

# Relaxing Accurate Initialization Constraint for 3D Gaussian Splatting

Jaewoo Jung<sup>\*</sup>, Jisang Han<sup>\*</sup>, Honggyu An<sup>\*</sup>,  
Jiwon Kang<sup>\*</sup>, Seonghoon Park<sup>\*</sup>, Seungryong Kim

Korea University  
<https://ku-cvlab.github.io/RAIN-GS>



Figure 1: **Effectiveness of our simple strategy.** Left and right show the results from 3DGS [1] and ours trained with randomly initialized point cloud respectively. Transition from 3DGS to ours simply requires our strategy consisted of sparse-large-variance (SLV) random initialization, progressive Gaussian low-pass filtering, and adaptive bound-expanding split (ABE-Split) algorithm.

## Abstract

3D Gaussian splatting (3DGS) has recently demonstrated impressive capabilities in real-time novel view synthesis and 3D reconstruction. However, 3DGS heavily depends on the accurate initialization derived from Structure-from-Motion (SfM) methods. When the quality of the initial point cloud deteriorates, such as in the presence of noise or when using randomly initialized point cloud, 3DGS often undergoes large performance drops. To address this limitation, we propose a novel optimization strategy dubbed **RAIN-GS** (**R**elaxing **A**ccurate **I**nitalization **C**onstraint for 3D **G**aussian **S**platting). Our approach is based on an in-depth analysis of the original 3DGS optimization scheme and the analysis of the SfM initialization in the frequency domain. Leveraging simple modifications based on our analyses, **RAIN-GS** successfully trains 3D Gaussians from sub-optimal point cloud (e.g., randomly initialized point cloud), effectively relaxing the need for accurate initialization. We demonstrate the efficacy of our strategy through quantitative and qualitative comparisons on multiple datasets, where **RAIN-GS** trained with random point cloud achieves performance on-par with or even better than 3DGS trained with accurate SfM point cloud.

<sup>\*</sup>: Equal contribution.

## 1 Introduction

Novel view synthesis is one of the essential tasks in computer vision and computer graphics, aiming to render novel views of a 3D scene given a set of images. It has a wide range of applications in various fields, including augmented reality and virtual reality [2], robotics [3], and data generation [4]. Neural radiance fields (NeRFs) [5] have demonstrated remarkable success in this task, learning implicit representations that capture intricate 3D geometry and specular effects solely from images. However, NeRFs’ reliance on multi-layer perceptrons (MLPs) [6–10] results in slow and computationally intensive volume rendering, hindering real-time applications.

Recently, 3D Gaussian splatting (3DGS) [1] has emerged as a compelling alternative for both high-quality results and real-time rendering. Unlike NeRFs training an MLP to model the scene with an implicit representation, 3DGS models the scene using explicit 3D Gaussians. The learned 3D Gaussians are then projected to the image space using an efficient CUDA-based differentiable tile rasterization [11, 1] technique which enables the rendering of the Gaussians in real-time.

Despite its remarkable results, 3DGS requires an additional input of initial point cloud compared to NeRFs. The quality of the initial point cloud is one of the essential requirements of 3DGS, showing large performance drops when trained with randomly initialized point cloud [1]. A similar type of performance degradation can occur in real-world scenarios, specifically in scenes where SfM techniques struggle to converge, such as scenes with symmetry, and textureless regions [12, 13]. Also, accurate initial point cloud become a strict requirement even for situations where camera poses can be obtained through external sensors [14, 15] or pre-defined such as in text-to-3D generation [16, 17]. In these cases, where the initial point cloud from SfM may contain noise or even unavailable, 3DGS requires an additional module to obtain the point cloud, adding extra computation cost [18, 19].

In this work, we start with a natural question: “*Why is accurate initial point cloud so important in 3D Gaussian splatting?*” and conduct an in-depth analysis of the original 3DGS optimization scheme and SfM initialization. Our analyses reveal that the original 3DGS optimization scheme struggles to *transport* Gaussians further from their initialized locations, leading to performance degradation when starting from random or noisy SfM initialization. In addition, the analysis of the SfM initialization in the frequency domain reveals that SfM initialization provides the low-frequency components of the true distribution. From the provided low-frequency components, 3DGS clones and splits existing Gaussians, which is then used to learn the remaining high-frequency details of the scene [1]. In this process, the low-frequency components guide the Gaussians to successfully learn the target distribution in a *coarse-to-fine* manner utilizing the clone/split process, without the need to transport the Gaussians to correct locations. Based on this observation, we propose that to train 3DGS with noisy or random point cloud, we need an effective strategy that can transport the Gaussians further from their initial positions and encourage the Gaussians to prioritize the learning of the coarse structure information, which will then guide the overall optimization process to learn in a similar coarse-to-fine manner.

Based on our analysis of accurate initial point cloud, we propose a novel optimization strategy called **RAIN-GS** (**R**elaxing **A**ccurate **I**Nitilization **C**onstraint for 3D **G**aussian **S**platting), which is composed of three key components: **1**) a novel initialization method starting with sparse Gaussians with large variance, **2**) progressive Gaussian low-pass filtering utilized in the rendering process, and **3**) a novel adaptive bound-expanding split (ABE-Split) algorithm utilized in the adaptive density control. We demonstrate that RAIN-GS effectively relaxes the requirement of accurate initialization for 3DGS, as applying RAIN-GS to random point cloud successfully prioritizes the learning of low-frequency components and guides the Gaussians to learn in a *coarse-to-fine* manner, and enables the Gaussians to traverse further from their initialized locations. After optimization, RAIN-GS applied to random point cloud achieves on-par or even better results compared to 3DGS trained with SfM. By effectively relaxing the requirement of initial point cloud, RAIN-GS opens up new possibilities for applying 3DGS to settings where obtaining accurate initial point cloud is challenging.

## 2 Related work

### 2.1 Novel view synthesis

Neural radiance fields (NeRF) [5] have succeeded in significantly boosting the performance of novel view synthesis by optimizing an MLP that can estimate the density and radiance of any continuous

3D coordinate. With the camera poses of the given images, NeRF learns the MLP by querying dense points along randomly selected rays, that outputs the density and color of each of the queried coordinates. Various follow-ups [20–27] adopted NeRF as their baseline model and further extend the ability of NeRF to model unbounded or dynamic scenes [20, 21, 24], lower the required number of images for successful training [26, 27], learn a generalizable prior to resolve the need of per-scene optimization [22, 23], or utilize an external hash-grid to fasten the overall optimization process [25]. Although all of these works show compelling results, the volume rendering from dense points along multiple rays makes NeRF hard to apply in real-time settings achieving lower rendering rates of < 1 fps.

Recently, 3D Gaussian splatting (3DGS) [1] has been recognized as a new solution for the task of novel view synthesis, succeeding in high-quality real-time novel-view synthesis at 1080p resolution with frame rates exceeding 90 fps. 3DGS achieves this by modeling the 3D scene with explicit 3D Gaussians, followed by an efficient and differentiable tile rasterization implemented in CUDA to accelerate the rendering process. Due to the efficient and fast architecture of 3DGS, it gained massive attention from various fields [28, 17, 29, 16], extended to model dynamic scenes [29, 28], or used as an alternative for text-to-3D tasks [16, 17] for fast generation. Nevertheless, the essential requirement of both accurate pose and initial point cloud limits the application of 3DGS to only scenes where SfM can provide accurate point cloud. In this work, we analyze the optimization strategy of 3DGS and present a simple yet effective change in the optimization strategy to expand the versatility of 3DGS to successfully learn the scene without accurate initialization.

## 2.2 Structure-from-Motion (SfM)

SfM techniques [30–32] have been one of the most widely used algorithms to reconstruct a 3D scene. Typical SfM methods output the pose of each input image and a sparse point cloud that includes rough color and position information for each point. These methods employ feature extraction, matching steps, and bundle adjustment. For the task of novel view synthesis, NeRF [5] utilizes the estimated pose from the SfM and trains an implicit representation to model the scene. The recently proposed 3DGS [1] utilizes both the accurate pose and initial point cloud as an initialization for the position and color of 3D Gaussians, showing large performance drops when the initial point cloud becomes noisy or not accessible.

Despite the effectiveness of SfM in generating accurately aligned point cloud and precise camera poses, its incremental nature and the computational intensity of the bundle adjustment process significantly increase its time complexity, often to  $O(n^4)$  with respect to  $n$  cameras involved [33]. Due to the high computational demand, SfM being a hard prerequisite limits the feasibility of NeRF or 3DGS for scenarios that require real-time processing. In addition, SfM often struggles to converge, such as in scenes with symmetry, specular properties, and textureless regions, and when the available views are limited. In these cases, an additional module is required to provide accurate initial point cloud to 3DGS, which can take longer times than SfM [19].

## 3 Preliminary: 3D Gaussian splatting (3DGS)

Recently, 3DGS [1] has emerged as a promising alternative for the novel view synthesis task, modeling the scene with multiple 3D Gaussians [34]. Each  $i$ -th Gaussian  $G_i$  represents the scene with the following attributes: a position vector  $\mu_i \in \mathbb{R}^3$ , an anisotropic covariance matrix  $\Sigma_i \in \mathbb{R}^{3 \times 3}$ , spherical harmonic (SH) coefficients [35, 25], and an opacity logit value  $\alpha_i \in [0, 1]$ . With these attributes, each Gaussian  $G_i$  is defined in the world space [34] as follows:

$$G_i(x) = e^{-\frac{1}{2}(x-\mu_i)^T \Sigma_i^{-1} (x-\mu_i)}. \quad (1)$$

To render an image from a pose represented by the viewing transformation  $W$ , the projected covariance  $\Sigma'_i$  is defined as follows:

$$\Sigma'_i = JW\Sigma_i W^T J^T, \quad (2)$$

where  $J$  is the Jacobian of the local affine approximation of the projective transformation. The 2D covariance matrix is simply obtained by skipping the third row and column of  $\Sigma'_i$  [34]. Finally, to render the color  $C(p)$  of the pixel  $p$ , 3DGS utilizes alpha blending according to the Gaussians depth.

Table 1: **Effect of noise on the initial SfM point cloud.**

Init.	PSNR↑	SSIM↑	LPIPS↓
SfM	27.205	0.815	0.214
SfM + $\epsilon$	24.944	0.771	0.270
SfM + constant	26.204	0.761	0.277

Table 2: **Analysis of Gaussians movement on Truck scene.**

	3DGS (SfM)	3DGS (Random)	Ours (Random)
Means	0.349	0.184	12.100
Stds	0.726	0.297	18.600
Top 1%	3.418	1.524	82.721

For example, when  $N$  Gaussians are sorted by depth, the color  $C(p)$  is calculated as follows:

$$C(p) = \sum_{i=1}^N c_i \alpha_i G'_i(p) \prod_{j=1}^{i-1} (1 - \alpha_j G'_j(p)), \quad (3)$$

where  $c_i$  is the view-dependent color value of each Gaussian calculated with the SH coefficients, and  $G'_i$  is the 3D Gaussian projected to the 2D screen space.

Since 3DGS starts learning from sparse point cloud, they utilize an adaptive density control to obtain a more accurate and denser representation. By considering if the scene is under-/over-constructed, each of the Gaussians undergoes a clone/split operation. Gaussians that have small covariance are cloned and Gaussians considered to be too large are splitted into two new smaller Gaussians, where the means of new Gaussians are sampled from the probability density function (PDF) of the original Gaussian. When the Gaussian is considered to be too small, the Gaussian is cloned and simply located in the mean of the original Gaussian.

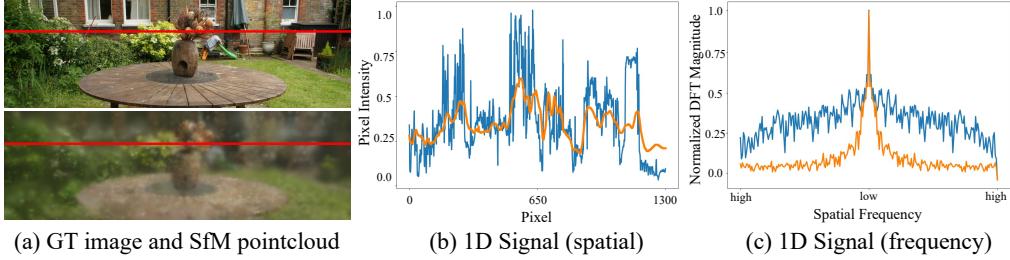
## 4 Motivation

Although point cloud can be noisy or unavailable in real-world scenarios [12, 13], 3DGS shows large performance drops depending on the accuracy of the initial point cloud [1]. To understand the large performance gap of 3DGS, we conduct an in-depth analysis of the original 3DGS optimization scheme and the analysis of SfM initialization in the frequency domain. Our analyses reveal two important characteristics of the optimization scheme of 3DGS: **1)** the optimization scheme of 3DGS struggles to transport Gaussians from their initialized locations and **2)** the coarse structure information (low-frequency components) provided by the accurate initialization enables the adaptive density control method of 3DGS to robustly model the remaining fine details of the scene in a coarse-to-fine manner.

**3DGS lacks the ability to transport Gaussians.** To represent and learn the scene with explicit 3D Gaussians, 3DGS first initializes the Gaussians  $G_i$  in the world space, whose means  $\mu_i$  are defined by the initial point cloud. The point cloud can be either achieved from SfM or initialized randomly.

As mentioned in [36], the process of fitting a 3DGS model is similar to fitting a Gaussian Mixture Model (GMM), which is well-known for being non-convex and generally solved with the Expectation-Maximization (EM) algorithm [37]. They further note that, similar to the EM algorithm, training 3DGS from randomly initialized point cloud becomes prone to falling into local minima due to two main reasons. **1)** The Gaussians can only receive gradients close to their means, mostly from the range not exceeding the distance of a few standard deviations, and **2)** there is no existing path for the Gaussians that will decrease the loss monotonically.

Although [36] only analyzes the case of starting from random initialization, we verify that Gaussians can also easily fall into local minima when SfM-initialized point cloud become noisy. As shown in Table 1, we find that adding a small constant noise or adding a small noise  $\epsilon$  sampled from a normal distribution ( $\epsilon \sim \mathcal{N}(0, 1)$ ) to the SfM-initialized point cloud, leads to large performance drops. Based on these observations, we hypothesize that the optimization scheme of 3DGS lacks the ability to correct or move the positions of the Gaussians. We empirically verify our hypothesis by calculating the average distance each Gaussian traversed after optimization, as shown in Table 2. It can be seen that the average distance each Gaussian moved is close to zero, indicating that the optimization scheme of 3DGS lacks the ability to move Gaussians, which can lead to the failure of capturing objects located far from the initial positions of the Gaussians. This emphasizes the need for a strategy that can enable the Gaussians to transport further from their initialized locations, in order to successfully train 3DGS from sub-optimal initializations.



**Figure 2: Analysis of SfM initialization in 3DGS.** (a) The top shows the GT image, and the bottom is the rendered image by 3DGS after only 10 steps with SfM initialization. We can observe that the rendered image is already coarsely-close to GT image. We randomly sample a horizontal line from the image marked in red. (b) The pixel intensity along this line are shown, with the GT indicated in blue and the rendered image in orange. (c) This graph visualizes the magnitude of the frequency components of (b). Since frequencies further from the middle of the x-axis represent high-frequency components, we observe that SfM provides *coarse* approximation of the true distribution.

**Accurate initialization guides 3DGS to learn in a coarse-to-fine manner.** To further investigate the benefits of SfM initialization, we analyze the rendered images in the frequency domain using Fourier transform [38]. As shown in Figure 2, the analysis in the frequency domain demonstrates that SfM initialization provides a coarse approximation of the target distribution.

As the goal of novel view synthesis is to understand the 3D distribution of the scene, it is necessary to model both low- and high-frequency components of the true distribution. However, prior NeRF frameworks [39, 40, 27] argue that NeRF is prone to overfitting and naïve optimization leads to over-fast convergence of high-frequency components, expressed with high-frequency artifacts in the rendered image. To circumvent this problem, they adopt a coarse-to-fine learning strategy, which regularizes NeRF to learn the low-frequency components first. Similarly, prior works [41, 42] utilizing GMMs for the task of point cloud registration or generation also mention that naïve fitting of GMMs can result in converging to local minima. In order to robustly train GMMs, they also adopt a coarse-to-fine strategy, implemented by starting with a small number of Gaussians and recursively increasing the number of total Gaussians. In both NeRFs and GMMs, coarse-to-fine strategy guides the network to learn more robustly, leading to better performance.

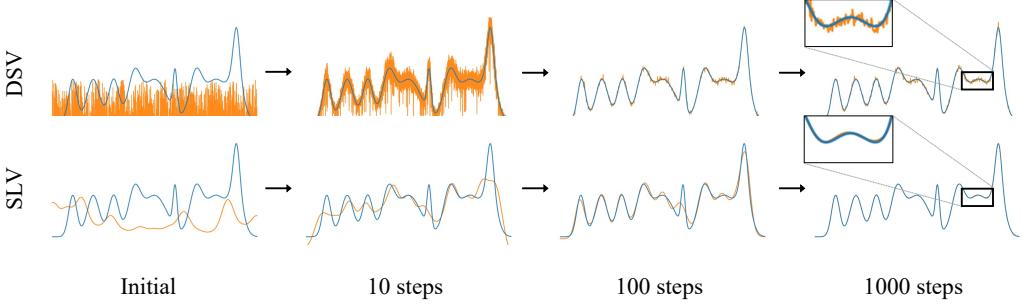
In this perspective, starting the optimization of 3DGS from SfM-initialized point cloud can be understood as benefitting from a similar coarse-to-fine process, where SfM provides the low-frequency components (Figure 2), and the adaptive density control method of 3DGS adds the Gaussians to learn the remaining high-frequency details. Based on our observations, the success of 3DGS from accurate initialization can be attributed to the low-frequency components guiding the overall training process, preventing the Gaussians from falling into local minima. This highlights the need for a strategy that can prioritize the learning of the low-frequency components even from sub-optimal initializations, which will then be used to guide the remaining optimization process of 3DGS.

## 5 Methodology

Following our motivation (Section 4), we propose a novel optimization strategy, dubbed **RAIN-GS** (**R**elaxing **A**ccurate **I**Nitilization **C**onstraint for 3D **G**aussian **S**platting), which enables the Gaussians to prioritize the learning of low-frequency components, and enables the Gaussians to transport further from their initialized locations. Our strategy consists of three key components: **1**) sparse-large-variance (SLV) initialization (Section 5.1), **2**) progressive Gaussian low-pass filtering (Section 5.2), and **3**) a novel adaptive bound-expanding split (ABE-Split) algorithm (Section 5.3).

### 5.1 Sparse-large-variance (SLV) initialization

Drawing inspiration from GMMs [41, 18], which gradually increase the number of Gaussians to accurately model target point cloud, we observe that the adaptive density control of 3DGS can be viewed as a similar process. Through cloning and splitting operations, 3DGS generally increases the number of Gaussians to find the adequate number of Gaussians required to represent the scene. Based on our findings, we hypothesize that initializing 3DGS with a sparse set of Gaussians will prioritize the learning of low-frequency components, akin to the progressive refinement approach employed by GMMs. This sparse initialization strategy is expected to capture the overall structure of



**Figure 3: Toy experiment to analyze different initialization methods.** This figure visualizes the result of our toy experiment predicting the target distribution using a collection of 1D Gaussians, starting from different initialization methods.

the target point cloud in the early stages of the optimization process, with finer details being added as the number of Gaussians increases.

To verify our hypothesis, we conduct a toy experiment in a simplified 1D regression task. Following the original 3DGS which can be interpreted as the learning process of a 3D target distribution with multiple Gaussians, we use  $N$  Gaussians each with learnable means, variances, and weights, which are then blended to model a 1D target signal. Specifically, we follow the initialization methods of 3DGS [1], where the means are initialized randomly and the variances are initialized based on the distances of the three nearest neighbors. As a result, sparse initialization of Gaussians leads to a larger initial covariance (SLV) and dense initialization leads to a smaller covariance (DSV). To verify our hypothesis that learning with sparse Gaussians will prioritize the learning of low-frequency components, we conduct our toy experiment using  $N = 15$  and  $N = 1000$  for the SLV and DSV initialization respectively. Note that our 1D toy experiment without the adaptive density control method of 3DGS provides a controlled environment isolating the effects of initialization. A detailed explanation of our toy experiment can be found in Section C of the supplementary materials.

As shown in Figure 3, SLV initialization prioritizes the learning of low-frequency components compared to DSV initialization verifying our hypothesis. After 1,000 steps, SLV also shows a better prediction of the target distribution. Similar results can be observed when SLV is applied to 3DGS, as lowering the number of initial Gaussians  $N$  in randomly initialized settings significantly improves performance. Following the random initialization method of [1], which randomly samples point cloud from a scene extent defined as three times the bounding box of the camera poses, SLV prioritizes the learning of low-frequency components, producing fewer high-frequency artifacts. Surprisingly, SLV becomes more effective even until extremely sparse settings (e.g., as low as  $N = 10$ ), verifying the effectiveness of our novel SLV initialization method.

## 5.2 Progressive Gaussian low-pass filter control

Although our SLV initialization method is effective, we find that after multiple densification steps, the number of 3D Gaussians increases exponentially due to the adaptive density control, which can collapse into similar problems with the DSV initialization. In order to guide the 3D Gaussians to learn in a coarse-to-fine manner and let the 3D Gaussians sufficiently explore the low-frequency components during the early stage of training, we propose a novel progressive control of the Gaussian low-pass filter which is utilized in the rendering stage.

**Gaussian low-pass filter for 3DGS.** In the rendering stage of 3DGS, the 2D Gaussian  $G'_i$  projected from a 3D Gaussian  $G_i$  is defined as follows:

$$G'_i(x) = e^{-\frac{1}{2}(x-\mu'_i)^T \Sigma'_i^{-1} (x-\mu'_i)}. \quad (4)$$

However, directly using projected 2D Gaussians can lead to visual artifacts when they become smaller than the size of a single pixel [1, 43]. To ensure coverage of at least one pixel, [1] enlarge the 2D Gaussian's scale by adding a small value to the covariance's diagonal elements as follows:

$$G'_i(x) = e^{-\frac{1}{2}(x-\mu'_i)^T (\Sigma'_i + sI)^{-1} (x-\mu'_i)}, \quad (5)$$

where  $s$  is a pre-defined value of  $s = 0.3$  and  $I$  is an identity matrix. This process can also be interpreted as the convolution between the projected 2D Gaussian  $G'_i$  and a Gaussian low-pass filter

$h$  (mean  $\mu = 0$  and variance  $\sigma^2 = 0.3$ ) of  $G'_i \otimes h$ , which is shown to be an essential step to prevent aliasing [34]. After applying convolution with the low-pass filter, the area of the projected Gaussian  $G'_i$  is approximated by a circle. The radius of this circle is defined by three times the larger eigenvalue from the 2D covariance matrix  $(\Sigma'_i + sI)^1$ .

**Progressive low-pass filter control.** Instead of using a fixed value of  $s$  through the entire optimization process, we notice that this value  $s$  can ensure the minimum area each Gaussians have to cover in the screen space. As Gaussians only receive gradients inside the range of a few standard deviations [36], learning from wider areas is essential for the Gaussians to learn the coarse structure information of the scene. Therefore, we control  $s$  to regularize the Gaussians to cover wider areas during the early stage of training and progressively learn from a more local region. Specifically, as the value  $s$  ensures the projected Gaussians area to be larger than  $9\pi s^2$ , we define the value  $s$  such that  $s = HW/9\pi N$ , where  $N$  indicates the number of Gaussians and  $H, W$  indicates the height and width of the image respectively.

### 5.3 Adaptive bound-expanding split (ABE-Split) algorithm

As discussed in Section 4, in 3DGS, Gaussians struggle to model scenes far from their initialized locations. Through visualizations of the positions of the Gaussians after training, we empirically find that Gaussians particularly struggle to model scenes *outside* the initial bound defined by the starting point cloud positions (see Section D.5 of the supplementary materials for visualizations).

This type of behavior is largely attributed to the characteristics of the adaptive density control method of 3DGS [1], which applies a clone/split algorithm to the Gaussians. As the new Gaussian is sampled from the PDF of the original Gaussian, it is likely to be located near the original Gaussians, which leads to an increased density of Gaussians near initial locations. Consequently, 3DGS more easily models scenes *inside* the initial bound compared to those *outside* of it, highlighting the need for a strategy that can encourage the Gaussians to move outside the initial bound.

Based on these observations, we propose the adaptive bound-expanding split algorithm, a novel splitting algorithm that can encourage the Gaussians to move outside the initial bound during early steps. Specifically, ABE-Split algorithm splits each of the Gaussians into three, where two of the Gaussians undergo the same split process as the original split algorithm. However, for the additional third Gaussian, rather than sampling from the PDF of the original Gaussian, we initialize the new location by multiplying a constant value (proportional to the size of the scene extent). Also, we retain the same scale as the original Gaussian without dividing it. This is to encourage the transported Gaussians to receive gradients from larger regions in their new positions. We provide a more detailed explanation of ABE-Split in Section A in the supplementary materials.

### 5.4 Analysis of RAIN-GS

All three key components of our strategy can be interpreted as a component that acts as the “warm-up” step for the Gaussians, which guides the Gaussians to first learn the coarse structure of the scene. SLV initialization and progressive Gaussian low-pass filtering facilitate coarse-to-fine optimization whereas ABE-Split enables the Gaussians to transport to further locations. We demonstrate that our strategy successfully guides 3DGS to learn the coarse structure without any given information (e.g., accurate initial point cloud) which can widen the application of 3DGS to real-world scenarios where the initial point cloud can be noisy or absent. Specifically, as RAIN-GS does not have any restrictions or conditions for the initial point cloud other than being sparse (SLV initialization), we can apply our strategy to both point cloud achieved from SfM [32] or random. The effectiveness and versatility of our strategy are further evaluated in Section 6.

## 6 Experiments

### 6.1 Experimental Settings

**Datasets.** To assess the effectiveness of our strategy, we conduct qualitative and quantitative comparisons on the Mip-NeRF360 dataset [21], Tanks&Temples dataset [44], and Deep Blending

---

<sup>1</sup>We provide a detailed proof in Section B.1 in the supplementary materials.

<sup>2</sup>We provide a detailed proof in Section B.2 in the supplementary materials.

Table 3: **Quantitative comparison on Mip-NeRF360, Tanks&Temples and Deep Blending datasets in different initial points conditions.** We compare our method with 3DGS under three different initial point conditions: SfM points, noisy SfM points, and the absence of initial points.

Methods	Init. points	Mip-NeRF360 Outdoor Scene														
		bicycle			flowers			garden			stump			treehill		
		PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓
3DGS [1]	SfM	25.246 (0.957)	0.771 (0.079)	0.205 (+0.102)	21.520 (0.492)	0.605 (0.040)	0.336 (0.038)	27.410 (0.895)	0.868 (0.022)	0.103 (+0.030)	26.550 (0.522)	0.775 (0.022)	0.210 (+0.042)	22.490 (0.762)	0.638 (0.031)	0.317 (+0.063)
	Noisy SfM	24.289 (0.957)	0.692 (0.079)	0.307 (+0.102)	21.028 (0.492)	0.565 (0.040)	0.374 (0.038)	26.515 (0.895)	0.846 (0.022)	0.133 (+0.030)	26.028 (0.522)	0.743 (0.022)	0.252 (+0.042)	21.728 (0.762)	0.607 (0.031)	0.380 (+0.063)
	Random	21.034 (0.421)	0.575 (0.242)	0.378 (+0.173)	17.815 (3.705)	0.469 (-22.48)	0.403 (+0.067)	23.217 (4.193)	0.783 (0.085)	0.175 (+0.072)	20.745 (5.805)	0.618 (0.157)	0.345 (+0.135)	18.986 (3.504)	0.550 (0.088)	0.413 (+0.096)
Ours	Random	25.042	0.747	0.238	21.762	0.616	0.324	26.884	0.854	0.114	26.680	0.768	0.215	22.528	0.621	0.342
Methods	Init. points	Mip-NeRF360 Indoor Scene												Mip-NeRF360		
		PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	Average SSIM↑	LPIPS↓	
3DGS [1]	SfM	30.632 (0.068)	0.914 (0.013)	0.220 (+0.040)	28.700 (-4.828)	0.905 (0.067)	0.204 (+0.081)	30.317 (5.840)	0.922 (0.045)	0.129 (+0.046)	31.980 (6.157)	0.938 (0.063)	0.205 (+0.060)	27.205 (2.281)	0.815 (0.044)	0.214 (+0.056)
	Noisy SfM	30.740 (0.947)	0.901 (0.020)	0.260 (+0.045)	23.872 (5.092)	0.838 (0.072)	0.285 (+0.072)	24.477 (4.239)	0.877 (0.059)	0.175 (+0.052)	25.823 (13.642)	0.875 (0.219)	0.265 (+0.196)	24.944 (5.015)	0.771 (0.111)	0.270 (+0.099)
	Random	29.685 (0.947)	0.894 (0.020)	0.265 (+0.045)	23.608 (5.092)	0.833 (0.072)	0.276 (+0.072)	26.078 (4.239)	0.893 (0.059)	0.161 (+0.052)	18.538 (13.642)	0.719 (0.219)	0.401 (+0.196)	22.190 (5.015)	0.704 (0.111)	0.313 (+0.099)
Ours	Random	30.809	0.906	0.247	28.529	0.895	0.223	31.270	0.920	0.137	31.547	0.934	0.218	27.228	0.807	0.229
Methods	Init. points	Tanks&Temples						Deep Blending								
		PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	Playroom	SSIM↑	LPIPS↓
3DGS [1]	SfM	25.187 (0.880)	0.879 (0.001)	0.148 (+0.047)	21.097 (0.885)	0.802 (0.040)	0.218 (0.040)	28.766 (0.162)	0.899 (0.056)	0.244 (+0.057)	30.044 (0.079)	0.906 (0.059)	0.241 (+0.051)			
	Noisy SfM	23.367 (0.880)	0.838 (0.001)	0.195 (+0.047)	20.212 (0.885)	0.753 (0.040)	0.284 (0.040)	27.084 (0.162)	0.882 (0.057)	0.283 (+0.059)	30.194 (0.079)	0.901 (0.059)	0.252 (+0.051)			
	Random	20.149 (0.038)	0.758 (0.121)	0.248 (+0.080)	20.824 (0.275)	0.772 (0.030)	0.255 (0.027)	28.668 (0.098)	0.894 (0.085)	0.258 (+0.014)	28.358 (1.886)	0.896 (0.010)	0.258 (+0.017)			
Ours	Random	24.816	0.865	0.169	21.436	0.786	0.244	28.675	0.896	0.260	30.165	0.903	0.250			

dataset [45] previously utilized in the evaluation of the 3DGS method [1]. We evaluate the PSNR, LPIPS, and SSIM metrics by constructing a train/test split using every 8th image for testing, as suggested in Mip-NeRF360 [21]. We use the same image resolution as 3DGS.

**Tasks.** As mentioned in Section 4, SfM can fail in real-world scenarios where the point cloud become noisy or sometimes inaccessible. To assess the robustness of 3DGS [1] in these scenarios, we first compare our strategy with 3DGS each trained with SfM-initialized point cloud, SfM-initialized point cloud with added noise  $\epsilon$  ( $\epsilon \sim N(0, 1)$ ), and random point cloud, where our strategy only utilizes random point cloud (Table 3). In addition, to assess the performance of our strategy, we compare our strategy with Plenoxels [35], InstantNGP-Base, InstantNGP-Big [25], and 3DGS [1] (Table 4). In the latter experiment, we evaluate 3DGS trained with SfM-initialized point cloud, and random point cloud. For our strategy, we train RAIN-GS with SfM-initialized point cloud and random point cloud.

## 6.2 Implementation details

We implement our model based on 3DGS [1]. We follow the same training process of the existing implementation in all datasets. We apply our strategy to both randomly initialized point cloud and SfM-initialized point cloud. When applying RAIN-GS to randomly initialized point cloud (Table 3, Table 4), we set the initial number of Gaussians to  $N = 10$ . When applying RAIN-GS to SfM-initialized point cloud (Table 4), we select a sparse set from the entire point cloud. We either use the top 10% with the least reprojection error (denoted as <10%) or use cluster centers (denoted as cluster) obtained through HDBSCAN [46]. As RAIN-GS guides the Gaussians to learn the coarse structure of the scene in the early stages of training, we set the first 10,000 steps as the warm-up phase and extend the densification steps by 10,000 steps. During the warm-up phase, the learning rate of the mean  $\mu$  is unchanged, and ABE-Split replaces the original split algorithm. Additional details are included in supplementary materials A.

## 6.3 Quantitative comparison with 3DGS in real-word scenarios

As mentioned above, SfM might fail or produce noisy point cloud in real-world scenarios. To assess the robustness of 3DGS [1] in these cases, we evaluate 3DGS trained with SfM points, noisy SfM points, and random points. The results are compared with ours trained with random points.

As shown in Table 3, 3DGS is highly dependent on the accuracy of initial point cloud, showing large performance drops when trained with noisy SfM point cloud (denoted as Noisy SfM) and random point cloud (denoted as Random). In contrast, our strategy trained from random point cloud shows on-par or even better results compared to 3DGS trained with accurate SfM point cloud.

**Table 4: Quantitative comparison on Mip-NeRF360, Tanks&Temples and Deep Blending datasets.** We compare our method with previous approaches, including the random initialization method described in the original 3DGS [1]. ‘Cluster’ and ‘<10%’ means our model utilizing clustered SfM points and top 10% of points with the least reprojection error, respectively.

Method	SfM points	Mip-NeRF360 Outdoor Scene														
		bicycle			flowers			garden			stump			treehill		
		PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓
Plenoxels [35]	✗	21.912	0.496	0.506	20.097	0.431	0.521	23.495	0.606	0.386	20.661	0.523	0.503	22.248	0.509	0.540
INGP-Base [25]	✗	22.193	0.491	0.487	20.348	0.450	0.481	24.599	0.649	0.312	23.626	0.574	0.450	22.364	0.518	0.489
INGP-Big [25]	✗	22.171	0.512	0.446	20.652	0.486	0.441	25.069	0.701	0.257	23.466	0.594	0.421	22.373	0.542	0.450
3DGS [1]	✗	21.034	0.575	0.378	17.815	0.469	0.403	23.217	0.783	0.175	20.745	0.618	0.345	18.986	0.550	0.413
3DGS [1]	✓	<b>25.246</b>	<b>0.771</b>	<b>0.205</b>	21.520	0.605	0.336	<b>27.410</b>	<b>0.868</b>	<b>0.103</b>	26.550	<b>0.775</b>	0.210	22.490	<b>0.638</b>	<b>0.317</b>
Ours	✗	25.042	0.747	0.238	<b>21.762</b>	0.616	0.324	26.884	0.854	0.114	26.680	0.768	0.215	22.528	0.621	0.342
Ours	cluster	25.211	0.767	0.211	21.704	0.617	0.321	27.166	0.862	<b>0.103</b>	26.816	0.775	0.204	22.589	0.630	0.328
Ours	<10%	25.214	0.769	0.208	<b>21.858</b>	<b>0.619</b>	<b>0.320</b>	27.130	0.861	0.107	<b>26.899</b>	<b>0.777</b>	<b>0.202</b>	<b>22.618</b>	0.632	0.327
Method	SfM points	Mip-NeRF360 Indoor Scene												Mip-NeRF360		
		PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	Average	PSNR↑	SSIM↑
Plenoxels [35]	✗	27.594	0.842	0.419	23.624	0.759	0.441	23.420	0.648	0.447	24.669	0.814	0.398	23.080	0.625	0.462
INGP-Base [25]	✗	29.269	0.855	0.301	26.439	0.798	0.342	28.548	0.818	0.254	30.337	0.890	0.227	25.302	0.671	0.371
INGP-Big[25]	✗	29.690	0.871	0.261	26.691	0.817	0.306	29.479	0.858	0.195	30.685	0.906	0.205	25.586	0.699	0.331
3DGS [1]	✗	29.685	0.894	0.265	23.608	0.833	0.276	26.078	0.893	0.161	18.538	0.719	0.401	22.190	0.704	0.313
3DGS [1]	✓	30.632	<b>0.914</b>	<b>0.220</b>	28.700	<b>0.905</b>	<b>0.204</b>	30.317	0.922	<b>0.129</b>	<b>31.980</b>	<b>0.938</b>	<b>0.205</b>	27.205	<b>0.815</b>	<b>0.214</b>
Ours	✗	30.809	0.906	0.247	28.529	0.895	0.223	<b>31.270</b>	0.920	0.137	31.547	0.934	0.218	27.228	0.807	0.229
Ours	cluster	30.363	0.909	0.233	28.594	0.901	0.211	31.136	<b>0.923</b>	0.133	31.695	<b>0.935</b>	0.217	27.253	0.813	0.218
Ours	<10%	<b>30.843</b>	0.912	0.230	<b>28.703</b>	0.902	0.209	<b>31.364</b>	0.922	0.133	31.829	<b>0.938</b>	0.214	<b>27.384</b>	<b>0.815</b>	0.217
Methods	SfM points	Tanks&Temples						Deep Blending								
		PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	DrJohnson	Playroom	
Plenoxels [35]	✗	23.221	0.774	0.335	18.927	0.663	0.422	23.142	0.787	0.521	22.980	0.802	0.465			
INGP-Base [25]	✗	23.260	0.779	0.274	20.170	0.666	0.386	27.750	0.839	0.381	19.483	0.754	0.465			
INGP-Big[25]	✗	23.383	0.800	0.249	20.456	0.689	0.360	28.257	0.854	0.352	21.665	0.779	0.428			
3DGS [1]	✗	21.149	0.758	0.248	20.824	0.772	0.255	28.668	0.894	0.258	28.358	0.896	0.258			
3DGS [1]	✓	<b>25.187</b>	<b>0.879</b>	<b>0.148</b>	21.097	<b>0.802</b>	<b>0.218</b>	<b>28.766</b>	<b>0.899</b>	<b>0.244</b>	30.044	<b>0.906</b>	<b>0.241</b>			
Ours	✗	24.816	0.865	0.169	<b>21.436</b>	0.786	0.244	28.675	0.896	0.260	<b>30.165</b>	0.903	0.250			
Ours	cluster	24.893	0.869	0.160	21.180	0.786	0.243	28.529	0.895	0.260	29.994	0.902	0.250			
Ours	<10%	<b>25.190</b>	0.873	0.159	<b>21.741</b>	0.796	0.235	28.567	0.896	0.261	30.065	0.902	0.249			



Figure 4: Qualitative results on Mip-NeRF360, Tanks&Temples, and Deep Blending dataset.

#### 6.4 Qualitative and quantitative comparison

In Table 4, we compare our model with Plenoxels [35], InstantNGP-Base, InstantNGP-Big [25], and 3DGS [1] on the Mip-NeRF360 dataset [21], Tanks&Temples dataset [44], and Deep Blending dataset [45]. All NeRF methods [35, 25, 20] are trained without SfM points. We evaluate 3DGS [1] and ours trained with SfM-initialized and randomly initialized point cloud. For ours trained with SfM-initialized point cloud, we evaluate utilizing different clustering methods of selecting the top 10% with the least reprojection error (denoted as <10%) or using cluster centers (denoted as cluster) obtained through HDBSCAN [46].

In Figure 4, we provide qualitative results on novel test views. We compare with InstantNGP-Base (a)), 3DGS trained with random point cloud ((b)) and SfM initialized point cloud ((c)), and ours

trained with random point cloud ((d)). By comparing (b) and (d), we demonstrate the effectiveness of our strategy, which removes any unwanted high-frequency artifacts or floaters and also successfully captures scene objects located far away from the viewpoint. By comparing (c) and (d), ours trained with random point cloud show better results than 3DGS trained with SfM initialized point cloud, capturing fine details more effectively and rendering the background more accurately. More qualitative images are included in Section E in the supplementary materials.

### 6.5 Ablation studies

In Table 5, we validate the effectiveness of each component in our method trained in the Mip-NeRF360 dataset [21] with randomly initialized point cloud. The comparison of Sparse-large-variance (SLV) and dense-small-variance (DSV) initialization is done by using  $N = 10$  and  $N = 1,000,000$  respectively. We compare our progressive Gaussian low-pass filter control method with using a constant low-pass filter value ( $s = 0.3$ ) as done in the original 3DGS [1]. We also compare utilizing our novel ABE-Split method and the original split algorithm utilized in 3DGS. All three ablations show that using our components achieves significantly better results and using all three key components achieves the best results. More detailed and various ablation studies can be found in Section D.2 in the supplementary materials.

**Table 5: Ablation on core components**

Low-pass filter	Init.	ABE-Split	PSNR↑	SSIM↑	LPIPS↓
Constant	DSV	✗	22.190	0.704	0.313
Constant	SLV	✗	25.675	0.749	0.288
Ours	SLV	✗	26.180	0.761	0.280
<b>Ours</b>	<b>SLV</b>	<b>✓</b>	<b>27.228</b>	<b>0.807</b>	<b>0.229</b>

## 7 Conclusion

In this work, we introduce **RAIN-GS** (**R**elaxing **A**ccurate **I**Nitilization Constraint for 3D **GS**

## Supplementary materials

In the supplementary materials, we provide a more detailed analysis of our experiments and implementation details, together with additional results of rendered images using our strategy. Specifically, in Section A, we provide the implementation details of how our **RAIN-GS** (**R**elaxing **A**ccurate **I**Nitiation Constraint for 3D **GSs (diagonal values of the covariance matrix  $sI$  of the Gaussian low-pass filter) for our progressive Gaussian low-pass filter control. In Section C, we explain the details of our toy experiment shown in Section 5.1 of our main paper: learning to regress 1D signal with multiple 1D Gaussians. In Section D, we provide additional analysis including comparisons of computational resources with different methods and limitations of our strategy. In Section E, we provide additional qualitative results. In Section F, we show how our strategy can be applied to training 3DGS with a limited number of images.**

## A Implementation Details

### A.1 RAIN-GS

We implement our strategy based on the official code of 3DGS [1]. As mentioned in Section 6.2 of our main paper, except for the hyperparameters for increasing spherical harmonic (SH) degrees and 3D Gaussian scale division ratio, we follow the original implementation details and hyperparameters of 3DGS [1]. We start increasing SH degree every 1,000 steps after 5,000 steps and we lower 3D Gaussian scale division ratio 1.6 to 1.4. Our novel strategy consists of three components: **1)** sparse-large-variance (SLV) random initialization, **2)** progressive Gaussian low-pass filter control and **3)** ABE-Split. While 3DGS begins with the SfM point cloud which provides a coarse approximation of the scene, RAIN-GS starts with significantly fewer Gaussians and without any coarse information. Consequently to achieve an initial coarse approximation, it requires more time and steps to densify the Gaussians. Thus we add warm-up phase in training and our three main components are only applied during the warm-up phase. Specifically, the traditional 3DGS training can be divided into two main phases: (1) densification and (2) optimization. The densification phase typically involves about 15,000 steps, followed by 15,000 steps of optimization. The learning rate of the mean  $\mu$  start with  $1.6e^{-4}$  and exponentially decay to  $1.6e^{-6}$  during densification phase. For RAIN-GS, however, the optimization phase is reduced to 5,000 steps, and 10,000 steps of warm-up phase is introduced prior to densification phase. While warm-up phase, the learning rate of the mean  $\mu$  is kept  $1.6e^{-4}$  and our components are applied to help the Gaussians learn a coarse approximation. Then, the remaining process follows the same learning process as the original 3DGS.

#### A.1.1 Sparse-large-variance (SLV) random initialization

Following the original implementation of 3DGS [1], we initialize random point cloud within the boundary defined as three times the size of the camera’s bounding box. Instead of the original dense-small-variance random initialization where the initial number of Gaussians  $N$  is set as  $N > 100K$ , we only initialize 10 Gaussians as  $N = 10$ . As the initial covariance of 3D Gaussian is defined based on the distances of the three nearest neighbors, sparse initialization leads to a larger initial covariance, resolving the requirement for additional modification. This simple change of code (expressed with a gray box in Algorithm 1) largely improves the performance of 3DGS based on our analysis of sparse-large-variance initialization. An illustrations of SfM initialization, dense-small-variance (DSV) initialization and sparse-large-variance (SLV) initialization are shown in Figure 5.

#### A.1.2 Progressive Gaussian low-pass filter control

In the original implementation of 3DGS, the Gaussian low-pass filter is used in the rendering stage to ensure the projected 2D Gaussians to cover at least one pixel in the screen space. To enlarge the 2D Gaussians, 3DGS uses a fixed Gaussian low-pass filter (mean  $\mu = 0$  and variance  $\sigma^2 = s = 0.3$ ). Instead of using a fixed sigma value  $\sigma$  for this low-pass filter, we propose a progressive Gaussian low-pass filter control, where the sigma value starts with a large value and progressively reduces to  $\sigma^2 = 0.3$ . This can be efficiently implemented. Instead of passing a fixed value of  $s = 0.3$  as the

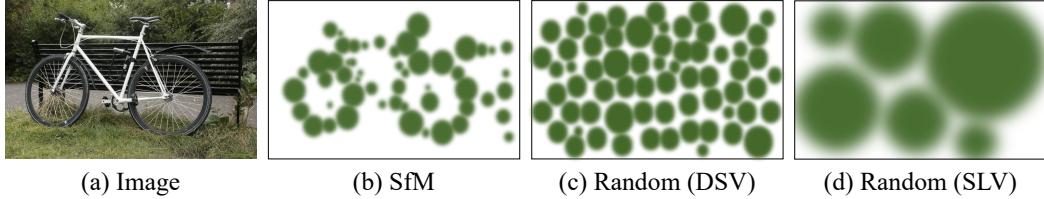


Figure 5: **Visualization of different initialization methods.** This figure illustrates the effect of different initialization methods. **(a)** Ground truth image. **(b)** Initialized point cloud from Structure-from-Motion (SfM). **(c)** Dense-small-variance (DSV) random initialization with small initial covariances due to the short distance between Gaussians. **(d)** Sparse-large-variance (SLV) random initialization with large initial covariances due to wider distance between Gaussians.

---

**Algorithm 1** Sparse-large-variance (SLV) random initialization

---

```

1:  $N \leftarrow$  Number of initial Gaussians (e.g.,  $N = 10$ )
2:  $P \leftarrow$  Cameras
3:  $D \leftarrow \text{MaxCameraDistance}(P)$ 
4:  $i \leftarrow 0$  ▷ Iteration Count
5: while  $i < N$  do
6:    $\mu \leftarrow \text{RandomCubicSample}(3 \times D)$ 
7:    $\Sigma \leftarrow \text{AvgNeighborDistance}()$ 
8:    $c \leftarrow \text{RandomInit}()$ 
9:    $\alpha \leftarrow \text{InverseSigmoid}(1)$ 
10:   $(M, S, C, A) \leftarrow (\mu, \Sigma, c, \alpha)$  ▷ Appends
11:   $i \leftarrow i + 1$ 
12: end while
13: return  $(M, S, C, A)$ 

```

---

diagonal values of the covariance matrix  $sI$  of the Gaussian low-pass filter, we pass a progressive value adaptively defined with the image height, width, and number of Gaussians taken into account as  $\min(\max(HW/9\pi N, 0.3), 300.0)$  every 1,000 steps. Our implementation is expressed in line 8 of Algorithm 2.

### A.1.3 ABE-Split

In addition to the split of 3DGS, the ABE-Split generates an additional Gaussian and transport it to a distant location. Specifically, new position is determined by calculating the vector between the center of bounding box and the original Gaussian position, then multiplying it by 0.3 times the scene extent. The pseudo code for this process is provided in line 21 and 22 of Algorithm 2. Due to our SLV initialization and progressive Gaussian low-pass filtering, the Gaussians are sparse and have large covariance during the early steps of training, resulting in a majority of the Gaussians only undergoing the split process. Thus, the ABE-Split can transport a sufficient number of Gaussians in early stage.

---

**Algorithm 2** Progressive Gaussian low-pass filter control & ABE-Split

---

```

1:  $M \leftarrow$  SfM Points ▷ Positions
2:  $S, C, A \leftarrow$  InitAttributes() ▷ Covariances, Colors, Opacities
3:  $s \leftarrow$  Scene Extent
4:  $B_0 \leftarrow$  Center Point of Bounding Box
5:  $i \leftarrow 0$  ▷ Iteration Count
6: while not converged do
7:    $V, \hat{I}, H, W \leftarrow$  SampleTrainingView() ▷ Camera  $V$ , Image, Height and Width
8:   if LowPassRefinementIteration( $i$ ) then
9:      $h \leftarrow \text{Min}(\text{Max}(HW/9\pi N, 0.3), 300)$  ▷ Progressive low-pass filter control
10:    end if
11:     $I \leftarrow \text{Rasterize}(M, S, C, A, V, h)$  ▷ Rasterization with low-pass filter
12:     $L \leftarrow \text{Loss}(I, \hat{I})$  ▷ Loss
13:     $M, S, C, A \leftarrow \text{Adam}(\nabla L)$  ▷ Backprop & Step
14:    if IsRefinementIteration( $i$ ) then
15:      for all Gaussians  $(\mu, \Sigma, c, \alpha)$  in  $(M, S, C, A)$  do
16:        if  $\alpha < \epsilon$  or IsTooLarge( $\mu, \Sigma$ ) then ▷ Pruning
17:          RemoveGaussian()
18:        end if
19:        if  $\nabla_p L > \tau_p$  then ▷ Densification
20:          if  $\|S\| > \tau_S$  then ▷ Over-reconstruction
21:            SplitGaussian( $\mu, \Sigma, c, \alpha$ )
22:             $\mu' \leftarrow \lambda_s s \cdot (\mu - B_0) + B_0$ 
23:            CloneGaussian( $\mu', \Sigma, c, \alpha$ ) ▷ ABE-Split
24:          else ▷ Under-reconstruction
25:            CloneGaussian( $\mu, \Sigma, c, \alpha$ )
26:          end if
27:        end if
28:      end for
29:    end if
30:     $i \leftarrow i + 1$ 
31: end while

```

---

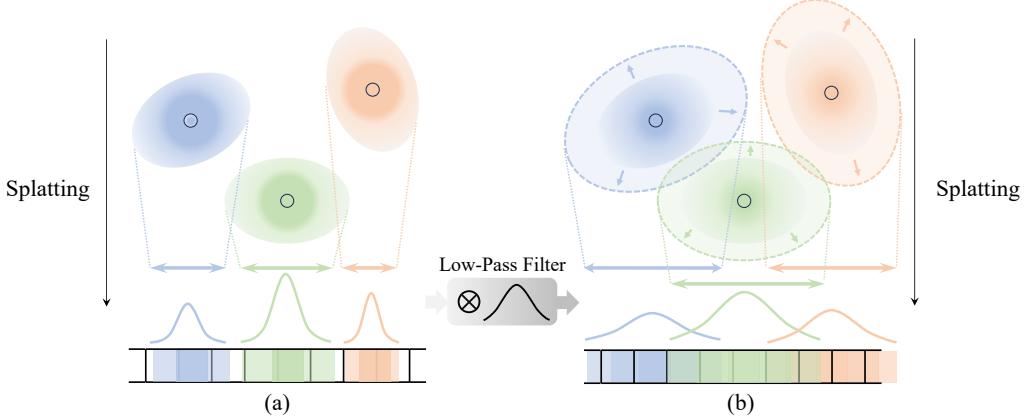


Figure 6: **Visualization of low-pass filter.** This figure shows the visualization of the effect of the low-pass filter. As shown in (b), the convolution of the splatted 2D Gaussian with the low-pass filter expands the area the Gaussian is splatted onto, resulting in the Gaussians affecting larger areas than naïve splatting as shown in (a).

## B Proof

### B.1 Proof on radius of a Gaussian convolved with a low-pass filter

As mentioned in Section 5.2 of our main paper, the 3D Gaussians  $G_i$  is projected to 2D Gaussians  $G'_i$  in the screen space as follows:

$$G'_i(x) = e^{-\frac{1}{2}(x-\mu'_i)^T \Sigma'_i^{-1}(x-\mu'_i)}. \quad (6)$$

To ensure the 2D Gaussian  $G'_i$  to cover at least one pixel, 3DGS adds a small value  $s$  to the diagonal elements of the 2D covariance  $\Sigma'_i$  as follows:

$$G'_i(x) = e^{-\frac{1}{2}(x-\mu'_i)^T (\Sigma'_i + sI)^{-1}(x-\mu'_i)}, \quad (7)$$

where  $I$  is the  $2 \times 2$  identity matrix. This process can be understood as the convolution between the 2D Gaussian  $G'_i$  and the Gaussian low-pass filter  $h$  (mean  $\mu = 0$  and variance  $\sigma^2 = s = 0.3$ ) of  $G'_i \otimes h$ . This is due to the nature of Gaussians where the convolution of Gaussians with the variance matrices  $V$  and  $Z$  results in a Gaussian with the variance matrix  $V + Z$  as follows:

$$G_1(x) = e^{-\frac{1}{2}(x-\mu_1)^T V^{-1}(x-\mu_1)} \quad G_2(x) = e^{-\frac{1}{2}(x-\mu_2)^T Z^{-1}(x-\mu_2)}, \quad (8)$$

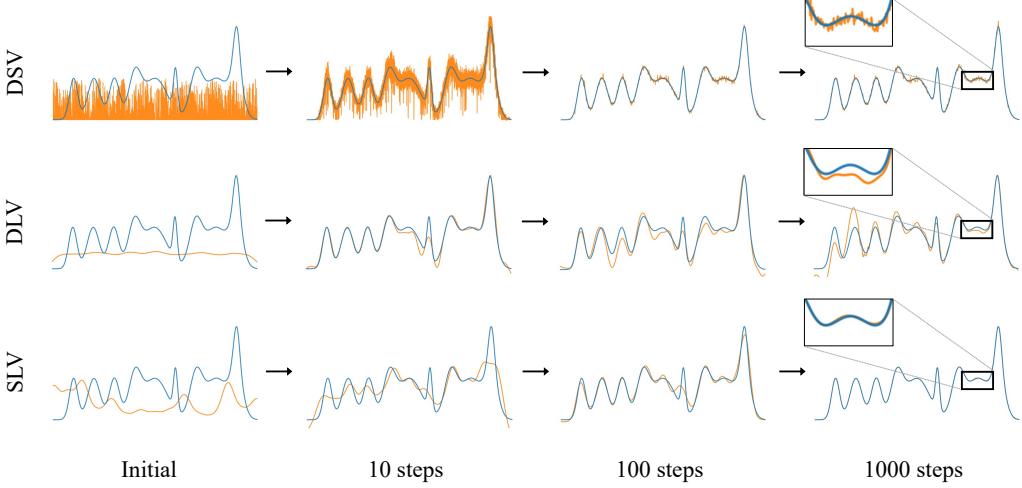
$$(G_1 \otimes G_2)(x) = e^{-\frac{1}{2}(x-\mu_1)^T (V+Z)^{-1}(x-\mu_1)}. \quad (9)$$

Following the convolution process, 3DGS estimates the projected 2D Gaussian's area to identify its corresponding screen tiles. This is done by calculating  $k$  times the square root of the larger eigenvalue of  $(\Sigma'_i + sI)$ , which represents the radius of the approximated circle, and  $k$  is the hyperparameter that determines the confidence interval of the 2D Gaussian. Figure 6 illustrates the low-pass filter's effect, where the projected Gaussian is splatted to wider areas in (b) compared to (a).

### B.2 Proof on progressive low-pass filter size

In Section 5.2 of our main paper, we define the value  $s$  for our progressive Gaussian low-pass filter control based on the fact that the area of the projected 2D Gaussians is at least  $9\pi s$ . As the area of the projected 2D Gaussian is defined as the circle whose radius is  $k$  times the square root of the larger eigenvalue of  $(\Sigma'_i + sI)$ , we have to first calculate the eigenvalues of  $(\Sigma'_i + sI)$ . If we define the eigenvalues of  $\Sigma_i$  as  $\lambda_{i1}, \lambda_{i2}$ , since the eigenvalue of  $sI$  is  $s$ , the eigenvalues of  $(\Sigma'_i + sI)$  can be defined as  $\lambda_{i1} + s, \lambda_{i2} + s$ . This leads to the following proof:

$$\begin{aligned} r &= k \cdot \sqrt{\max(\lambda_{i1}, \lambda_{i2}) + s}, \\ r &\geq k \cdot \sqrt{s}, \\ \pi r^2 &\geq k^2 \pi s, \end{aligned} \quad (10)$$



**Figure 7: Toy experiment to analyze different initialization methods.** This figure visualizes the result of our toy experiment predicting the target distribution using a collection of 1D Gaussians, starting from different initialization methods. Dense-small-variance (DSV) and dense-large-variance (DLV) initialize 1,000 1D Gaussians, where DLV is initialized with large variances by adding  $s$  to the initial variance. Small-large-variance (SLV) initializes 15 1D Gaussians with the same  $s$  added to the initial variance. We can observe the over-fast convergence of high-frequency components on DSV initialization, which is resolved in the DLV initialization but fails to converge due to fluctuation. SLV initialization resolves both problems, learning the low-frequency components first and also converging to successfully model the target distribution.

where  $k$  is the hyperparameter that defines the confidence interval of the Gaussian. We follow the original implementation of 3DGS as  $k = 3$  which gives the 99.73% confidence interval. Using the value  $k = 3$  leads to the proof of the area of each Gaussian being at least  $9\pi s$ .

## C Toy experiments

### C.1 Implementation details

In this section, we provide a detailed explanation of the 1D regression toy experiment shown in Section 5.1 of our main paper. We design the experiment to learn a target distribution  $Y(x)$ , where  $x$  is in the range  $[0, 10,000]$ , using  $N$  1D Gaussians. Each of the  $N$  1D Gaussians includes its mean  $\mu_i$  and variance  $\sigma_i^2$  as learnable parameters to model a random 1D signal  $Y(x)$  as the ground truth distribution. The target distribution  $Y(x)$  is generated from random 10 1D Gaussian distributions. By blending  $N$  1D Gaussians with a learnable weight  $w_i$ , we train the parameters  $\{\mu_i, \sigma_i, w_i\}$  through the L1 loss between the blended distribution from  $N$  1D Gaussians and  $Y(x)$  as follows:

$$\mathcal{L} = \sum_x \left\| Y(x) - \sum_{i=0}^N w_i \cdot \exp \left( -\frac{(x - \mu_i)^2}{2\sigma_i^2} \right) \right\|, \quad (11)$$

For the dense-small-variance (DSV) scenario, we initialize the number of Gaussians as  $N = 1,000$  and randomly select  $\mu_i$  and  $\sigma_i$  values within the range  $[0, 1)$ . For the dense-large-variance (DLV) scenario, we use the same number of Gaussians as  $N = 1,000$  but choose  $\mu_i$  and  $\sigma_i$  values randomly from the range  $[300, 301]$ . For the sparse-large-variance (SLV) scenario, we set the number of initial Gaussians as  $N = 15$  and randomly select  $\mu_i$  and  $\sigma_i$  values from the range  $[300, 301]$ . In all scenarios, we train the parameters for a total of 1,000 steps using the Adam optimizer with a learning rate of 0.01.

## C.2 Analysis on toy experiments

As mentioned in Section 5.1, we conduct a toy experiment in a simplified 1D regression task to examine how DSV and SLV initialization influences the optimization process. As shown in the top row of Figure 7, training with the DSV random initialization exhibits a tendency towards over-fast convergence on high frequencies. This is evident in the model’s ability to capture high-frequency components after only 10 steps. However, this rapid focus on high-frequencies leads to undesired high-frequency artifacts in the final reconstruction (zoomed-in box at 1,000 steps). We hypothesize that this behavior arises from the small initial variances of the dense Gaussians, which restrict each Gaussian to model a very localized region, increasing the susceptibility to overfitting.

To verify our hypothesis, we repeat the experiment with a value  $s$  added to the initial variance changing our initialization to dense-large-variance (DLV). Note that this modification effectively encourages the Gaussians to learn from wider areas. The results (middle row of Figure 7) support our hypothesis: the learned signal at 10 steps demonstrates a prioritization of low-frequency components when compared to DSV initialization. However, even with the ability to prune Gaussians by learning weights as  $w_i = 0$ , the dense initialization causes fluctuations in the learned signal, preventing convergence after 1,000 steps. These observations highlight a crucial point: while learning from wider areas (enabled by a large variance) is necessary to prioritize low-frequency components, a dense initialization can still lead to instability and convergence issues.

To resolve both problems, we propose a sparse-large-variance (SLV) initialization. The sparsity reduces fluctuations throughout the optimization, while the large variance ensures initial focus on the low-frequency distribution. This is demonstrated in the bottom row of Figure 7, where the experiment is repeated with  $N = 15$  Gaussians with  $s$  added to the initial variance. SLV initialization succeeds in both the prioritization of low-frequency components at 10 steps and the modeling of the true distribution with minimal errors after 1,000 steps.

Therefore, although initializing random points from the same volume which is defined as three times the size of the camera’s bounding box, we initialize a significantly sparser set of 3D Gaussians. As the initial covariance of 3D Gaussian is defined based on the distances of the three nearest neighbors, sparse initialization leads to a larger initial covariance, encouraging each Gaussian to model a wider region of the scene.

## C.3 Analysis on dense-large-variance (DLV) initialization in 3DGS

We compare the influence of different dense initialization methods through the toy experiment of learning to regress a random 1D signal with multiple 1D Gaussians. Through the toy experiment, we analyzed that while Gaussian learning from wider areas (enabled by the large initial variance) leads to prioritizing low-frequency components, the dense initialization results in instability and convergence issues. As shown in Figure 7, starting from Gaussians with large variance (dense-large-variance (DLV)) leads to prioritizing the learning of low-frequency components, compared to dense-small-variance (DSV) initialization, observed in the coarse signal in the bottom row of step 10.

Although DLV initialization succeeds in learning the low-frequency components of the true distribution first, due to the dense number of Gaussians, DLV initialization fails to converge with large fluctuations. This can be observed by comparing the learned signals from steps 10, 100, and 1000. The learned signal is already quite close to the true distribution in 10 steps but fails to converge even after a sufficient number of steps.

To verify that our findings also apply to the initialization of 3D Gaussians for 3DGS, we implement the DLV initialization and evaluate the performance of 3DGS (DLV) on the Mip-NeRF360 dataset. For this experiment, we set the initial number of Gaussians as  $N = 1,000,000$ , equal to our setting for DSV initialization. As the initial covariance is defined by the mean distance of the three nearest neighbors, we manually scale up the initial covariance to guide the Gaussians to learn from wider areas. The results are shown in Table 6, where the performance of DLV random initialization is between DSV and SLV random initialization, aligned with the results of our toy experiment. Note that 3DGS (SLV) in Table 6 indicates 3DGS trained using only the sparse-large-variance (SLV) random initialization of our strategy.

Table 6: **Quantitative comparison on Mip-NeRF360 dataset with various initialization methods.** We compare our method with the DSV random initialization method described in the original 3DGS [1], DLV random initialization, and SLV random initialization method. We report PSNR, SSIM, LPIPS.

Method	Outdoor Scene											
	bicycle			flowers			garden			stump		
PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	
3DGS (DSV)	21.034	0.575	0.378	17.815	0.469	0.403	23.217	0.783	0.175	20.745	0.618	0.345
3DGS (DLV)	21.595	0.552	0.426	18.959	0.469	0.445	23.338	0.757	0.218	19.845	0.555	0.427
<b>3DGS (SLV)</b>	<b>23.227</b>	<b>0.610</b>	<b>0.373</b>	<b>20.498</b>	<b>0.528</b>	<b>0.392</b>	<b>25.670</b>	<b>0.816</b>	<b>0.157</b>	<b>23.748</b>	<b>0.647</b>	<b>0.339</b>

Method	Indoor Scene									Average		
	room			counter			kitchen					
PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	
3DGS (DSV)	29.685	<b>0.894</b>	<b>0.265</b>	23.608	0.833	0.276	26.078	0.893	0.161	18.538	0.719	0.401
3DGS (DLV)	29.284	0.886	0.280	26.253	0.857	0.272	27.975	0.900	0.161	22.363	0.793	0.360
<b>3DGS (SLV)</b>	<b>29.966</b>	0.892	0.268	<b>27.473</b>	<b>0.868</b>	<b>0.260</b>	<b>29.934</b>	<b>0.915</b>	<b>0.159</b>	<b>29.132</b>	<b>0.900</b>	<b>0.251</b>

## D Additional analysis

### D.1 Analysis on prioritized learning of low-frequency components

In Section 4 of our main paper, we find that SfM initialization guides 3D Gaussians with a coarse approximation of the true distribution to robustly learn the remaining high-frequency components. In Section 5.1 of our main paper, we then show that sparse-large-variance (SLV) random initialization successfully guides the Gaussians to prioritize the learning of low-frequency components. To verify that our strategy successfully guides 3DGS to prioritize the learning of low-frequency components, we provide additional analysis of our strategy through the rendered images achieved from different steps of training.



Figure 8: **Visualization of our strategy prioritizing the learning of coarse approximation.** This figure visualizes the rendering result of the ‘bicycle’ scene from the Mip-NeRF360 dataset trained with 3DGS using different initialization methods. We show the images rendered from different steps of 100, 3,000, 5,000, and 30,000. The results of 3DGS trained from SfM initialization, DSV random initialization, and Ours are shown from top to bottom.

As shown in Figure 8, we compare the rendered results of 3DGS trained with different initialization. As SfM provides a coarse approximation of the true distribution, 3DGS trained with SfM initialization (first row) directly learns the remaining high-frequency details. This tendency to directly learn the high-frequency components is highlighted in the second row where 3DGS is trained with DSV random initialization. Without any strategy to prioritize the learning of low-frequency components, the high-frequency components such as the edges of the bench and bicycle are already learned as shown in the rendered image of step 3,000. In contrast, our strategy (last row) successfully guides the

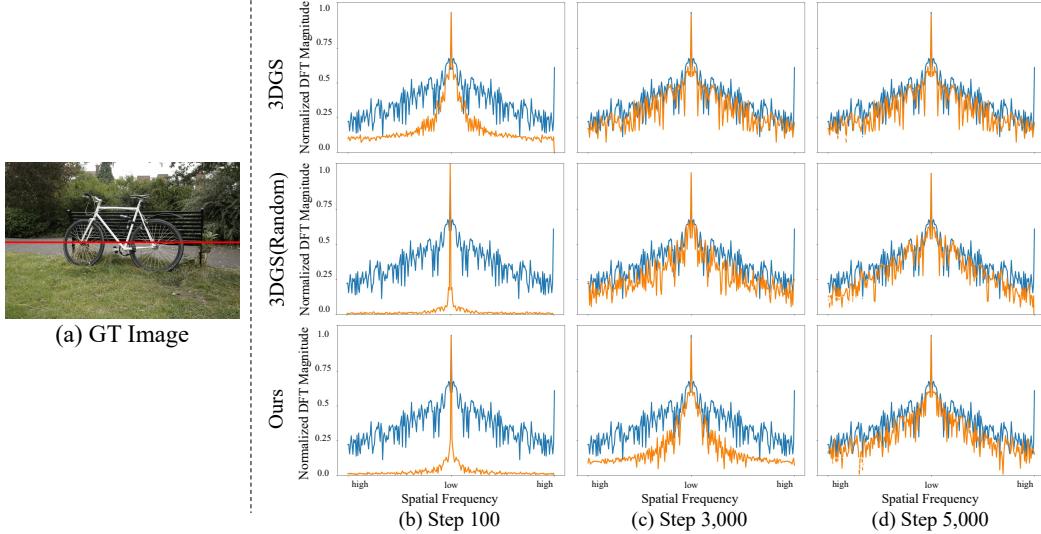


Figure 9: **Images from Figure 8 analyzed in the frequency domain.**

Gaussians to first model the scene’s coarse approximation. At step 3,000, high-frequency details are absent, resulting in smoothed appearances for the grass, road, and background trees.

In Figure 9, we further analyze the rendered images of Figure 8 in the frequency domain. We sample a horizontal line from the images as in (a) and perform FFT [38]. From (b) to (d), the normalized DFT magnitude of the transformed signal from the GT image and the rendered image is shown in sequential timesteps of 100, 3,000, and 5,000.

As mentioned in Section 4 of our main paper, 3DGs (top row) starts from a coarse approximation of the signal, which can be observed by the low-frequency components already being captured in step 100 ((b), top row). This is more evident when compared to the transformed signal of 3DGS (DSV) and Ours at step 100 ((b), middle and bottom row), where the low-frequency components are largely different from the GT signal. Aligned with our analysis of the toy experiment in Section 5.1 of our main paper, DSV random initialization exhibits a tendency towards over-fast convergence on high-frequencies. This can be observed by comparing (b) and (c), where the high-frequency components are quickly learned in (c). This leads to high-frequency artifacts after optimization, as shown in the rendered image (step 30,000, middle row) in Figure 8.

On the other hand, our strategy successfully guides the Gaussians to prioritize the learning of low-frequency components. When comparing (b), (c), and (d) of the last row, the low-frequency components are learned in (c) and then the high-frequency details are captured in (d). Note that this prioritization of learning low-frequency components makes the overall training process similar to starting with SfM initialization. The similarity is further highlighted by comparing the transformed signals of 3DGs in (b) and Ours in (c).

## D.2 Ablation study on warm up phase

**Ablation settings for low-pass filter control.** To demonstrate the effectiveness of our progressive Gaussian low-pass filter control strategy, we employ three different decreasing functions of convex, linear, and concave to control the Gaussian low-pass filter value  $s$ . Different from our strategy, where the value  $s$  is defined adaptively by image height, width, and the number of Gaussians  $N$  at each time step, the remaining functions are manually defined to achieve  $s = 300$  at step 0 and  $s = 0.3$  at about 3,000 steps across all scenes. The intuition behind this design is based on our analysis that our adaptive Gaussian low-pass filter value reaches 0.3 between 2,000-3,000 steps. Also, we empirically find that the initial Gaussian low-pass filter value  $s > 300$  offers no significant improvement, only making the overall computation inefficient. Based on these findings, we define the max value of the Gaussian low-pass filter as  $s = 300$ .

For the convex function, we use the following formula for  $s$  scheduling:

$$s = \max(7^{-\frac{x}{1000}} * 300, 0.3). \quad (12)$$

For the linear function, we use the following formula for  $s$  scheduling:

$$s = \max(300 - 0.0997084x, 0.3). \quad (13)$$

For the concave function, we use the following formula for  $s$  scheduling:

$$s = \max(300 * (1 + 7^{-3} - 7^{\frac{x-3000}{1000}}), 0.3). \quad (14)$$

The illustration of different Gaussian low-pass value formulas is shown in Figure 10 and Figure 11 where our formula is adaptively defined, showing different functions for each scene.

**Comparison.** In Table 10, we show a more detailed ablation study on the components belonging to the warm up phase. Additional experiments regarding (1) fixing learning rate during the warm up phase (2) the other progressive low-pass filter control functions mentioned above are shown here. Results show that our choice of dynamically adjusting the low-pass filter value based on the number of Gaussians achieves the best performance compared to the others. Also as can be seen in the Table, each of fixing learning rate, SLV initialization and ABE-Split solely contributes to the performance improvement, verifying the effect of our proposed methods. Compared to the original DSV random initialization, we achieve an significant average PSNR improvement over 5dB.

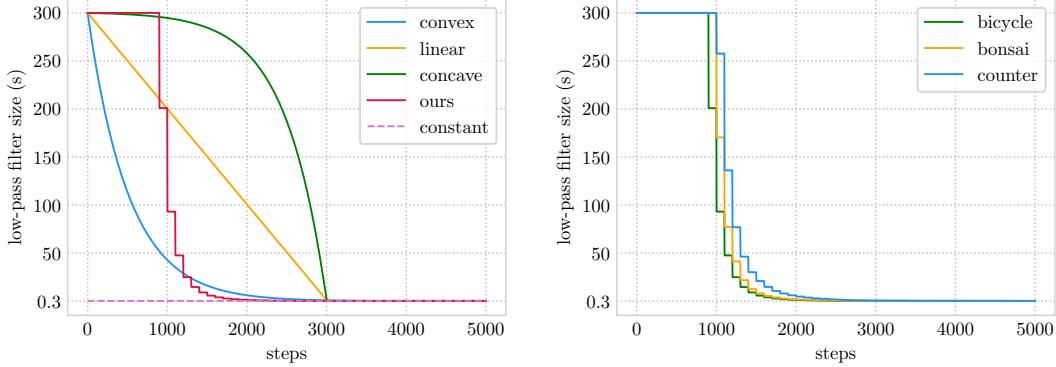


Figure 10: Illustration of different Gaussian low-pass filter value formulas.

Figure 11: Illustration of our Gaussian low-pass filter value formula.

Table 7: Ablation study on warm up phase.

Low-pass filter	Init.	Fixed learning rate	ABE-Split	Mip-NeRF360		
				PSNR↑	SSIM↑	LPIPS↓
Constant	DSV	✗	✗	22.190	0.704	0.313
Constant	SLV	✗	✗	25.675	0.749	0.288
Progressive (Convex)	SLV	✗	✗	25.799	0.756	0.286
Progressive (Linear)	SLV	✗	✗	25.898	0.754	0.290
Progressive (Concave)	SLV	✗	✗	25.974	0.754	0.288
Ours	SLV	✗	✗	26.180	0.761	0.280
Ours	SLV	✓	✗	26.362	0.791	0.241
<b>Ours</b>	<b>SLV</b>	<b>✓</b>	<b>✓</b>	<b>27.228</b>	<b>0.807</b>	<b>0.229</b>

### D.3 Ablation on sparse SfM initialization

In Table 8, we compare two methods for creating sparse SfM points to reduce the uncertainty of the SfM point cloud and use our coarse-to-fine method with the SfM point cloud: cluster and <10%. For

Table 8: **Quantitative comparison on Mip-NeRF360 dataset in sparse SfM points setting.** We compare our method with 3DGS in sparse SfM points setting, We report PSNR, SSIM and LPIPS.

Method	SfM points	Outdoor Scene														
		bicycle			flowers			garden			stump			treehill		
		PSNR↑	SSIM↑	LPIPS↓												
3DGS	cluster	24.542	0.703	0.289	21.044	0.569	0.365	26.883	0.852	0.118	25.774	0.735	0.250	22.102	0.601	0.376
3DGS	<10%	24.456	0.700	0.294	21.260	0.580	0.360	26.832	0.850	0.127	26.165	0.744	0.247	22.260	0.612	0.365
3DGS	full	<b>25.246</b>	<b>0.771</b>	<b>0.205</b>	21.520	0.605	0.336	<b>27.410</b>	<b>0.868</b>	<b>0.103</b>	26.550	<b>0.775</b>	0.210	22.490	<b>0.638</b>	<b>0.317</b>
Ours	cluster	25.211	0.767	0.211	<b>21.704</b>	0.617	0.321	27.166	0.862	<b>0.103</b>	<b>26.816</b>	<b>0.775</b>	<b>0.204</b>	<b>22.589</b>	0.630	0.328
Ours	<10%	25.214	0.769	0.208	<b>21.858</b>	<b>0.619</b>	<b>0.320</b>	27.130	0.861	0.107	<b>26.899</b>	<b>0.777</b>	<b>0.202</b>	<b>22.618</b>	0.632	0.327
Method	SfM points	Indoor Scene												Average		
		PSNR↑	SSIM↑	LPIPS↓												
3DGS	cluster	29.919	0.899	0.255	28.256	0.893	0.227	30.127	0.917	0.140	31.247	0.932	0.220	26.655	0.789	0.249
3DGS	<10%	30.795	0.908	0.243	28.523	0.896	0.223	30.120	0.915	0.140	31.625	0.934	0.221	26.893	0.793	0.247
3DGS	full	<b>30.632</b>	<b>0.914</b>	<b>0.220</b>	28.700	<b>0.905</b>	<b>0.204</b>	30.317	<b>0.922</b>	<b>0.129</b>	<b>31.980</b>	<b>0.938</b>	<b>0.205</b>	27.205	<b>0.815</b>	<b>0.214</b>
Ours	cluster	30.363	0.909	0.233	28.594	0.901	0.211	<b>31.136</b>	<b>0.923</b>	0.133	31.695	<b>0.935</b>	0.217	<b>27.253</b>	<b>0.813</b>	0.218
Ours	<10%	<b>30.843</b>	<b>0.912</b>	<b>0.230</b>	<b>28.703</b>	<b>0.902</b>	<b>0.209</b>	<b>31.364</b>	<b>0.922</b>	0.133	<b>31.829</b>	<b>0.938</b>	<b>0.214</b>	<b>27.384</b>	<b>0.815</b>	<b>0.217</b>

the cluster, we use HDBSCAN [46] to cluster the points based on the location of the SfM point cloud, and use the centroids of the clusters for the sparse SfM initialization. We conduct experiments using the default hyperparameters in scikit-learn [47] for HDBSCAN [46]. For <10%, we create sparse SfM points by using only the top 10% of points utilizing the reprojection error value that comes out with the SfM output. We confirm that our method achieves additional performance improvements by using the SfM points together.

Table 9: **Quantitative comparison on Mip-NeRF360 dataset in noisy initial SfM point cloud settings.** We compare 3DGS method with different noisy initial SfM point cloud. We report PSNR, SSIM, LPIPS.

Initialization	Outdoor Scene														
	bicycle			flowers			garden			stump			treehill		
	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓
SfM	25.246	0.771	0.205	21.520	0.605	0.336	27.410	0.868	0.103	26.550	0.775	0.210	22.490	0.638	0.317
SfM + $\epsilon$	24.289	0.692	0.307	21.028	0.565	0.374	26.515	0.846	0.133	26.028	0.743	0.252	21.728	0.607	0.380
SfM+constant	23.619	0.625	0.358	21.139	0.569	0.364	25.663	0.809	0.163	23.382	0.641	0.335	21.989	0.593	0.380
Initialization	Indoor Scene												Average		
	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓
SfM	30.632	0.914	0.220	28.700	0.905	0.204	30.317	0.922	0.129	31.980	0.938	0.205	27.205	0.815	0.214
SfM + $\epsilon$	30.740	0.901	0.260	23.872	0.838	0.285	24.477	0.877	0.175	25.823	0.875	0.265	24.944	0.771	0.270
SfM+constant	30.790	0.899	0.264	28.041	0.875	0.250	30.203	0.914	0.145	31.006	0.924	0.230	26.204	0.761	0.277

#### D.4 Analysis on the impact of noise on the initial SfM point cloud

In Table 1, we present detailed results to further investigate the ability of the 3DGS optimization scheme to transport Gaussians to the correct 3DGS locations on Mip-NeRF360 datasets. Here, we conduct this experiments by adding random noise  $\epsilon \sim \mathcal{N}(0, 1)$  and constant systematic noise, whose value equals 2, to the initial SfM points. The results shown in Table 9 prove that 3DGS strongly depends on the initial point. Figure 12 shows initial SfM points and points with noise  $\epsilon$ . Even with the small amount of noise, 3DGS fails to move to the correct position.

#### D.5 Analysis on movement of each Gaussian

To observe the movement of the each Gaussian, we measure how far the Gaussian moves from its initial position during the training. An additional parameter is incorporated to record the initial position, ensuring that even when Gaussians are split or cloned, the initial position parameters are retained. Then, the movement is calculated as difference between the final position of each Gaussian after training and its respective initial position.

We conduct analysis on “Truck” scene of Tanks&Temples dataset, comparing the settings of 3DGS with SfM point initialization, 3DGS with random initialization, and our method. The mean, standard deviation, and the top 1% values of the movements is shown in Table 2. Additionally, Figure 13 shows the overall scene from the same camera viewpoint for each experiment to observe the differences in distribution of overall Gaussians between the beginning and 30,000 step. In case such as 3DGS

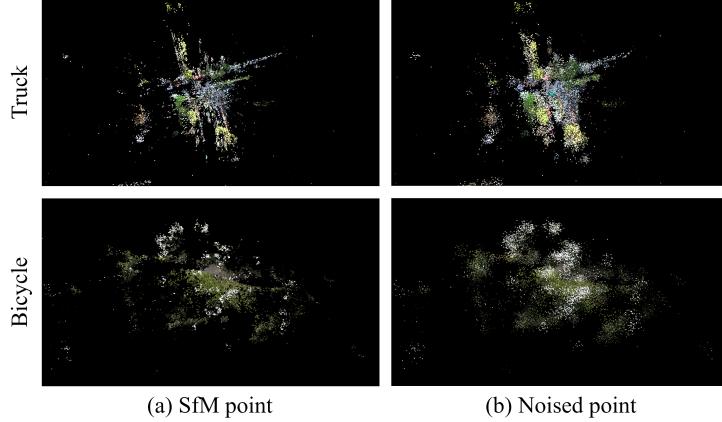


Figure 12: **Visualization of SfM points and points with noise  $\epsilon$ .**

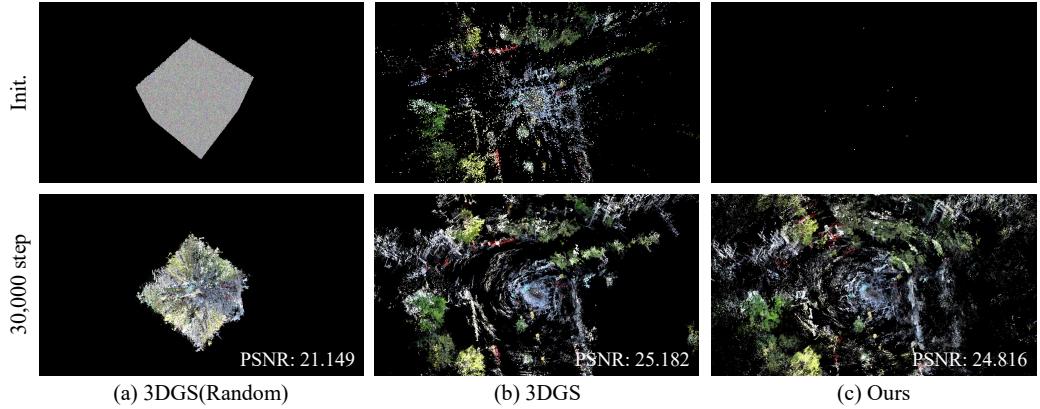


Figure 13: **Displacements of Gaussians from initial positions**

with SfM initialization and random initialization, the positions of the Gaussians does not change significantly. However, our method shows substantial changes in comparison.

#### D.6 Expanding Scene bound

As discussed in Section 4 and can be seen in supplementary materials Section D.5, Gaussians struggle to move further from the initialized locations. Therefore, in Table 10, we compare the results according to the initial scene bound and our method using the ABE-Split algorithm on the Mip-NeRF360 dataset to see if it is sufficient to initialize with a sufficiently wide scene bound. Extent  $\times 3$  and Extent  $\times 10$  are the initial scene bounds created by multiplying the camera extent given in the dataset by 3 and 10, respectively, and 3 is used in the original 3DGS. As can be seen in Extent  $\times 10$ , it is confirmed that simply starting with a wide initial scene bound does not yield sufficient performance.

#### D.7 Detailed results of Oracle scene bound experiments

To find out if the performance degradation of using noisy or random point cloud *solely* comes from losing the information of where the Gaussians should be located, we conduct a simple experiment of training 3DGS with random point cloud sampled from the oracle scene bounds. Specifically, here we assume the oracle scene bounds as the final scene bound defined by the final positions of the Gaussians, which are trained from point cloud achieved from SfM. Table 11 presents the results for each scene. The experiment reveals that SfM initialization provides additional benefits to 3DGS beyond merely placing the Gaussians in proper locations, as starting from random point cloud even

in oracle scene bounds leads to sub-optimal results compared to the one starting from SfM-initialized point cloud.

Table 10: **Quantitative comparison on Mip-NeRF360 dataset within different scene bound settings.** We compare our method (SLV) with 3DGS using random initialization within different scale bounding box. We report PSNR, SSIM, LPIPS.

Scene bound	Outdoor Scene														
	bicycle			flowers			garden			stump					
PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓				
Extent×3	21.034	0.575	0.378	17.815	0.469	0.403	23.217	0.783	0.175	20.745	0.618	0.345			
Extent×10	23.864	0.643	0.354	20.974	0.553	0.381	25.862	0.823	0.151	24.876	0.696	0.300			
Ours	<b>25.042</b>	<b>0.747</b>	<b>0.238</b>	<b>21.762</b>	<b>0.616</b>	<b>0.324</b>	<b>26.884</b>	<b>0.854</b>	<b>0.114</b>	<b>26.680</b>	<b>0.768</b>	<b>0.215</b>	<b>22.528</b>	<b>0.621</b>	<b>0.342</b>

Scene bound	Indoor Scene												Average				
	room			counter			kitchen			bonsai							
PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓
Extent×3	29.685	0.894	0.265	23.608	0.833	0.276	26.078	0.893	0.161	18.538	0.719	0.401	22.190	0.704	0.313		
Extent×10	29.834	0.889	0.276	27.441	0.863	0.267	30.424	0.913	0.147	30.482	0.917	0.242	26.203	0.765	0.279		
Ours	<b>30.809</b>	<b>0.906</b>	<b>0.247</b>	<b>28.529</b>	<b>0.895</b>	<b>0.223</b>	<b>31.270</b>	<b>0.920</b>	<b>0.137</b>	<b>31.547</b>	<b>0.934</b>	<b>0.218</b>	<b>27.228</b>	<b>0.807</b>	<b>0.229</b>		

Table 11: **Quantitative comparison on Mip-NeRF360 dataset in oracle scene bound settings.** We compare 3DGS method with random initialization in paper setting, SfM initialization and random initialization in oracle scene bound. We report PSNR, SSIM, LPIPS.

Initialization	Outdoor Scene											
	bicycle			flowers			garden			stump		
PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	
3DGS (Random)	21.034	0.575	0.378	17.815	0.469	0.403	23.217	0.783	0.175	20.745	0.618	0.345
3DGS (SfM)	25.246	0.771	0.205	21.520	0.605	0.336	27.410	0.868	0.103	26.550	0.775	0.210
3DGS (Oracle)	24.227	0.664	0.328	21.353	0.582	0.356	26.791	0.843	0.131	24.634	0.678	0.318

Initialization	Indoor Scene												Average				
	room			counter			kitchen			bonsai							
PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓
3DGS (Random)	29.685	0.894	0.265	23.608	0.833	0.276	26.078	0.893	0.161	18.538	0.719	0.401	22.190	0.704	0.313		
3DGS (SfM)	30.632	0.914	0.220	28.700	0.905	0.204	30.317	0.922	0.129	31.980	0.938	0.205	27.205	0.815	0.214		
3DGS (Oracle)	30.192	0.895	0.265	27.812	0.873	0.251	30.797	0.917	0.140	30.303	0.916	0.238	26.481	0.774	0.267		

## D.8 Comparison of computational resources

In this section, we now compare the computational resources of Plenoxels [35], InstantNGP [25], 3DGS [1], 3DGS with random initialization, and 3DGS with our strategy. We evaluate different methods on the Mip-NeRF360 [21], Tanks&Temples [44], and Deep Blending [45] dataset.

**Training time.** Note that when pre-defined camera poses are available, methods other than 3DGS [1] can bypass the time-consuming Structure-from-Motion (SfM) process. To highlight this overhead, in Table 12 we compare the training time of 3DGS with the SfM process taken into account. For 3DGS, we find that the number of Gaussians in each training step has a direct correlation to the overall training time - more Gaussians lead to longer training time.

Table 12: **Training time.**<sup>3</sup>

	Mip-NeRF360	Tanks&Temples	Deep Blending
Plenoxels	25m49s	25m5s	27m49s
INGP-Base	5m37s	5m26s	6m31s
INGP-Big	7m30s	6m59s	8m
3DGS <sup>4</sup>	41m33s + (19m9s) <sup>†</sup>	26m54s + (19m43s) <sup>†</sup>	36m2s + (6m23s) <sup>†</sup>
3DGS(Random)	28m22s	18m31s	29m33s
Ours	31m34s	15m13s	27m51s

<sup>3</sup>Except for the SfM processing time and training times of 3DGS(Random) and Ours, we copied the values from 3DGS [1]. Note that 3DGS utilized a single A6000 GPU and we utilized a single RTX 3090 GPU for evaluation.

<sup>4</sup>The processing time of SfM is denoted with a †.

## D.9 Limitations

Although **RAIN-GS** robustly guides 3D Gaussians to model the scene even from randomly initialized point cloud, our novel strategy is not without limitations. As our strategy prioritizes learning the coarse approximation, it sometimes fails to detect the need for further densification to capture high-frequency details. This happens especially for regions where the L1 rendering loss cannot distinguish between a coarse approximation and a high-frequency true distribution. An example of this limitation is shown in Figure 14, where the grass region in the front of the rendered image lacks high-frequency details compared to the ground truth image. However, we find that this limitation also appears when training 3DGS with SfM initialized point cloud, which is also shown to start optimization from a coarse approximation of the true distribution in our main paper. We posit that this limitation mainly comes from the lack of supervision, using the L1 rendering loss as the main signal. Although adding additional supervision from depth or error maps could open up new possibilities to resolve these limitations, we leave this direction as future work.



**Figure 14: Visualization of failure cases.** This figure shows the rendering results of the “flowers” scene of the Mip-NeRF360 dataset from Ours and 3DGS. When compared to the GT image shown in (c), the front of the images of (a) and (b) lack high-frequency details.

## E Additional qualitative results

We show additional qualitative results in Figure 15.

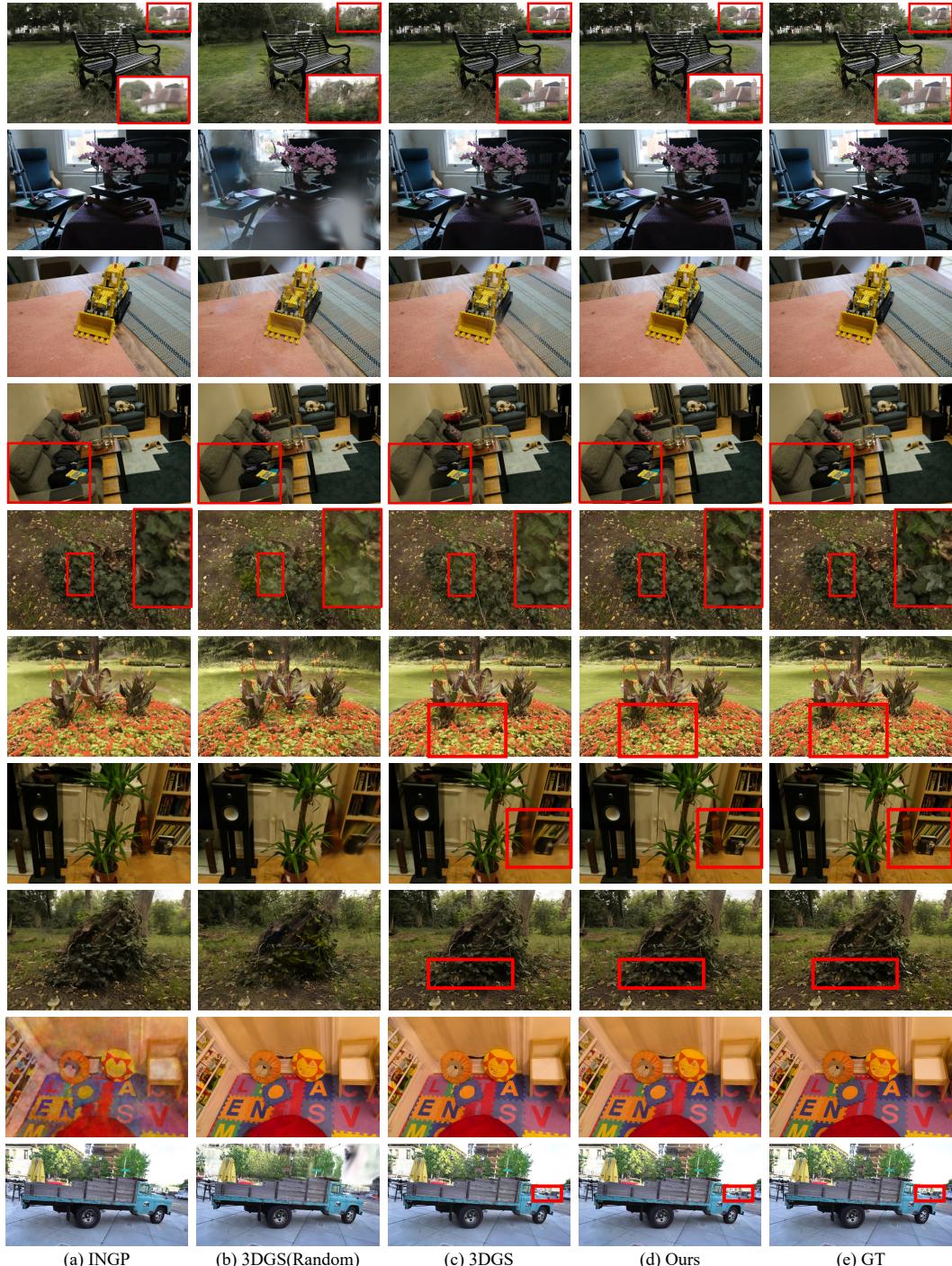
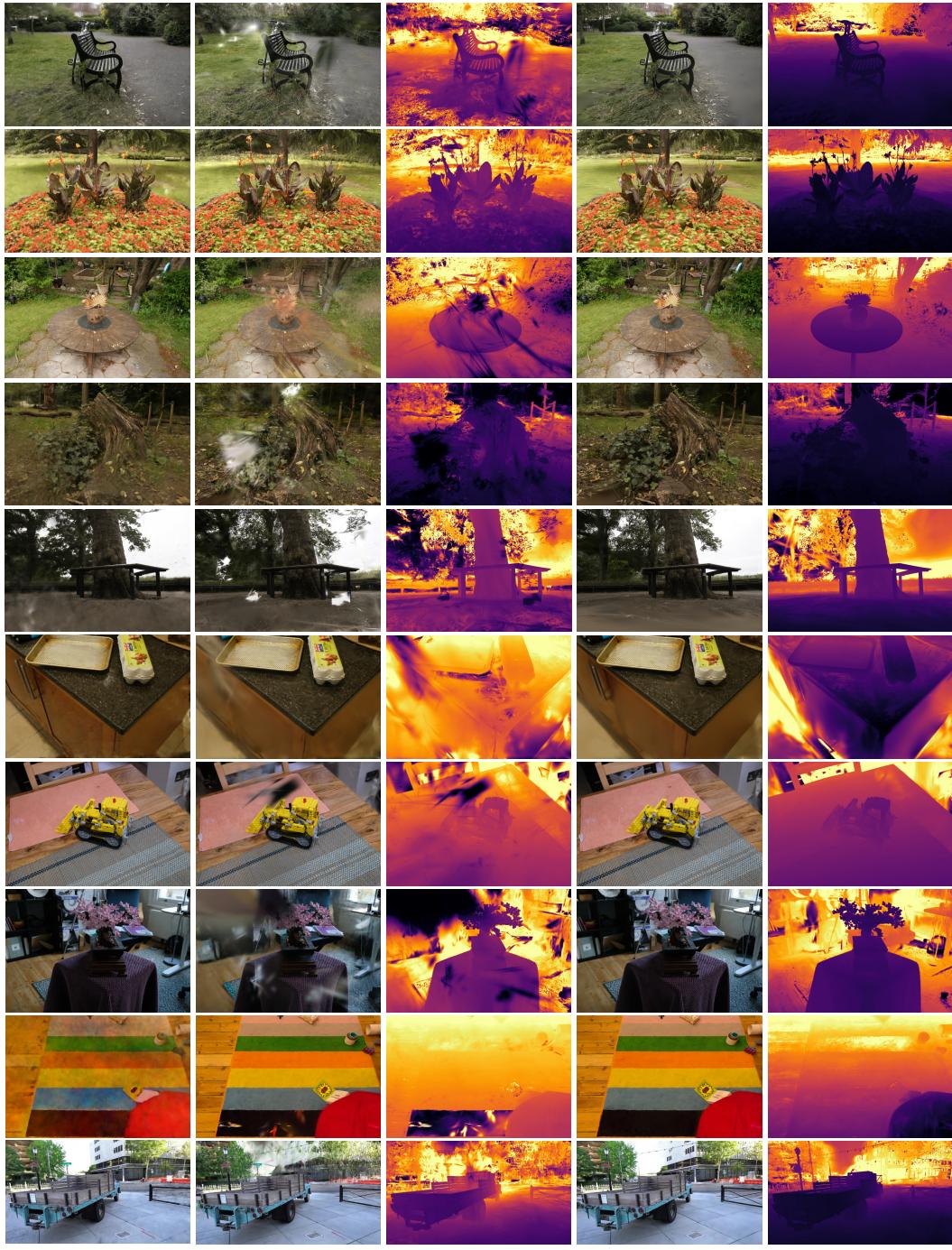


Figure 15: **Additional qualitative results.**



(a) INGP

(b) 3DGS (DSV)

Figure 16: Additional qualitative results with depth.

(c) Depth of (b)

(d) Ours

(e) Depth of (d)

## F Additional results of 3DGS in sparse view settings

Due to the impressive performance shown by NeRF, various follow-ups have been proposed, specifically to succeed in learning NeRF with a limited number of images [48, 27, 49, 26, 50]. However, due to the requirement of accurate pose and initial point cloud of 3DGS, the task of successfully guiding 3D Gaussians with a limited number of images becomes a much more complicated task. In this section, we evaluate our strategy in training 3DGS with few-shot images, where SfM struggles to converge. In Table 13 and Figure 17, we show quantitative and qualitative results comparing 3DGS(Random) and Ours in the sparse view setting in the Mip-NeRF360 dataset [21]. To show the effectiveness of our strategy when SfM struggles to converge, after excluding every 8th image for evaluation, we train the model on randomly sampled 10% of the remaining images. Ours outperforms 3DGS trained with random initialization both qualitatively and quantitatively, demonstrating the potential of our work to be extended to the task of training 3DGS with few-shot images. In Figure 17, it is shown that the rendered results of Ours much more robustly models the real geometry of the scene.

Table 13: **Quantitative comparison on Mip-NeRF360 dataset in sparse-view setting.** We compare our method with the random initialization method described in the original 3DGS [1]. We report PSNR, SSIM, LPIPS.

Method	Outdoor Scene												Average		
	bicycle			flowers			garden			stump					
PSNR↑	SSIM↑	LPIPS↓													
3DGS(Random)	11.787	<b>0.170</b>	0.637	10.948	0.118	<b>0.648</b>	16.752	0.377	0.444	<b>15.239</b>	<b>0.194</b>	<b>0.602</b>	12.675	<b>0.206</b>	0.599
Ours	<b>12.436</b>	0.157	<b>0.636</b>	<b>11.546</b>	<b>0.120</b>	0.649	<b>17.955</b>	<b>0.463</b>	<b>0.425</b>	14.675	0.142	0.614	<b>12.820</b>	0.191	<b>0.597</b>

Method	Indoor Scene												Average		
	room			counter			kitchen			bonsai					
PSNR↑	SSIM↑	LPIPS↓													
3DGS(Random)	18.864	0.702	<b>0.418</b>	15.311	0.575	0.475	18.181	0.629	0.384	13.246	0.508	0.538	14.778	0.386	0.527
Ours	<b>19.804</b>	<b>0.722</b>	0.424	<b>18.201</b>	<b>0.637</b>	<b>0.451</b>	<b>21.474</b>	<b>0.718</b>	<b>0.359</b>	<b>20.664</b>	<b>0.721</b>	<b>0.408</b>	<b>16.620</b>	<b>0.430</b>	<b>0.507</b>

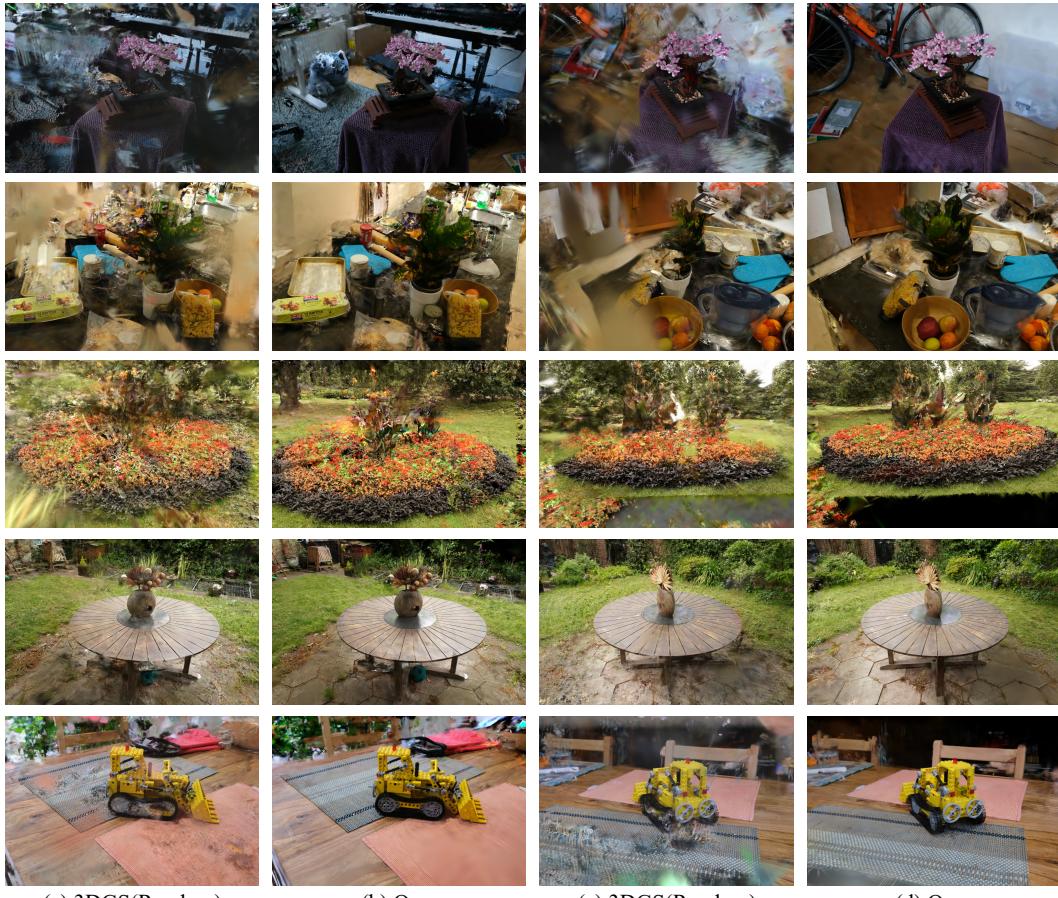


Figure 17: **Qualitative results in sparse view settings.**

## References

- [1] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics*, 42(4), 2023.
- [2] Lining Xu, Vasu Agrawal, William Laney, Tony Garcia, Aayush Bansal, Changil Kim, Samuel Rota Bulò, Lorenzo Porzi, Peter Kortschieder, Aljaž Božič, et al. Vr-nerf: High-fidelity virtualized walkable spaces. In *SIGGRAPH Asia 2023 Conference Papers*, pages 1–12, 2023.
- [3] Michał Adamkiewicz, Timothy Chen, Adam Caccavale, Rachel Gardner, Preston Culbertson, Jeannette Bohg, and Mac Schwager. Vision-only robot navigation in a neural radiance world. *IEEE Robotics and Automation Letters*, 7(2):4606–4613, 2022.
- [4] Yunhao Ge, Harkirat Behl, Jiashu Xu, Suriya Gunasekar, Neel Joshi, Yale Song, Xin Wang, Laurent Itti, and Vibhav Vineet. Neural-sim: Learning to generate training data with nerf. In *European Conference on Computer Vision*, pages 477–493. Springer, 2022.
- [5] 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. *Communications of the ACM*, 65(1):99–106, 2021.
- [6] Sara Fridovich-Keil, Giacomo Meanti, Frederik Rahbæk Warburg, Benjamin Recht, and Angjoo Kanazawa. K-planes: Explicit radiance fields in space, time, and appearance. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12479–12488, 2023.
- [7] Lingjie Liu, Jiatao Gu, Kyaw Zaw Lin, Tat-Seng Chua, and Christian Theobalt. Neural sparse voxel fields. *Advances in Neural Information Processing Systems*, 33:15651–15663, 2020.
- [8] Christian Reiser, Songyou Peng, Yiyi Liao, and Andreas Geiger. Kilonerf: Speeding up neural radiance fields with thousands of tiny mlps, 2021.
- [9] Matthew Tancik, Vincent Casser, Xincheng Yan, Sabeek Pradhan, Ben Mildenhall, Pratul P. Srinivasan, Jonathan T. Barron, and Henrik Kretzschmar. Block-nerf: Scalable large scene neural view synthesis, 2022.
- [10] Kai Zhang, Gernot Riegler, Noah Snavely, and Vladlen Koltun. Nerf++: Analyzing and improving neural radiance fields, 2020.
- [11] Georgios Kopanas, Julien Philip, Thomas Leimkühler, and George Drettakis. Point-based neural rendering with per-view optimization. In *Computer Graphics Forum*, volume 40, pages 29–43. Wiley Online Library, 2021.
- [12] Wenjing Bian, Zirui Wang, Kejie Li, Jia-Wang Bian, and Victor Adrian Prisacariu. Nope-nerf: Optimising neural radiance field with no pose prior. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4160–4169, 2023.
- [13] Jason Y Zhang, Deva Ramanan, and Shubham Tulsiani. Relpose: Predicting probabilistic relative rotation for single objects in the wild. In *European Conference on Computer Vision*, pages 592–611. Springer, 2022.
- [14] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *The International Journal of Robotics Research*, 32(11):1231–1237, 2013.
- [15] Jürgen Sturm, Nikolas Engelhard, Felix Endres, Wolfram Burgard, and Daniel Cremers. A benchmark for the evaluation of rgb-d slam systems. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pages 573–580. IEEE, 2012.
- [16] Jiaxiang Tang, Jiawei Ren, Hang Zhou, Ziwei Liu, and Gang Zeng. Dreamgaussian: Generative gaussian splatting for efficient 3d content creation. *arXiv preprint arXiv:2309.16653*, 2023.
- [17] Taoran Yi, Jiemin Fang, Junjie Wang, Guanjun Wu, Lingxi Xie, Xiaopeng Zhang, Wenyu Liu, Qi Tian, and Xinggang Wang. Gaussiandreamer: Fast generation from text to 3d gaussians by bridging 2d and 3d diffusion models. In *CVPR*, 2024.
- [18] Alex Nichol, Heewoo Jun, Prafulla Dhariwal, Pamela Mishkin, and Mark Chen. Point-e: A system for generating 3d point clouds from complex prompts. *arXiv preprint arXiv:2212.08751*, 2022.
- [19] Shuzhe Wang, Vincent Leroy, Yohann Cabon, Boris Chidlovskii, and Jerome Revaud. Dust3r: Geometric 3d vision made easy. *arXiv preprint arXiv:2312.14132*, 2023.

- [20] Jonathan T Barron, Ben Mildenhall, Matthew Tancik, Peter Hedman, Ricardo Martin-Brualla, and Pratul P Srinivasan. Mip-nerf: A multiscale representation for anti-aliasing neural radiance fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5855–5864, 2021.
- [21] Jonathan T Barron, Ben Mildenhall, Dor Verbin, Pratul P Srinivasan, and Peter Hedman. Mip-nerf 360: Unbounded anti-aliased neural radiance fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5470–5479, 2022.
- [22] Yilun Du, Cameron Smith, Ayush Tewari, and Vincent Sitzmann. Learning to render novel views from wide-baseline stereo pairs. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4970–4980, 2023.
- [23] Sunghwan Hong, Jaewoo Jung, Heeseong Shin, Jiaolong Yang, Seungryong Kim, and Chong Luo. Unifying correspondence, pose and nerf for pose-free novel view synthesis from stereo pairs. *arXiv preprint arXiv:2312.07246*, 2023.
- [24] Zhengqi Li, Qianqian Wang, Forrester Cole, Richard Tucker, and Noah Snavely. Dynibar: Neural dynamic image-based rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4273–4284, 2023.
- [25] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Transactions on Graphics (ToG)*, 41(4):1–15, 2022.
- [26] Jiuhn Song, Seonghoon Park, Honggyu An, Seokju Cho, Min-Seop Kwak, Sungjin Cho, and Seungryong Kim. Därf: Boosting radiance fields from sparse input views with monocular depth adaptation. *Advances in Neural Information Processing Systems*, 36, 2024.
- [27] Jiawei Yang, Marco Pavone, and Yue Wang. Freenerf: Improving few-shot neural rendering with free frequency regularization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8254–8263, 2023.
- [28] Jonathon Luiten, Georgios Kopanas, Bastian Leibe, and Deva Ramanan. Dynamic 3d gaussians: Tracking by persistent dynamic view synthesis. In *3DV*, 2024.
- [29] Ziyi Yang, Xinyu Gao, Wen Zhou, Shaohui Jiao, Yuqing Zhang, and Xiaogang Jin. Deformable 3d gaussians for high-fidelity monocular dynamic scene reconstruction. *arXiv preprint arXiv:2309.13101*, 2023.
- [30] Sameer Agarwal, Yasutaka Furukawa, Noah Snavely, Ian Simon, Brian Curless, Steven M Seitz, and Richard Szeliski. Building rome in a day. *Communications of the ACM*, 54(10):105–112, 2011.
- [31] Jan-Michael Frahm, Pierre Fite-Georgel, David Gallup, Tim Johnson, Rahul Raguram, Changchang Wu, Yi-Hung Jen, Enrique Dunn, Brian Clipp, Svetlana Lazebnik, et al. Building rome on a cloudless day. In *Computer Vision–ECCV 2010: 11th European Conference on Computer Vision, Heraklion, Crete, Greece, September 5–11, 2010, Proceedings, Part IV 11*, pages 368–381. Springer, 2010.
- [32] Johannes L Schonberger and Jan-Michael Frahm. Structure-from-motion revisited. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4104–4113, 2016.
- [33] Changchang Wu. Towards linear-time incremental structure from motion. In *2013 International Conference on 3D Vision-3DV 2013*, pages 127–134. IEEE, 2013.
- [34] Matthias Zwicker, Hanspeter Pfister, Jeroen Van Baar, and Markus Gross. Ewa splatting. *IEEE Transactions on Visualization and Computer Graphics*, 8(3):223–238, 2002.
- [35] Alex Yu, Sara Fridovich-Keil, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. Plenoxels: Radiance fields without neural networks, 2021.
- [36] David Charatan, Sizhe Li, Andrea Tagliasacchi, and Vincent Sitzmann. pixelsplat: 3d gaussian splats from image pairs for scalable generalizable 3d reconstruction. In *arXiv*, 2023.
- [37] Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the royal statistical society: series B (methodological)*, 39(1):1–22, 1977.
- [38] Henri J Nussbaumer and Henri J Nussbaumer. *The fast Fourier transform*. Springer, 1982.
- [39] Chen-Hsuan Lin, Wei-Chiu Ma, Antonio Torralba, and Simon Lucey. Barf: Bundle-adjusting neural radiance fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5741–5751, 2021.

- [40] Keunhong Park, Utkarsh Sinha, Jonathan T Barron, Sofien Bouaziz, Dan B Goldman, Steven M Seitz, and Ricardo Martin-Brualla. Nerfies: Deformable neural radiance fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5865–5874, 2021.
- [41] Benjamin Eckart, Kihwan Kim, Alejandro Troccoli, Alonzo Kelly, and Jan Kautz. Accelerated generative models for 3d point cloud data. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5497–5505, 2016.
- [42] Amir Hertz, Rana Hanocka, Raja Giryes, and Daniel Cohen-Or. Pointgmm: A neural gmm network for point clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12054–12063, 2020.
- [43] Zehao Yu, Anpei Chen, Binbin Huang, Torsten Sattler, and Andreas Geiger. Mip-splatting: Alias-free 3d gaussian splatting. *arXiv preprint arXiv:2311.16493*, 2023.
- [44] Arno Knapitsch, Jaesik Park, Qian-Yi Zhou, and Vladlen Koltun. Tanks and temples: Benchmarking large-scale scene reconstruction. *ACM Transactions on Graphics (ToG)*, 36(4):1–13, 2017.
- [45] Peter Hedman, Julien Philip, True Price, Jan-Michael Frahm, George Drettakis, and Gabriel Brostow. Deep blending for free-viewpoint image-based rendering. *ACM Transactions on Graphics (ToG)*, 37(6):1–15, 2018.
- [46] Leland McInnes and John Healy. Accelerated hierarchical density based clustering. In *2017 IEEE international conference on data mining workshops (ICDMW)*, pages 33–42. IEEE, 2017.
- [47] Oliver Kramer and Oliver Kramer. Scikit-learn. *Machine learning for evolution strategies*, pages 45–53, 2016.
- [48] Michael Niemeyer, Jonathan T. Barron, Ben Mildenhall, Mehdi S. M. Sajjadi, Andreas Geiger, and Noha Radwan. Regnerf: Regularizing neural radiance fields for view synthesis from sparse inputs, 2021.
- [49] Jaewoo Jung, Jisang Han, Jiwon Kang, Seongchan Kim, Min-Seop Kwak, and Seungryong Kim. Self-evolving neural radiance fields, 2023.
- [50] Min seop Kwak, Jiuhn Song, and Seungryong Kim. Geconerf: Few-shot neural radiance fields via geometric consistency, 2023.