional challenges [31]. Beyond controlling the peak number of Gaussians, memory consumption must also consider Gaussian attributes as well as dynamic memory allocation for rasterization. Besides, Mini-Splatting2 holds promise for efficient point cloud reconstruction. By Leveraging multi-view geometry techniques [13], it is possible to achieve an optimal balance between computational efficiency and geometric fidelity for dense 3D reconstruction.

### 8. Conclusion

In this work, we present Mini-Splatting2, a method designed for fast Gaussian Splatting optimization by introducing aggressive Gaussian densification and visibility Gaussian culling, significantly improving computational efficiency. Specifically, our aggressive densification strategy significantly increases the number of critical Gaussians early in optimization, while the visibility culling reduces computational demands by leveraging per-view Gaussian importance. Experimental results confirm the effectiveness of Mini-Splatting2 in balancing optimization speed, the number of Gaussians, and rendering quality, laying a strong foundation for future advancements in Gaussian Splatting-based scene reconstruction with substantial gains in both efficiency and quality.

## References

- [1] Jonathan T. Barron, Ben Mildenhall, Dor Verbin, Pratul P. Srinivasan, and Peter Hedman. Mip-NeRF 360: Unbounded Anti-Aliased Neural Radiance Fields. *CVPR*, 2022. 1, 5, 6, 12
- [2] Jonathan T Barron, Ben Mildenhall, Dor Verbin, Pratul P Srinivasan, and Peter Hedman. Zip-NeRF: Anti-Aliased Grid-Based Neural Radiance Fields. *ICCV*, 2023. 1, 6, 12
- [3] Samuel Rota Bulò, Lorenzo Porzi, and Peter Kontschieder. Revising Densification in Gaussian Splatting. In *ECCV*, 2024. 1, 2, 4, 11
- [4] David Charatan, Sizhe Li, Andrea Tagliasacchi, and Vincent Sitzmann. pixelSplat: 3D Gaussian Splats from Image Pairs for Scalable Generalizable 3D Reconstruction. In *CVPR*, 2024. 2
- [5] Anpei Chen, Haofei Xu, Stefano Esposito, Siyu Tang, and Andreas Geiger. LaRa: Efficient Large-Baseline Radiance Fields. In *ECCV*, 2024. 2
- [6] Yuedong Chen, Haofei Xu, Chuanxia Zheng, Bohan Zhuang, Marc Pollefeys, Andreas Geiger, Tat-Jen Cham, and Jianfei Cai. MVSSplat: Efficient 3D Gaussian Splatting from Sparse Multi-View Images. In *ECCV*, 2024. 2
- [7] Kai Cheng, Xiaoxiao Long, Kaizhi Yang, Yao Yao, Wei Yin, Yuexin Ma, Wenping Wang, and Xuejin Chen. Gaussian-Pro: 3D Gaussian Splatting with Progressive Propagation. In *ICML*, 2024. 1, 2
- [8] Jiadi Cui, Junming Cao, Yuhui Zhong, Liao Wang, Fuqiang Zhao, Penghao Wang, Yifan Chen, Zhipeng He, Lan Xu, Yujiao Shi, et al. LetsGo: Large-Scale Garage Modeling and Rendering via LiDAR-Assisted Gaussian Primitives. *arXiv preprint arXiv:2404.09748*, 2024. 2
- [9] Xiaobiao Du, Yida Wang, and Xin Yu. MVGS: Multi-view-regulated Gaussian Splatting for Novel View Synthesis. *arXiv preprint arXiv:2410.02103*, 2024. 1, 2
- [10] Sankeerth Durvasula, Adrian Zhao, Fan Chen, Ruofan Liang, Pawan Kumar Sanjaya, and Nandita Vijaykumar. DISTWAR: Fast Differentiable Rendering on Raster-based Rendering Pipelines. *arXiv preprint arXiv:2401.05345*, 2023. 2
- [11] Zhiwen Fan, Wenyang Cong, Kairun Wen, Kevin Wang, Jian Zhang, Xinghao Ding, Danfei Xu, Boris Ivanovic, Marco Pavone, Georgios Pavlakos, Zhangyang Wang, and Yue Wang. InstantSplat: Unbounded Sparse-view Pose-free Gaussian Splatting in 40 Seconds, 2024. 2
- [12] Guangchi Fang and Bing Wang. Mini-Splatting: Representing Scenes with a Constrained Number of Gaussians. In *ECCV*, 2024. 2, 3, 4, 5, 6, 7, 12
- [13] Richard Hartley. *Multiple View Geometry in Computer Vision*. Cambridge university press, 2003. 8
- [14] Peter Hedman, Julien Philip, True Price, Jan-Michael Frahm, George Drettakis, and Gabriel Brostow. Deep Blending for Free-Viewpoint Image-Based Rendering. *SIGGRAPH Asia*, 2018. 6
- [15] Lukas Hölein, Aljaž Božič, Michael Zollhöfer, and Matthias Nießner. 3DGS-LM: Faster Gaussian-Splatting Optimization with Levenberg-Marquardt. *arXiv preprint arXiv:2409.12892*, 2024. 3
- [16] Binbin Huang, Zehao Yu, Anpei Chen, Andreas Geiger, and Shenghua Gao. 2D Gaussian Splatting for Geometrically Accurate Radiance Fields. In *SIGGRAPH*. Association for Computing Machinery, 2024. 2
- [17] Nikhil Keetha, Jay Karhade, Krishna Murthy Jatavallabhula, Gengshan Yang, Sebastian Scherer, Deva Ramanan, and Jonathon Luiten. SplatTAM: Splat Track & Map 3D Gaussians for Dense RGB-D SLAM. In *CVPR*, pages 21357–21366, 2024. 2
- [18] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3D Gaussian Splatting for Real-Time Radiance Field Rendering. *TOG*, 42(4), 2023. 1, 2, 3, 4, 5, 6, 7, 11, 12
- [19] Shakiba Kheradmand, Daniel Rebain, Gopal Sharma, Weiwei Sun, Jeff Tseng, Hossam Isack, Abhishek Kar, Andrea Tagliasacchi, and Kwang Moo Yi. 3D Gaussian Splatting as Markov Chain Monte Carlo. *NeurIPS*, 2024. 1, 2, 4, 5, 11
- [20] Sieun Kim, Kyungjin Lee, and Youngki Lee. Color-cued Efficient Densification Method for 3D Gaussian Splatting. In *CVPR*, pages 775–783, 2024. 2
- [21] Arno Knapsch, Jaesik Park, Qian-Yi Zhou, and Vladlen Koltun. Tanks and Temples: Benchmarking Large-Scale Scene Reconstruction. *TOG*, 36(4), 2017. 5
- [22] Georgios Kopanas, Julien Philip, Thomas Leimkühler, and George Drettakis. Point-Based Neural Rendering with Per-View Optimization. *Computer Graphics Forum*, 2021. 2
- [23] Lingzhi Li, Zhen Shen, Zhongshu Wang, Li Shen, and Liefeng Bo. Compressing Volumetric Radiance Fields to 1 MB. In *CVPR*, pages 4222–4231, 2023. 4, 5
- [24] Zhuoxiao Li, Shanliang Yao, Yijie Chu, Angel F Garcia-Fernandez, Yong Yue, Eng Gee Lim, and Xiaohui Zhu. MVG-Splatting: Multi-View Guided Gaussian Splatting with Adaptive Quantile-Based Geometric Consistency Densification. *arXiv preprint arXiv:2407.11840*, 2024. 1, 2
- [25] Lingjie Liu, Jiatao Gu, Kyaw Zaw Lin, Tat-Seng Chua, and Christian Theobalt. Neural Sparse Voxel Fields. *NeurIPS*, 33:15651–15663, 2020. 2
- [26] Tianqi Liu, Guangcong Wang, Shoukang Hu, Liao Shen, Xinyi Ye, Yuhang Zang, Zhiguo Cao, Wei Li, and Ziwei Liu. Mvsgaussian: Fast generalizable gaussian splatting reconstruction from multi-view stereo. In *ECCV*, 2024. 2
- [27] Hidenobu Matsuki, Riku Murai, Paul HJ Kelly, and Andrew J Davison. Gaussian Splatting Slam. In *CVPR*, pages 18039–18048, 2024. 2
- [28] 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. *ECCV*, 2020. 1, 2, 3
- [29] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant Neural Graphics Primitives with a Multiresolution Hash Encoding. *SIGGRAPH*, 2022. 1, 2
- [30] Lukas Radl, Michael Steiner, Mathias Parger, Alexander Weinrauch, Bernhard Kerbl, and Markus Steinberger. StopThePop: Sorted Gaussian Splatting for View-Consistent Real-time Rendering. *TOG*, 43(4):1–17, 2024. 3
- [31] Saswat Mallick and Rahul Goel, Bernhard Kerbl, Francisco Vicente Carrasco, Markus Steinberger, and Fernando

- De La Torre. Taming 3DGS: High-Quality Radiance Fields with Limited Resources. In *SIGGRAPH Asia*, 2024. 1, 2, 6, 7, 8, 12
- [32] Johannes Lutz Schönberger and Jan-Michael Frahm. Structure-from-motion revisited. *CVPR*, 2016. 1, 2
- [33] Johannes Lutz Schönberger, Enliang Zheng, Marc Pollefeys, and Jan-Michael Frahm. Pixelwise View Selection for Unstructured Multi-View Stereo. In *ECCV*, 2016. 7, 8, 11, 13
- [34] Cheng Sun, Min Sun, and Hwann-Tzong Chen. Direct Voxel Grid Optimization: Super-fast Convergence for Radiance Fields Reconstruction. *CVPR*, 2022. 2
- [35] Stanislaw Szymanowicz, Eldar Insafutdinov, Chuanxia Zheng, Dylan Campbell, João F Henriques, Christian Ruprecht, and Andrea Vedaldi. Flash3D: Feed-Forward Generalisable 3D Scene Reconstruction from a Single Image. *arXiv preprint arXiv:2406.04343*, 2024. 2
- [36] Jiaxiang Tang, Jiawei Ren, Hang Zhou, Ziwei Liu, and Gang Zeng. DreamGaussian: Generative Gaussian Splatting for Efficient 3D Content Creation. *ICLR*, 2024. 2
- [37] Qiangeng Xu, Zexiang Xu, Julien Philip, Sai Bi, Zhixin Shu, Kalyan Sunkavalli, and Ulrich Neumann. Point-NeRF: Point-based Neural Radiance Fields. In *CVPR*, pages 5438–5448, 2022. 2
- [38] 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 *CVPR*, pages 19595–19604, 2024. 2
- [39] Vickie Ye, Ruilong Li, Justin Kerr, Matias Turkulainen, Brent Yi, Zhuoyang Pan, Otto Seiskari, Jianbo Ye, Jeffrey Hu, Matthew Tancik, et al. gsplat: An Open-Source Library for Gaussian Splatting. *arXiv preprint arXiv:2409.06765*, 2024. 2
- [40] Zongxin Ye, Wenyu Li, Sidun Liu, Peng Qiao, and Yong Dou. AbsGS: Recovering Fine Details for 3D Gaussian Splatting. In *ACMMM*, pages 1053–1061, 2024. 1, 2
- [41] Zehao Yu, Torsten Sattler, and Andreas Geiger. Gaussian Opacity Fields: Efficient Adaptive Surface Reconstruction in Unbounded Scenes. *TOG*, 2024. 2
- [42] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The Unreasonable Effectiveness of Deep Features as a Perceptual Metric. In *CVPR*, pages 586–595, 2018. 6
- [43] Zheng Zhang, Wenbo Hu, Yixing Lao, Tong He, and Hengshuang Zhao. Pixel-GS: Density Control with Pixel-aware Gradient for 3D Gaussian Splatting. In *ECCV*, 2024. 1, 2
- [44] Matthias Zwicker, Hanspeter Pfister, Jeroen Van Baar, and Markus Gross. EWA Splatting. *TVCG*, 8(3):223–238, 2002. 1, 2, 3

## Appendix

### A. Additional Ablation Studies

We further provide additional ablation studies on critical Gaussian identification and aggressive Gaussian clone.

#### A.1. Critical Gaussian Identification

Table 5 presents the results of our ablation study on critical Gaussian identification on the Mip-NeRF360 dataset. In this study, we compare different criteria for identifying critical Gaussians: random selection (Random), opacity (Opacity), accumulated blending weights ( $\sum_i w_i$ ) and maximum blending weights ( $w_{i_{max}}$ ). For each criterion, the predefined threshold is tuned to ensure a consistent number of Gaussians. The results demonstrate that weight-based criteria, specifically  $\sum_i w_i$  and  $w_{i_{max}}$ , perform better than other choices. Among these, maximum blending weights yields the best performance for our aggressive Gaussian densification approach.

While incorporating image-space information (e.g., 2D gradient and error) and inherent Gaussian attributes (e.g., opacity and scales) might further refine the identification criterion, such methods would introduce case-dependent and hand-crafted heuristics. To maintain simplicity and robustness, we adopt the maximum blending weights,  $w_{i_{max}}$ , as the basis for our critical Gaussian identification strategy.

Table 5. Ablation study of critical Gaussian identification.

	SSIM $\uparrow$	PSNR $\uparrow$	LPIPS $\downarrow$	Train $\downarrow$	Num
Random	0.819	27.27	0.218	3min47s	0.54
Opacity	0.820	27.28	0.215	3min49s	0.62
$\sum_i w_i$	0.820	27.34	0.215	3min46s	0.62
$w_{i_{max}}$	0.820	27.36	0.215	3min36s	0.62

#### A.2. Aggressive Gaussian Clone

Table 6 presents the results of our ablation study on aggressive Gaussian clone on the Mip-NeRF360 dataset. This study specifically compare the detailed densification operations. In Table 6, ‘Vanilla’ refers to the original Gaussian clone and split in 3DGS [18]. As noted in the main paper, these operations introduce a bias that implicitly amplifies the influence of densified Gaussians, thereby disrupting the optimization process [3]. To address this issue, our aggressive Gaussian clone only performs the clone operation and replaces the opacity of the resulting Gaussians, following [3], denoted as ‘Add  $\alpha^{new}$ ’, in Table 6. Additionally, we incorporate the replacement of the covariance matrix as described in [19], which is indicated as ‘Add  $\Sigma^{new}$ ’. As shown in Table 6, for our aggressive Gaussian densification, both modified densification operations achieve similar rendering quality, surpassing the performance of the vanilla one. For our aggressive Gaussian clone, we adopt the clone formulation proposed in [19], as it accommodates a more general

cloning scenario (i.e., cloning a single Gaussian into a pre-defined number of Gaussians).

Table 6. Ablation study of aggressive Gaussian clone.

	SSIM $\uparrow$	PSNR $\uparrow$	LPIPS $\downarrow$	Train $\downarrow$	Num
Vanilla	0.818	27.04	0.213	3min28s	0.68
Add $\alpha^{new}$ [3]	0.820	27.37	0.217	3min34s	0.59
Add $\Sigma^{new}$ [19]	0.820	27.36	0.215	3min36s	0.61

### B. Additional Qualitative Results

We provide additional qualitative results in Fig. 7. As illustrated, our MSv2-D consistently outperforms the 3DGS baseline. While our sparse Gaussian model, MSv2, achieves similar rendering metrics to 3DGS, it can introduce distortions in background regions. This issue arises from the simplification process, which employs a high sampling ratio (e.g., reducing approximately 3 million Gaussians to 0.6 million). To address this limitation, future work could explore the development of auxiliary visual metrics and optimization objectives that balance both visual quality and Gaussian quantity. Such strategies would help identify an optimal trade-off between rendering fidelity and model efficiency.

### C. Additional Quantitative Results

Table 7 presents additional quantitative results across all three datasets. Both our MSv2 and MSv2-D achieve competitive performance compared to their Mini-Splatting counterparts, while significantly reducing optimization time.

### D. Additional Point Reconstruction Results

Figure 8 illustrates the point cloud reconstruction results for additional seven scans from the Mip-NeRF360 dataset. The reconstructed point clouds align closely with the results presented in the main paper. Notably, our algorithm achieves this within 2 to 3 minutes, compared to the multi-view stereo (MVS) approach [33], which requires approximately 2 to 3 hours.

Overall, the Gaussian Splatting-based method demonstrates significant potential for highly efficient 3D reconstruction while maintaining high-quality results, particularly for foreground objects. However, our current implementation faces challenges in reconstructing background regions with sparse overlapping views. For these regions, the MVS approach provides superior and more robust reconstruction results. Integrating Gaussian Splatting with MVS to address this limitation remains an avenue for future research.



Figure 7. Additional qualitative results of our Mini-Splatting2.

Table 7. Quantitative evaluation of our Mini-Splatting2 and previous works.

	SSIM $\uparrow$												
	bicycle	flowers	garden	stump	treehill	room	counter	kitchen	bonsai	truck	train	johson	playroom
mip-NeRF 360 [1]	0.693	0.583	0.816	0.746	0.632	0.913	0.895	0.920	0.939	0.857	0.660	0.901	0.900
Zip-NeRF [2]	0.769	0.642	0.860	0.800	0.681	0.925	0.902	0.928	0.949	-	-	-	-
3DGs [18]	0.765	0.606	0.866	0.773	0.633	0.919	0.909	0.928	0.942	0.882	0.814	0.901	0.907
3DGs-accel [18, 31]	0.753	0.598	0.863	0.763	0.631	0.917	0.907	0.925	0.940	0.882	0.816	0.905	0.909
Mini-Splatting-D [12]	0.798	0.641	0.878	0.804	0.640	0.928	0.913	0.932	0.948	0.889	0.821	0.905	0.906
MSv2-D (Ours)	0.788	0.623	0.873	0.791	0.646	0.928	0.914	0.935	0.948	0.886	0.816	0.905	0.910
Mini-Splatting [12]	0.773	0.626	0.847	0.806	0.652	0.921	0.905	0.926	0.939	0.882	0.810	0.907	0.912
MSv2 (Ours)	0.771	0.612	0.850	0.797	0.651	0.922	0.908	0.929	0.944	0.875	0.806	0.910	0.914

	PSNR $\uparrow$												
	bicycle	flowers	garden	stump	treehill	room	counter	kitchen	bonsai	truck	train	johson	playroom
mip-NeRF 360 [1]	24.40	21.64	26.94	26.36	22.81	31.40	29.44	32.02	33.11	24.91	19.52	29.14	29.66
Zip-NeRF [2]	25.80	22.40	28.20	27.55	23.89	32.65	29.38	32.50	34.46	-	-	-	-
3DGs [18]	25.19	21.57	27.39	26.60	22.53	31.47	29.07	31.51	32.07	25.38	22.04	29.09	29.98
3DGs-accel [18, 31]	25.16	21.48	27.30	26.41	22.54	31.31	29.00	31.11	32.14	25.26	21.97	29.33	30.05
Mini-Splatting-D [12]	25.53	21.43	27.65	27.11	22.16	31.56	28.75	31.28	32.25	25.42	21.40	29.22	30.26
MSv2-D (Ours)	25.45	21.27	27.52	26.90	22.67	31.42	28.85	31.67	32.14	25.36	21.40	29.43	30.04
Mini-Splatting [12]	25.21	21.58	26.82	27.19	22.63	31.17	28.57	31.25	31.45	25.33	21.53	29.50	30.46
MSv2 (Ours)	25.25	21.37	26.80	27.10	22.76	31.24	28.59	31.29	31.72	24.97	21.29	29.60	30.47

	LPIPS $\downarrow$												
	bicycle	flowers	garden	stump	treehill	room	counter	kitchen	bonsai	truck	train	johson	playroom
mip-NeRF 360 [1]	0.289	0.345	0.164	0.254	0.338	0.211	0.203	0.126	0.177	0.159	0.354	0.237	0.252
Zip-NeRF [2]	0.208	0.273	0.118	0.193	0.242	0.196	0.185	0.116	0.173	-	-	-	-
3DGs [18]	0.211	0.336	0.107	0.215	0.324	0.219	0.200	0.126	0.203	0.147	0.207	0.245	0.244
3DGs-accel [18, 31]	0.233	0.345	0.114	0.230	0.338	0.225	0.202	0.129	0.205	0.146	0.204	0.242	0.250
Mini-Splatting-D [12]	0.158	0.255	0.091	0.169	0.263	0.188	0.171	0.116	0.175	0.101	0.179	0.220	0.205
MSv2-D (Ours)	0.173	0.276	0.098	0.184	0.275	0.190	0.173	0.114	0.175	0.114	0.189	0.224	0.221
Mini-Splatting [12]	0.225	0.327	0.150	0.198	0.314	0.213	0.198	0.129	0.200	0.139	0.222	0.244	0.239
MSv2 (Ours)	0.220	0.331	0.142	0.205	0.321	0.207	0.193	0.125	0.190	0.150	0.222	0.238	0.242

	Optimization Time (s) $\downarrow$												
	bicycle	flowers	garden	stump	treehill	room	counter	kitchen	bonsai	truck	train	johson	playroom
3DGs [18]	2070	1455	2156	1660	1487	1435	1416	1712	1246	1064	748	1677	1307
3DGs-accel [18, 31]	740	499	768	538	519	494	559	855	452	448	379	555	406
Mini-Splatting-D [12]	1842	1700	1974	1668	1671	2072	2417	2379	2014	1494	1435	1817	1597
MSv2-D (Ours)	458	407	577	401	376	591	654	742	587	348	363	438	329
Mini-Splatting [12]	957	1020	1021	958	1027	1377	1642	1611	1381	747	765	1097	976
MSv2 (Ours)	171	165	205	157	153	241	279	298	256	135	147	180	151

	Num of Gaussians (million)												
	bicycle	flowers	garden	stump	treehill	room	counter	kitchen	bonsai	truck	train	johson	playroom
3DGs [18]	6.10	3.63	5.84	4.79	3.89	1.55	1.20	1.81	1.26	2.60	1.09	3.31	2.32
3DGs-accel [18, 31]	3.97	2.49	3.48	3.85	2.58	1.17	1.06	1.74	1.16	1.96	1.07	2.91	1.71
Mini-Splatting-D [12]	6.04	4.86	5.82	5.45	4.85	3.93	3.75	3.69	3.75	4.57	3.94	4.91	4.34
MSv2-D (Ours)	3.85	3.23	4.23	3.66	3.13	3.20	3.45	3.60	3.96	2.48	2.90	5.18	3.05
Mini-Splatting [12]	0.53	0.57	0.56	0.61	0.57	0.39	0.41	0.43	0.36	0.32	0.28	0.60	0.51
MSv2 (Ours)	0.68	0.61	0.73	0.67	0.58	0.52	0.55	0.60	0.62	0.34	0.36	0.80	0.49



Figure 8. Additional visualization of Mini-Splatting2 Gaussian centers and MVS points [33] in MeshLab.