Relaxing Accurate Initialization Constraint for 3D Gaussian Splatting

Jaewoo Jung*, Jisang Han*, Honggyu An*,
Jiwon Kang*, Seonghoon Park*, and Seungryong Kim

Korea University, Seoul, Korea
https://ku-cvlab.github.io/RAIN-GS

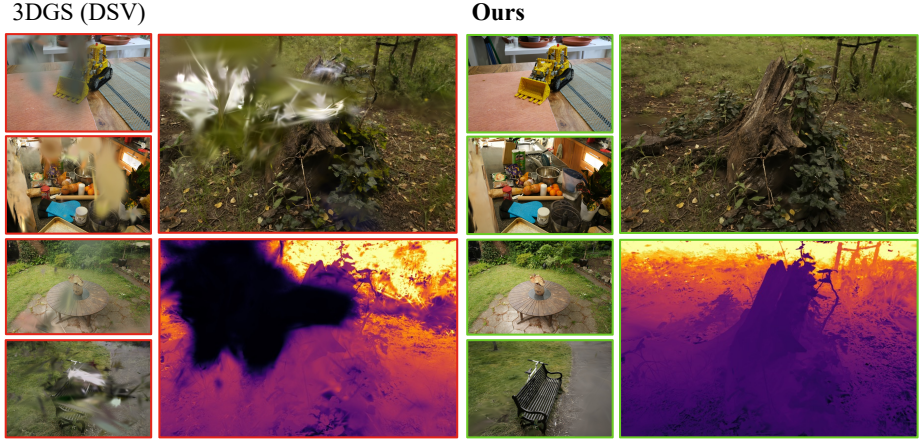


Fig. 1: Effectiveness of our simple strategy. Left and right show the results from 3DGS [13] trained with dense-small-variance (DSV) random initialization (random initialization method used in the original 3DGS) and 3DGS trained with ours, respectively. Transition from 3DGS to ours simply requires sparse-large-variance (SLV) random initialization and progressive Gaussian low-pass filtering. Remarkably, each of our strategies can be implemented with a simple change in ***one line of code***.

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 trained with randomly initialized point clouds, 3DGS often fails to maintain its ability to produce high-quality images, undergoing large performance drops of 4-5 dB in PSNR in general. Through extensive analysis of SfM initialization in the frequency domain and analysis of a 1D regression task with multiple 1D Gaussians, we propose a novel optimization strategy dubbed **RAIN-GS** (**R**elaxing **A**ccurate **IN**itialization Constraint for 3D **G**aussian **S**platting) that successfully trains 3D Gaussians from randomly initialized point clouds. We show the effectiveness of our strategy through quantitative and qualitative comparisons on standard datasets, largely improving the performance in all settings.

*: 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 [35], robotics [1], and data generation [8]. Neural radiance fields (NeRFs) [21] have demonstrated remarkable success in this field, learning implicit representations that capture intricate 3D geometry and specular effects solely from images. However, NeRFs' reliance on multi-layer perceptrons (MLPs) [7, 19, 27, 31, 41] results in slow and computationally intensive volume rendering, hindering real-time applications.

Recently, 3D Gaussian splatting (3DGS) [13] has emerged as a compelling alternative for both high-quality results and real-time rendering. Unlike NeRFs' implicit representations, 3DGS models the scene using explicit 3D Gaussians. In addition, an efficient CUDA-based differentiable tile rasterization [13, 15] technique enables the rendering of the learned 3D Gaussians in real-time.

Despite its remarkable results, 3DGS exhibits a significant performance drop when trained with randomly initialized point clouds instead of those achieved from Structure-from-Motion (SfM) [13]. This limitation becomes particularly pronounced in scenarios where SfM techniques struggle to converge, such as scenes with symmetry, specular properties, and textureless regions, and limited available views. Also, due to the dependency of 3DGS on initial point clouds, SfM becomes a hard prerequisite even for situations where camera poses can be obtained through external sensors or pre-calibrated cameras [9, 30].

In this work, we start with a natural question: *"Why is the initial point cloud so important in 3D Gaussian splatting?"* and analyze the difference between SfM and randomly-initialized point clouds. First of all, we analyze the signals in the frequency domain on rendered images and find that SfM initialization can be interpreted as starting with a *coarse* approximation of the true distribution. In 3DGS, this coarse approximation acts as a foundation for subsequent refinement throughout the optimization process, preventing Gaussians from falling into local minima.

Stemming from this analysis, we conduct a toy experiment of a simplified 1D regression task, to identify the essential elements to guide the Gaussians to robustly learn the ground truth distribution from scratch. This experiment reveals that initially learning the coarse approximation (low-frequency components) of the true distribution is crucial for successful reconstruction. Similar to the SfM initialization, we find the initially learned coarse approximation acts as a guidance when learning the high-frequency components of the distribution.

Expanding upon this observation, we propose a novel optimization strategy, dubbed **RAIN-GS** (**R**elaxing **A**ccurate **IN**itialization Constraint for 3D **G**aussian **S**platting), which combines **1)** a novel initialization method of starting with sparse Gaussians with large variance and **2)** progressive Gaussian low-pass filtering in the rendering process. Aligned with our observation, we demonstrate that our strategy successfully guides 3D Gaussians to learn the coarse distribution first and robustly learn the remaining high-frequency components later.

We show that this simple yet effective strategy largely improves the results when starting from randomly initialized point clouds without any regularization, training, or external models, effectively relaxing the requirement of accurately initialized point clouds from SfM.

2 Related work

Novel view synthesis. Synthesizing images from novel viewpoints of a 3D scene given images captured from multiple viewpoints is one of the essential but challenging tasks in computer vision. Neural radiance fields (NeRF) [21] have succeeded in significantly boosting the performance of this task 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. Due to the impressive ability to understand and model complex 3D scenes, various follow-ups [3–5, 11, 17, 22, 29, 36] adopted NeRF as their baseline model and further extend the ability of NeRF to model unbounded or dynamic scenes [3, 4, 17], lower the required number of images for successful training [29, 36], learn a generalizable prior to resolve the need of per-scene optimization [5, 11], or utilize an external hash-grid to fasten the overall optimization process [22]. 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) [13] has emerged as a promising alternative, 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 [20, 33, 37, 38], extended to model dynamic scenes [20, 37], or used as an alternative for text-to-3D tasks [33, 38] for fast generation. Nevertheless, the essential requirement of both accurate pose and initial point clouds limits the application of 3DGS to only scenes where SfM is applicable. 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.

Structure-from-Motion (SfM). SfM techniques [2, 6, 28] 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 [21] utilizes the estimated pose from the SfM and trains an implicit representation to model the scene. The recently proposed 3DGS [13] 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 this initialization is not accessible.

Despite the effectiveness of SfM in generating accurately aligned point clouds 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 [34]. 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.

3 Preliminary: 3D Gaussian splatting (3DGS)

Recently, 3DGS [13] has emerged as a promising alternative for the novel view synthesis task, modeling the scene with multiple 3D Gaussians [42]. Each i-th Gaussians G_i represents the scene with the following attributes: a position vector $\mu_i \in \mathbb{R}^3$, an anisotropic covariance matrix $\Sigma_i \in \mathbb{R}^{3\times3}$, spherical harmonic (SH) coefficients [22,39], and an opacity logit value $\alpha_i \in [0, 1)$. With these attributes, each Gaussian G_i is defined in the world space [42] as follows:

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

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

$$\Sigma_i' = JW\Sigma_i W^T J^T, \tag{2}$$

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 Σ_i' [42]. Finally, to render the color $C(p)$ of the pixel p, 3DGS utilizes alpha blending according to the Gaussians depth. For example, when N Gaussians are sorted by depth, the color $C(p)$ is calculated as follows:

$$C(p) = \sum_{i=1}^{N} c_i \alpha_i G_i'(p) \prod_{j=1}^{i-1}(1 - \alpha_j G_j'(p)), \tag{3}$$

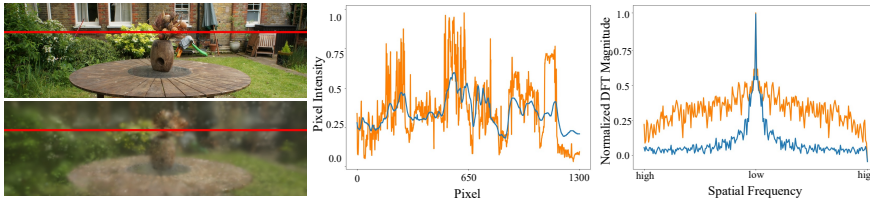
where c_i is the view-dependent color value of each Gaussian calculated with the SH coefficients, and G_i' is the 3D Gaussian projected to the 2D screen space.

Since 3DGS starts learning from sparse point clouds, they utilize an adaptive density control to obtain a more accurate and denser representation. By considering if the scene is under-/over-constructed, the Gaussian undergoes a clone/split operation to represent the scene with an adequate number of Gaussians.

4 Motivation

4.1 SfM initialization in 3DGS

To understand the large performance gap of 3DGS [13] depending on different initialization of point clouds, we first analyze its behavior when trained using an



(a) GT image and SfM pointcloud (b) 1D Signal (spatial) (c) 1D Signal (frequency)

Fig. 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. To analyze the images in the frequency domain, we randomly sample a horizontal line from the image marked in red. (b) The pixel intensity values along this line are shown, with the GT image indicated in orange and the rendered image in blue. (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.

SfM point cloud. As explained in Section 2, SfM provides a sparse point cloud with rough information of color and position. 3DGS effectively leverages this output, initializing the Gaussian parameters μ_i and spherical harmonic (SH) coefficients to the position of the point clouds and estimated color. This initialization's advantage is evident in Figure 2 (bottom row, (a)): after only 10 steps (0.03% of the total training steps), the rendered result already exhibits reasonable quality and similarity to the true image (top row, (a)).

To further investigate the benefits of SfM initialization, we analyze the rendered images in the frequency domain using Fourier transform [24]. In Figure 2, we randomly select a horizontal line from both the ground-truth and rendered images as shown in red in (a). The pixel intensity values along this line are visualized in (b), with the ground-truth image in orange and the rendered image in blue. These signals are then transformed into the frequency domain and presented in (c). This analysis in the frequency domain demonstrates that SfM initialization provides a coarse approximation of the underlying 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. To enable this, NeRF [21,32] utilizes positional encoding to facilitate learning of high-frequency components. However, the over-fast convergence of high-frequency components hinders NeRF from exploring low-frequency components, leading NeRF to overfit to high-frequency artifcats [36]. To circumvent this problem, prior works [18, 25, 36] adopt a frequency annealing strategy, guiding NeRF to sufficiently explore the low-frequency components first.

In this perspective, starting from SfM initialization can be understood as following a similar process, as it already provides the low-frequency components (Figure 2). Expanding upon this analysis, we propose that a similar strategy that can guide 3DGS to learn the low-frequency component first is essential to enable robust training when starting with randomly initialized point clouds.

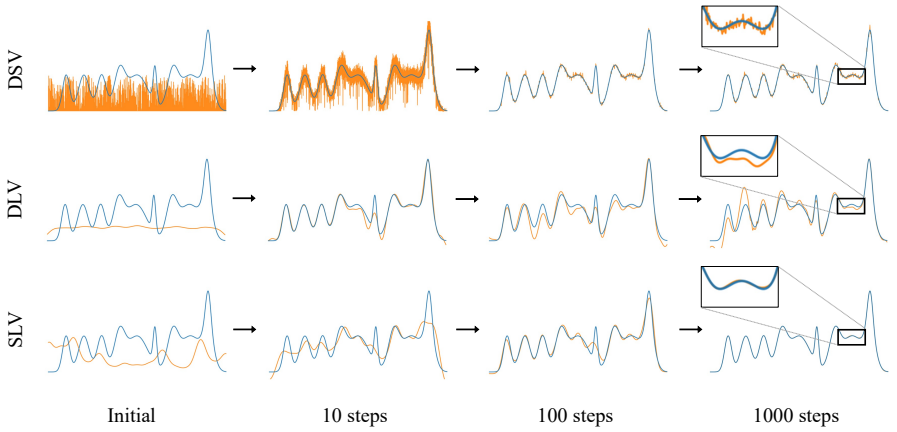


Fig. 3: Toy experiment to analyze different initialization methods. This figure visualizes the result of our toy experiment predicting the target distribution using a collection of 1D Gaussians, starting from different initialization methods. Dense-small-variance (DSV) and dense-large-variance (DLV) 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 Guassians 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.

4.2 Dense random initialization in 3DGS

For situations where initialized point clouds from SfM are unavailable, [13] proposes a dense-small-variance (DSV) random initialization method. They randomly sample dense point clouds within a cube three times the size of the camera's bounding box. As the initial covariance is defined as the mean distance to the three nearest neighboring points, this leads to initializing dense 3D Gaussians with small variances.

Building upon the importance of prioritizing low-frequency learning, we conduct a toy experiment in a simplified 1D regression task to examine how DSV initialization influences the optimization process. Using 1D Gaussians provides a controlled environment isolating the effects of initialization. Specifically, we define 1D Gaussians with its mean μ_i and variance σ_i^2 as learnable parameters to model a random 1D signal $Y(x)$ as the ground truth distribution. By blending the N Gaussians with a learnable weight w_i, we train the parameters with the L1 loss between the blended signal and $Y(x)$ as follows:

$$\mathcal{L} = \sum_x \left\| Y(x) - \sum_{i=0}^{N} w_i \cdot \exp\left(-\frac{(x-\mu_i)^2}{2\sigma_i^2} \right) \right\|, \tag{4}$$

where we set $N = 1,000$ for the DSV initialization setting.

As shown in the top row of Figure 3, 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 3) 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. More detailed settings and results of the toy experiment are provided in Section C in the supplementatry materials.

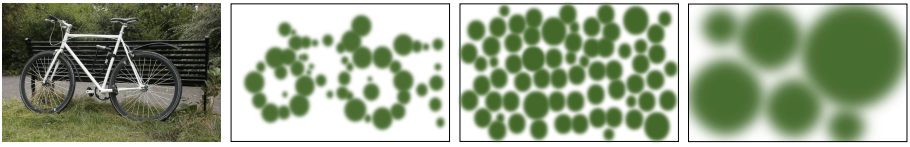
5 Methodology

Following our motivation (Section 4), we propose a novel optimization strategy, dubbed **RAIN-GS** (**R**elaxing **A**ccurate **IN**itialization Constraint for 3D **G**aussian **S**platting). This strategy guides 3D Gaussians towards prioritizing the learning of low-frequency components during the early phases of training, which enables the robust training of the remaining high-frequency components without falling into local minima. Our strategy consists of two key components: **1)** sparse-large-variance (SLV) initialization (Section 5.1) and **2)** progressive Gaussian low-pass filtering (Section 5.2).

5.1 Sparse-large-variance (SLV) initialization

As demonstrated in the 1D toy experiment (Section 4.2), optimization from a dense initialization fails to faithfully model the target distribution, where DLV produces undesired high-frequency artifacts and DSV fails to converge although succeeding in learning the low-frequency components first.

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 3, where the experiment is repeated with $N = 15$ Gaussians with s added to the initial variance. SLV initialization



(a) Image (b) SfM (c) Random (DSV) (d) Random (SLV)

Fig. 4: 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.

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.

When initializing N points, DSV initialization from [13] originally chooses $N > 100$K. Aligned with our analysis, lowering this value significantly improves performance by encouraging the initial focus on low-frequency components, producing fewer high-frequency artifacts. Surprisingly, this strategy becomes more effective even with extremely sparse initializations (e.g., as low as $N = 10$), verifying the effectiveness of our novel SLV initialization method. An illustration of different initialization methods is shown in Figure 4.

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, showing a tendency to collapse into similar problems with the DSV initialization. To regularize the 3D Gaussians to sufficiently explore the low-frequency components during the early stage of training, we propose a novel progressive control of the Gaussian low-pass filter 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:

$$G'_i(x) = e^{-\frac{1}{2}(x-\mu'_i)^T \Sigma'^{-1}_i (x-\mu'_i)}. \tag{5}$$

However, directly using projected 2D Gaussians can lead to visual artifacts when they become smaller than a pixel [13,40]. To ensure coverage of at least one pixel, 13 enlarge the 2D Gaussian's scale by adding a small value to the covariance's

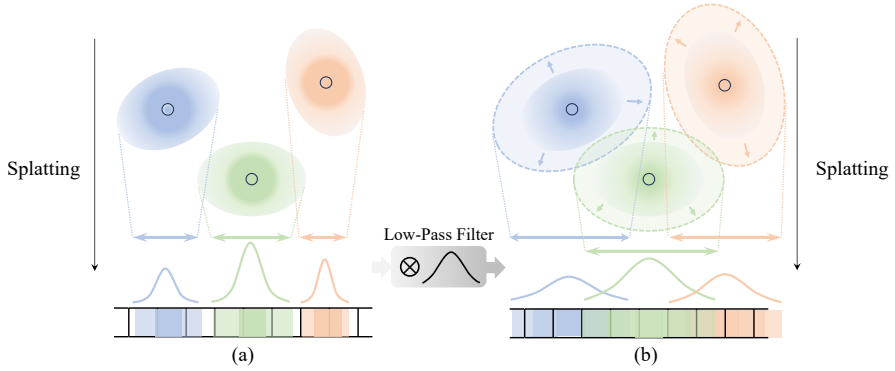


Fig. 5: Visualization of low-pass filter. This figure shows the visualization of the effect of the low-pass filter. As shown in (b), the convolution of the splatted 2D Gaussian with the low-pass filter expands the area the Gaussian is splatted onto, resulting in the Gaussians affecting larger areas than naïve splatting as shown in (a).

diagonal elements as follows:

$$G_i'(x) = e^{-\frac{1}{2}(x-\mu_i')^T(\Sigma_i'+sI)^{-1}(x-\mu_i')}, \tag{6}$$

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 [42]. After convolution with the low-pass filter, the area of the projected Gaussian G_i' is approximated by a circle with a radius defined by three times the larger eigenvalue of the 2D covariance matrix $(\Sigma_i' + sI)$.[1] Figure 5 illustrates the low-pass filter's effect, where the projected Gaussian is splatted to wider areas in (b) compared to (a).

Progressive Gaussian 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. Based on our analysis in Section 4.2 that learning from wider areas prioritizes low-frequency components, we **control** s to regularize the Gaussians to cover wide 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 $9\pi s$, 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.[2] We find utilizing both the SLV random initialization and the progressive low-pass filter control is crucial to robustly guide the 3D Gaussians to learn the low-frequency first. Although the number of Gaussians N fluctuates, we name our strategy as *progressive* Gaussian low-pass filter control due to the tendency of Gaussians to increase exponentially.

[1] We provide a detailed proof in Section B.1 in the Appendix.
[2] We provide a detailed proof in Section B.2 in the Appendix.

6 Experiments

6.1 Datasets

To assess the effectiveness of our strategy, we conduct qualitative and quantitative comparisons using the same datasets previously utilized in the evaluation of the 3DGS method [13]. Specifically, we train our method on the Mip-NeRF360 dataset [4], the Tanks&Temples dataset [14], and the Deep Blending dataset [10]. 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 [4]. In terms of image resolution, we follow 3DGS and utilize a pre-downscaled dataset. Specifically, we use 4 times downscaled images for outdoor scenes and 2 times downscaled images for indoor scenes in the Mip-NeRF360 dataset. For the Tanks&Temples and Deep Blending datasets, we use the original images for both training and testing.

6.2 Implementation details

We implement our model based on the 3DGS [13], which consists of the PyTorch framework [26] and CUDA kernels for rasterization [13,15]. We follow the same training process of the existing implementation in all datasets. For our sparse-large-variance (SLV) random initialization, we set the initial number of Gaussians to $N = 10$ which is empirically found to be robust even for different datasets. For progressive low-pass filter control, we find that re-defining the value s as $s = \min(\max(HW/9\pi N, 0.3), 300.0)$ every 1,000 steps results in better results compared to changing the value every step and adopt this strategy as default. For training SH coefficients, we set the maximum degree as 3 following the original implementation. Unlike 13, we start increasing SH degree every 1,000 steps after 5,000 steps as our strategy facilitates the learning of low-frequency components in the early stage of training. Also to prevent Gaussians from rapidly becoming Gaussians with small variances, we lower the divide factor from 1.6 to 1.4. All other hyperparameters are left unchanged.

6.3 Qualitative and quantitative comparison

We compare our model with four methods: Plenoxels [39], InstantNGP-Base, InstantNGP-Big [22], and 3DGS [13]. To evaluate the effectiveness of our strategy when trained without SfM initialization, we make a thorough comparison with 3DGS trained using the original DSV random initialization. For DSV initialization, we uniformly sample points within three times the extent of the camera's bounding box. We set the initial number of Gaussians to $N = 1,000,000$. Our method achieves state-of-the-art results in all datasets, outperforming other models by a large margin.

Table 1: Quantitative comparison on Mip-NeRF360 dataset. We compare our method with previous approaches, including the random initialization method (DSV) described in the original 3DGS [13]. We report PSNR, SSIM, LPIPS and color each cell as best, second best and third best. †: As 3DGS [13] is the only method that utilizes SfM point clouds, the values are only included for reference.

Outdoor Scene															
	bicycle			flowers			garden			stump			treehill		
Method	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM ↑	LPIPS↓	PSNR↑	SSIM ↑	LPIPS↓
3DGS†	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
Plenoxels	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	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	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 (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	18.986	0.550	0.413
Ours	23.716	0.623	0.368	20.707	0.542	0.391	26.028	0.819	0.164	24.258	0.673	0.321	21.846	0.587	0.397

Indoor Scene															
	room			counter			kitchen			bonsai			average		
Method	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM ↑	LPIPS↓	PSNR↑	SSIM ↑	LPIPS↓
3DGS†	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
Plenoxels	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	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	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 (DSV)	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	0.705	0.313
Ours	30.240	0.894	0.249	27.826	0.870	0.260	30.235	0.913	0.149	30.702	0.920	0.239	26.180	0.761	0.280

Table 2: Quantitative comparison on Tanks&Temples and Deep Blending dataset. We compare our method with previous approaches, including the random initialization method (DSV) described in the original 3DGS [13]. We report PSNR, SSIM, LPIPS and color each cell as best, second best and third best. †: As 3DGS [13] is the only method that utilizes SfM point clouds, the values are only included for reference.

	Tanks&Temples						Deep Blending					
	Truck			Train			DrJohnson			Playroom		
Method	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM ↑	LPIPS↓
3DGS†	25.187	0.879	0.148	21.097	0.802	0.128	28.766	0.899	0.244	30.044	0.906	0.241
Plenoxels	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	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	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 (DSV)	21,149	0.758	0.248	20.824	0.772	0.255	28.668	0.894	0.258	28.358	0.896	0.258
Ours	23.732	0.841	0.196	21.144	0.763	0.267	28.993	0.894	0.260	30.261	0.908	0.255

Mip-NeRF360 dataset. We conduct experiments on Mip-NeRF360 dataset and report our results in Table 1. Our method outperforms all the other methods when SfM point clouds are not available. Also, we provide qualitative results in Figure 6 on novel test views. Compared to InstantNGP and 3DGS trained with DSV random initialization, ours outperforms them in visual quality rendering fine-grained details with the least noise. From the comparison of (c) and (e), we can observe the effectiveness of our strategy, removing any unwanted high-frequency artifacts or floaters that are prevalent in the rendered depth shown in (c). More qualitative images are included in the supplementary material.

Tanks&Temples and Deep Blending dataset. Table 2 shows the quantitative results on the Tanks&Temples and Deep Blending dataset which contain both indoor and outdoor scenes. Our method outperforms all other methods by a large margin, showing robustness regardless of domain shifts from indoor and outdoor scenes. Notably, in the "Train", "Dr Johnson" and "Playroom" scenes,

(a) INGP (b) 3DGS (DSV) (c) Depth of (b) (d) Ours (e) Depth of (d)

Fig. 6: Qualitative results on Mip-NeRF360 dataset.

(a) 3DGS (DSV) (b) Depth of (a) (c) Ours (d) Depth of (c)

Fig. 7: Qualitative results on Tanks&Temples and Deep Blending dataset.

Table 3: Ablation on core components.

Low-pass filter (LPF)	init.	Average		
		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**

Table 4: Impact of N.

N	PSNR↑	
	w/o LPF	w/ LPF
1M	22.190	23.937
100K	24.364	25.149
10K	24.798	25.959
1K	24.827	25.832
100	25.563	26.044
10	**25.675**	**26.180**

our method achieved a **higher PSNR** score than the 3DGS†, which is trained with **SfM initialization**, showing the effectiveness of our method of relaxing accurately initialized point clouds for 3DGS training.

In Figure 7, we provide qualitative comparisons on the Tanks&Temples and Deep Blending datasets. Compared to 3DGS trained with DSV initialization, our method succeeds in rendering high-quality images and accurate depth. These results are also well-aligned with our motivation and methods explained in Section 4 and Section 5, reducing any unwanted high-frequency artifacts.

6.4 Ablation studies

Ablation on core components. In Table 3, we validate the effectiveness of each component in our method. We compare our sparse-large-variance (SLV) initialization ($N = 10$) against the original dense-small-variance (DSV) initialization ($N = 1,000,000$). We also test our progressive Gaussian low-pass filter control against a constant filter ($s = 0.3$) used in the original 3DGS [13]. To further validate our progressive Gaussian low-pass filter control formula ($s = HW/9\pi N$, 'Ours'), we compare it against controlling the low-pass filter values sampled from linear, convex, and concave functions. Results verify our choice of dynamically adjusting the low-pass filter value based on the number of Gaussians, achieving the best performance. In addition, the results show that using both of our components: **1)** sparse-large-variance (SLV) random initialization and **2)** progressive Gaussian low-pass control is crucial showing the best performance. See Section A of the supplementary materials for more details of training with different low-pass filter value formulas.

Ablation on initial number of Gaussians. In Table 4, we validate the effectiveness of our sparse-large-variance (SLV) initialization. Aligned with our analysis, the performance of both 3DGS trained with DSV initialization and our strategy increases as the initial number of Gaussians decreases. This further verifies our hypothesis that the sparse initialization of Gaussians leads to a more robust learning of the true distribution.

Table 5: Few-shot Quantitative comparison on specific scene.

Method	bonsai			garden			kitchen		
	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓
3DGS (DSV)	13.246	0.508	0.538	16.752	0.377	0.444	18.181	0.629	0.384
Ours	**20.421**	**0.705**	**0.422**	**18.310**	**0.469**	**0.416**	**22.053**	**0.745**	**0.332**

6.5 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 [12, 16, 23, 29, 36]. However, due to the requirement of accurate pose and initial point clouds 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.

We conduct our experiments on three scenes (bonsai, garden, kitchen) from the Mip-NeRF360 dataset, utilizing only 10% of the total training images. Quantitative and qualitative results of the sparse view setting are demonstrated in Table 5 and Figure 8. 3DGS trained with our strategy (Ours) outperforms 3DGS trained with DSV random initialization (3DGS (DSV)) 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 8, it is shown that the rendered results of 3DGS trained with our strategy much more robustly models the real geometry of the scene.

(a) 3DGS (DSV) (b) Ours

Fig. 8: Qualitative comparison at sparse view setting.

7 Conclusion

In this work, we introduce **RAIN-GS**, a novel optimization strategy that enables 3DGS to render high-quality images even from randomly initialized point clouds. By combining sparse-large-variance (SLV) random initialization and progressive Gaussian low-pass filter control, our strategy successfully guides 3D Gaussians to learn the low-frequency components first, which we demonstrate to be crucial for robust optimization. We evaluate the effectiveness of our strategy through comprehensive quantitative and qualitative comparisons, achieving state-of-the-art performance in all datasets. By effectively removing the strict reliance on accurate point clouds achieved from Structure-from-Motion (SfM), **RAIN-GS** opens up new possibilities for 3DGS in scenarios where accurate point clouds are not obtainable.

References

1. Adamkiewicz, M., Chen, T., Caccavale, A., Gardner, R., Culbertson, P., Bohg, J., Schwager, M.: Vision-only robot navigation in a neural radiance world. IEEE Robotics and Automation Letters **7**(2), 4606–4613 (2022) 2
2. Agarwal, S., Furukawa, Y., Snavely, N., Simon, I., Curless, B., Seitz, S.M., Szeliski, R.: Building rome in a day. Communications of the ACM **54**(10), 105–112 (2011) 3
3. Barron, J.T., Mildenhall, B., Tancik, M., Hedman, P., Martin-Brualla, R., Srinivasan, P.P.: Mip-nerf: A multiscale representation for anti-aliasing neural radiance fields. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 5855–5864 (2021) 3
4. Barron, J.T., Mildenhall, B., Verbin, D., Srinivasan, P.P., Hedman, P.: Mip-nerf 360: Unbounded anti-aliased neural radiance fields. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 5470–5479 (2022) 3, 10, 27, 30
5. Du, Y., Smith, C., Tewari, A., Sitzmann, V.: Learning to render novel views from wide-baseline stereo pairs. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 4970–4980 (2023) 3
6. Frahm, J.M., Fite-Georgel, P., Gallup, D., Johnson, T., Raguram, R., Wu, C., Jen, Y.H., Dunn, E., Clipp, B., Lazebnik, S., 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. pp. 368–381. Springer (2010) 3
7. Fridovich-Keil, S., Meanti, G., Warburg, F.R., Recht, B., Kanazawa, A.: K-planes: Explicit radiance fields in space, time, and appearance. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 12479–12488 (2023) 2
8. Ge, Y., Behl, H., Xu, J., Gunasekar, S., Joshi, N., Song, Y., Wang, X., Itti, L., Vineet, V.: Neural-sim: Learning to generate training data with nerf. In: European Conference on Computer Vision. pp. 477–493. Springer (2022) 2
9. Geiger, A., Lenz, P., Stiller, C., Urtasun, R.: Vision meets robotics: The kitti dataset. The International Journal of Robotics Research **32**(11), 1231–1237 (2013) 2
10. Hedman, P., Philip, J., Price, T., Frahm, J.M., Drettakis, G., Brostow, G.: Deep blending for free-viewpoint image-based rendering. ACM Transactions on Graphics (ToG) **37**(6), 1–15 (2018) 10, 27, 30
11. Hong, S., Jung, J., Shin, H., Yang, J., Kim, S., Luo, C.: Unifying correspondence, pose and nerf for pose-free novel view synthesis from stereo pairs. arXiv preprint arXiv:2312.07246 (2023) 3
12. Jung, J., Han, J., Kang, J., Kim, S., Kwak, M.S., Kim, S.: Self-evolving neural radiance fields (2023) 14
13. Kerbl, B., Kopanas, G., Leimkühler, T., Drettakis, G.: 3d gaussian splatting for real-time radiance field rendering. ACM Transactions on Graphics **42**(4) (2023) 1, 2, 3, 4, 6, 8, 10, 11, 13, 18, 19, 24, 27, 30
14. Knapitsch, A., Park, J., Zhou, Q.Y., Koltun, V.: Tanks and temples: Benchmarking large-scale scene reconstruction. ACM Transactions on Graphics (ToG) **36**(4), 1–13 (2017) 10, 27, 30
15. Kopanas, G., Philip, J., Leimkühler, T., Drettakis, G.: Point-based neural rendering with per-view optimization. In: Computer Graphics Forum. vol. 40, pp. 29–43. Wiley Online Library (2021) 2, 10

16. seop Kwak, M., Song, J., Kim, S.: Geconerf: Few-shot neural radiance fields via geometric consistency (2023) 14
17. Li, Z., Wang, Q., Cole, F., Tucker, R., Snavely, N.: Dynibar: Neural dynamic image-based rendering. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 4273–4284 (2023) 3
18. Lin, C.H., Ma, W.C., Torralba, A., Lucey, S.: Barf: Bundle-adjusting neural radiance fields. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 5741–5751 (2021) 5
19. Liu, L., Gu, J., Zaw Lin, K., Chua, T.S., Theobalt, C.: Neural sparse voxel fields. Advances in Neural Information Processing Systems **33**, 15651–15663 (2020) 2
20. Luiten, J., Kopanas, G., Leibe, B., Ramanan, D.: Dynamic 3d gaussians: Tracking by persistent dynamic view synthesis. In: 3DV (2024) 3
21. Mildenhall, B., Srinivasan, P.P., Tancik, M., Barron, J.T., Ramamoorthi, R., Ng, R.: Nerf: Representing scenes as neural radiance fields for view synthesis. Communications of the ACM **65**(1), 99–106 (2021) 2, 3, 5, 27
22. Müller, T., Evans, A., Schied, C., Keller, A.: Instant neural graphics primitives with a multiresolution hash encoding. ACM Transactions on Graphics (ToG) **41**(4), 1–15 (2022) 3, 4, 10, 27
23. Niemeyer, M., Barron, J.T., Mildenhall, B., Sajjadi, M.S.M., Geiger, A., Radwan, N.: Regnerf: Regularizing neural radiance fields for view synthesis from sparse inputs (2021) 14
24. Nussbaumer, H.J., Nussbaumer, H.J.: The fast Fourier transform. Springer (1982) 5, 26
25. Park, K., Sinha, U., Barron, J.T., Bouaziz, S., Goldman, D.B., Seitz, S.M., Martin-Brualla, R.: Nerfies: Deformable neural radiance fields. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 5865–5874 (2021) 5
26. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S.: Pytorch: An imperative style, high-performance deep learning library (2019) 10
27. Reiser, C., Peng, S., Liao, Y., Geiger, A.: Kilonerf: Speeding up neural radiance fields with thousands of tiny mlps (2021) 2
28. Schonberger, J.L., Frahm, J.M.: Structure-from-motion revisited. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 4104–4113 (2016) 3
29. Song, J., Park, S., An, H., Cho, S., Kwak, M.S., Cho, S., Kim, S.: Därf: Boosting radiance fields from sparse input views with monocular depth adaptation. Advances in Neural Information Processing Systems **36** (2024) 3, 14
30. Sturm, J., Engelhard, N., Endres, F., Burgard, W., Cremers, D.: A benchmark for the evaluation of rgb-d slam systems. In: 2012 IEEE/RSJ international conference on intelligent robots and systems. pp. 573–580. IEEE (2012) 2
31. Tancik, M., Casser, V., Yan, X., Pradhan, S., Mildenhall, B., Srinivasan, P.P., Barron, J.T., Kretzschmar, H.: Block-nerf: Scalable large scene neural view synthesis (2022) 2
32. Tancik, M., Srinivasan, P., Mildenhall, B., Fridovich-Keil, S., Raghavan, N., Singhal, U., Ramamoorthi, R., Barron, J., Ng, R.: Fourier features let networks learn high frequency functions in low dimensional domains. Advances in Neural Information Processing Systems **33**, 7537–7547 (2020) 5

33. Tang, J., Ren, J., Zhou, H., Liu, Z., Zeng, G.: Dreamgaussian: Generative gaussian splatting for efficient 3d content creation. arXiv preprint arXiv:2309.16653 (2023) 3
34. Wu, C.: Towards linear-time incremental structure from motion. In: 2013 International Conference on 3D Vision-3DV 2013. pp. 127–134. IEEE (2013) 4
35. Xu, L., Agrawal, V., Laney, W., Garcia, T., Bansal, A., Kim, C., Rota Bulò, S., Porzi, L., Kontschieder, P., Božič, A., et al.: Vr-nerf: High-fidelity virtualized walkable spaces. In: SIGGRAPH Asia 2023 Conference Papers. pp. 1–12 (2023) 2
36. Yang, J., Pavone, M., Wang, Y.: Freenerf: Improving few-shot neural rendering with free frequency regularization. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 8254–8263 (2023) 3, 5, 14
37. Yang, Z., Gao, X., Zhou, W., Jiao, S., Zhang, Y., Jin, X.: Deformable 3d gaussians for high-fidelity monocular dynamic scene reconstruction. arXiv preprint arXiv:2309.13101 (2023) 3
38. Yi, T., Fang, J., Wang, J., Wu, G., Xie, L., Zhang, X., Liu, W., Tian, Q., Wang, X.: Gaussiandreamer: Fast generation from text to 3d gaussians by bridging 2d and 3d diffusion models. In: CVPR (2024) 3
39. Yu, A., Fridovich-Keil, S., Tancik, M., Chen, Q., Recht, B., Kanazawa, A.: Plenoxels: Radiance fields without neural networks (2021) 4, 10, 27
40. Yu, Z., Chen, A., Huang, B., Sattler, T., Geiger, A.: Mip-splatting: Alias-free 3d gaussian splatting. arXiv preprint arXiv:2311.16493 (2023) 8
41. Zhang, K., Riegler, G., Snavely, N., Koltun, V.: Nerf++: Analyzing and improving neural radiance fields (2020) 2
42. Zwicker, M., Pfister, H., Van Baar, J., Gross, M.: Ewa splatting. IEEE Transactions on Visualization and Computer Graphics **8**(3), 223–238 (2002) 4, 9

Appendix

In the Appendix, 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 **RAIN-GS** (**R**elaxing **A**ccurate **IN**itialization Constraint for 3D **G**aussian **S**platting) can be applied to 3D Gaussian splatting (3DGS) [13]. In Section B, we give a more detailed explanation of the convolution process with the Gaussian low-pass filter used in the rendering stage and how we choose the low-pass filter value s (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 4.2: 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 3DGS

We implement our strategy based on the official code of 3DGS [13]. As mentioned in Section 6.2, except for the hyperparameters for increasing spherical harmonic (SH) degrees and 3D Gaussian scale division ratio, we follow the original implementation and hyperparameters.

A.2 RAIN-GS

Our novel strategy consists of two components: **1)** sparse-large-variance (SLV) random initialization and **2)** progressive Gaussian low-pass filter control. In this section, we provide the pseudo codes for how we implement our strategy above 3DGS. We denote our implementation in the pseudo codes with a gray box.

Sparse-large-variance (SLV) random initialization. Following the original implementation of 3DGS [13], we initialize random point clouds within the boundary defined as three times the size of the camera's bounding box. Instead of the original dense-small-variance (DSV) random initialization where the initial number of Gaussians N is set as $N > 100K$, we only initialize 10 Gaussians as $N = 10$. This simple change of code (expressed with a gray box in Algorithm 1) based on our analysis of sparse-large-variance (SLV) initialization, largely improves the performance of 3DGS. 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.

Algorithm 1 Random point cloud initialization

$N \leftarrow$ Number of initial Gaussians (e.g., $N = 10$)
$P \leftarrow$ Cameras
$D \leftarrow$ MaxCameraDistance(P)
$i \leftarrow 0$ ▷ Iteration Count
while $i < N$ **do**
    $\mu \leftarrow$ RandomCubicSample($3 \times$ D)
    $\Sigma \leftarrow$ AvgNeighborDistance()
    $c \leftarrow$ RandomInit()
    $\alpha \leftarrow$ InverseSigmoid(1)
    $(M, S, C, A) \leftarrow (\mu, \Sigma, c, \alpha)$ ▷ Appends
    $i \leftarrow i + 1$
end while
return (M, S, C, A)

Progressive Gaussian low-pass filter control. In the original implementation of 3DGS, the Gaussian low-pass 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 h (mean $\mu = 0$ and variance $\sigma^2 = s = 0.3$). Instead of using a fixed sigma value σ 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 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 $\min(\max(HW/9\pi N, 0.3), 300.0)$. Our implementation is expressed with a gray box in Algorithm 2.

Algorithm 2 Progressive Gaussian low-pass filter control

$M \leftarrow$ SfM Points ▷ Positions
$S, C, A \leftarrow$ InitAttributes() ▷ Covariances, Colors, Opacities
$i \leftarrow 0$ ▷ Iteration Count
while not converged **do**
  $V, \hat{I}, H, W \leftarrow$ SampleTrainingView() ▷ Camera V, Image, Height and Width
  if LowPassRefinementIteration(i) **then**
    $h \leftarrow \text{Min}(\text{Max}(HW/9\pi N, 0.3), 300)$ ▷ Progressive low-pass filter control
  end if
  $I \leftarrow$ Rasterize(M, S, C, A, V, h) ▷ Rasterization with low-pass filter
  $L \leftarrow Loss(I, \hat{I})$ ▷ Loss
  M, S, C, $A \leftarrow$ Adam(∇L) ▷ Backprop & Step
  if IsRefinementIteration(i) **then**
    for all Gaussians (μ, Σ, c, α) **in** (M, S, C, A) **do**
      if $\alpha < \epsilon$ or IsTooLarge(μ, Σ) **then** ▷ Pruning
        RemoveGaussian()
      end if
      if $\nabla_p L > \tau_p$ **then** ▷ Densification
        if $\|S\| > \tau_S$ **then** ▷ Over-reconstruction
          SplitGaussian(μ, Σ, c, α)
        else ▷ Under-reconstruction
          CloneGaussian(μ, Σ, c, α)
        end if
      end if
    end for
  end if
  $i \leftarrow i + 1$
end while

A.3 Ablation on core components details

In Table 3 of Section 6.4, to demonstrate the effectiveness of our progressive Gaussian low-pass filter control strategy, we employed an ablation study utilizing 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 timestep, the remaining functions were manually defined to achieve $s = 300$ at step 0 and $s = 0.3$ at about 3,000 steps across all scenes.
For the convex function, we use the following formula for s scheduling:

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

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

$$s = \max(300 - 0.0997084x, 0.3). \tag{8}$$

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

$$s = \max(300 * (1 + 7^{-3} - 7^{\frac{x-3000}{1000}}), 0.3). \tag{9}$$

As using larger values for the Gaussian low-pass filter enables each Gaussian to model larger regions, this leads to more Gaussians overlapping to each of the tiles which indicates that more Gaussians need to be considered during the alpha blending process. We empirically find that the initial Gaussian low-pass filter value over $s > 300$ offers no significant improvement over $s = 300$, only making the overall computation inefficient. Based on these findings, we define the max value of the Gaussian low-pass filter as $s = 300$. The intuition of our choice of designing convex, linear, and concave functions to reach $s = 0.3$ at step 3,000 is based on our analysis that our adaptive Gaussian low-pass filter value reaches 0.3 between 2,000-3,000 steps. The illustration of different Gaussian low-pass value formulas is shown in Figure 9 and Figure 10 where our formula is adaptively defined, showing different functions for each scene.

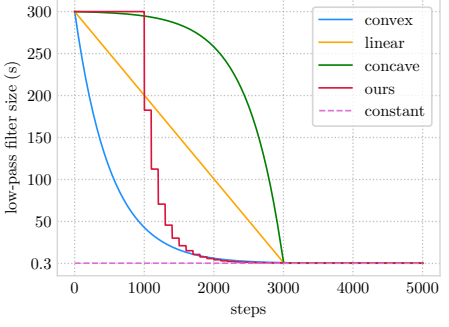


Fig. 9: Illustration of different Gaussian low-pass filter value formulas.

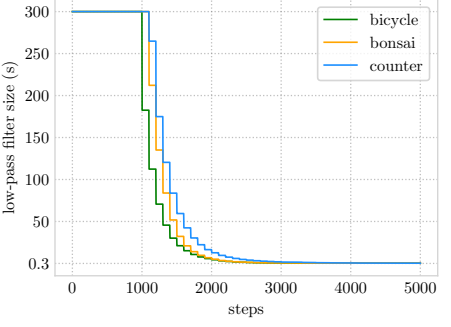


Fig. 10: Illustration of our Gaussian low-pass filter value formula.

B Proof

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

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

$$G'_i(x) = e^{-\frac{1}{2}(x-\mu'_i)^T \Sigma'^{-1}_i (x-\mu'_i)}. \tag{10}$$

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 Σ'_i as follows:

$$G'_i(x) = e^{-\frac{1}{2}(x-\mu'_i)^T (\Sigma'_i+sI)^{-1} (x-\mu'_i)}, \tag{11}$$

where I is the 2×2 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:

$$G_1(x) = e^{-\frac{1}{2}(x-\mu_i)^T V^{-1} (x-\mu_i)} \quad G_2(x) = e^{-\frac{1}{2}(x-\mu_i)^T Z^{-1} (x-\mu_i)}, \tag{12}$$

$$(G_1 \otimes G_2)(x) = e^{-\frac{1}{2}(x-\mu_i)^T (V+Z)^{-1} (x-\mu_i)}. \tag{13}$$

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.

B.2 Proof on progressive low-pass filter size

In Section 5.2, 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 Σ_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:

$$\begin{aligned} r &= k \cdot \sqrt{\max(\lambda_{i1}, \lambda_{i2}) + s}, \\ r &\geq k \cdot \sqrt{s}, \\ \pi r^2 &\geq k^2 \pi s, \end{aligned} \tag{14}$$

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

Implementation details. In this section, we provide a detailed explanation of the 1D regression toy experiment shown in Section 4.2. We design the experiment to learn a target distribution $Y(x)$, where x is in the range $[0, 10,000)$, using N 1D Gaussians. The target distribution $Y(x)$ is generated from random 10 1D Gaussian distributions. Each of the N 1D Gaussians includes learnable parameters of $\{\mu_i, \sigma_i, w_i\}$ and is trained through the L1 loss between the blended distribution from N 1D Gaussians with the weight w_i and $Y(x)$ as follows:

$$\mathcal{L} = \sum_{x} \left\| Y(x) - \sum_{i=0}^{N} w_i \cdot \exp\left(-\frac{(x-\mu_i)^2}{2\sigma_i^2}\right) \right\|, \tag{15}$$

For the dense-small-variance (DSV) scenario, we initialize the number of Gaussians as $N = 1,000$ and randomly select μ_i and σ_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 μ_i and σ_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 μ_i and σ_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.

D Additional analysis

D.1 Analysis on dense-large-variance (DLV) initialization

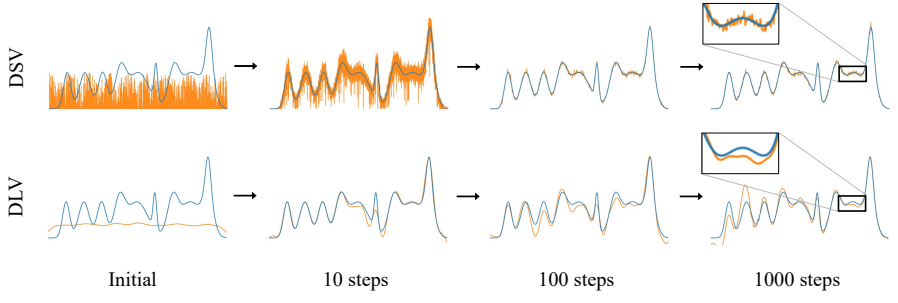


Fig. 11: Toy experiment learned from different dense initializations visualized in different steps. This figure visualizes the result of our toy experiment predicting the target distribution using a collection of 1D Gaussians, starting from different dense initialization methods. The top row shows the result with dense-small-variance (DSV) initialized Gaussians whereas the bottom row shows the result with dense-large-variance (DLV) initialized Gaussians.

In Section 4.2, we compare the influence of different dense initialization methods through the toy experiment of learning to regress a random 1D signal with

Table 6: Quantitative comparison on Mip-NeRF360 dataset with various initilize methods. We compare our method with the DSV random initialization method described in the original 3DGS [13] and DLV random initilization method. We report PSNR, SSIM, LPIPS.

Outdoor Scene															
	bicycle			flowers			garden			stump			treehill		
Method	PSNR↑	SSIM↑	LPIPS↓	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	18.986	0.550	0.413
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	19.605	0.550	0.440
3DGS (SLV)	**23.227**	**0.610**	**0.373**	**20.498**	**0.528**	**0.392**	**25.670**	**0.816**	**0.157**	**23.748**	**0.647**	**0.339**	**21.423**	**0.578**	**0.402**

Indoor Scene															
	room			counter			kitchen			bonsai			average		
Method	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM ↑	LPIPS↓	PSNR↑	SSIM ↑	LPIPS↓
3DGS (DSV)	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	0.705	0.313
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	23.246	0.702	0.337
3DGS (SLV)	**29.966**	0.892	0.268	**27.473**	**0.868**	**0.260**	**29.934**	**0.915**	**0.159**	**29.132**	**0.900**	**0.251**	**25.675**	**0.750**	**0.287**

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 11, 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.

D.2 Analysis on prioritized learning of low-frequency components

In Section 4.1, 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 4.2, we then show that sparse-large-variance (SLV) random initialization successfully guides the Gaussians to prioritize the learning of low-frequency components. Based on these findings, we propose a novel strategy (**RAIN-GS**) for 3DGS consisting of two main components: **1)** sparse-large-variance (SLV) random initialization and **2)** progressive Gaussian low-pass filter control. 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.

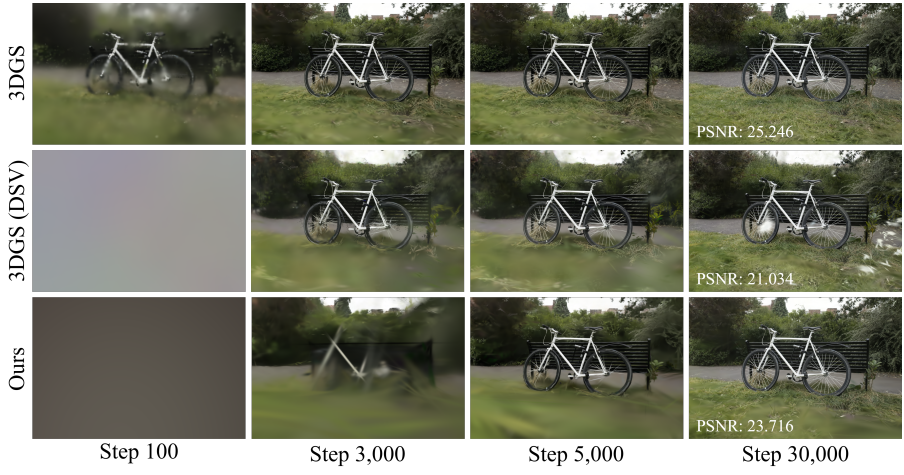


Fig. 12: Visualization of our strategy prioritizing the learning of coarse approximation. This figure visualizes the rendering result of the 'bicycle' scene from the Mip-NeRF360 dataset trained with 3DGS using different initialization methods. We show the images rendered from different steps of 100, 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 12, 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 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.

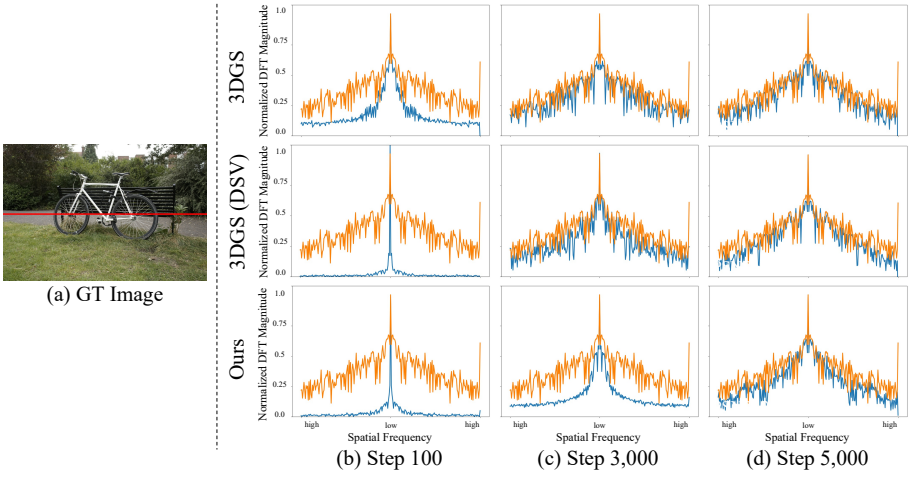


Fig. 13: Images from Figure 12 analyzed in the frequency domain.

In Figure 13, we further analyze the rendered images of Figure 12 in the frequency domain. We sample a horizontal line from the images as in (a) and perform FFT [24]. 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 4.1, 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 4.2, 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 12.

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.3 Comparison of computational resources

In this section, we now compare the computational resources of Plenoxels [39], InstantNGP [22], 3DGS [13], 3DGS with DSV random initialization, and 3DGS with our strategy. We evaluate different methods on the Mip-NeRF360 [4], Tanks&Temples [14], and Deep Blending [10] dataset.

Training time. Note that when pre-defined camera poses are available, methods other than 3DGS [13] can bypass the time-consuming Structure-from-Motion (SfM) process. To highlight this overhead, in Table 7 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. Our strategy, by utilizing sparse Gaussians in the early stage of training, reduces the training time compared to DSV random initialization.

Table 7: Training time.[3]

	Mip-NeRF360	Tanks&Temples	Deep Blending
Plenoxels	25m49s	25m5s	27m49s
INGP-Base	5m37s	5m26s	6m31s
INGP-Big	7m30s	6m59s	8m
3DGS[4]	41m33s + (19m9s)†	26m54s + (19m43s)†	36m2s + (6m23s)†
3DGS (DSV)	28m22s	18m31s	29m33s
Ours	25m47s	15m50s	24m4s

Memory size. In Table 8, we compare the memory size of different methods. InstantNGP [21], which utilizes a lightweight hash table outperforms all other methods in terms of memory using less than 50MB. As our strategy guides Gaussians to model larger regions in the early stages of training, we find that this leads to an additional advantage of modeling the scene with sparser Gaussians even after sufficient steps of training. This successfully reduces the memory requirements of Ours compared to other 3DGS approaches.

Table 8: Memory size.[5]

	Mip-NeRF360	Tanks&Temples	Deep Blending
Plenoxels	2.1GB	2.3GB	2.7GB
INGP-Base	13MB	13MB	13MB
INGP-Big	48MB	48MB	48MB
3DGS	734MB	411MB	676MB
3DGS (DSV)	592MB	365MB	582MB
Ours	581MB	359MB	545MB

[3] Except for the SfM processing time and training times of 3DGS(DSV) and Ours, we copied the values from 3DGS [13]. Note that 3DGS utilized a single A6000 GPU and we utilized a single RTX 3090 GPU for evaluation.

[4] The processing time of SfM is denoted with a †.

[5] Except for 3DGS(DSV) and Ours, we copied the memory size of different models from 3DGS [13].

D.4 Limitations

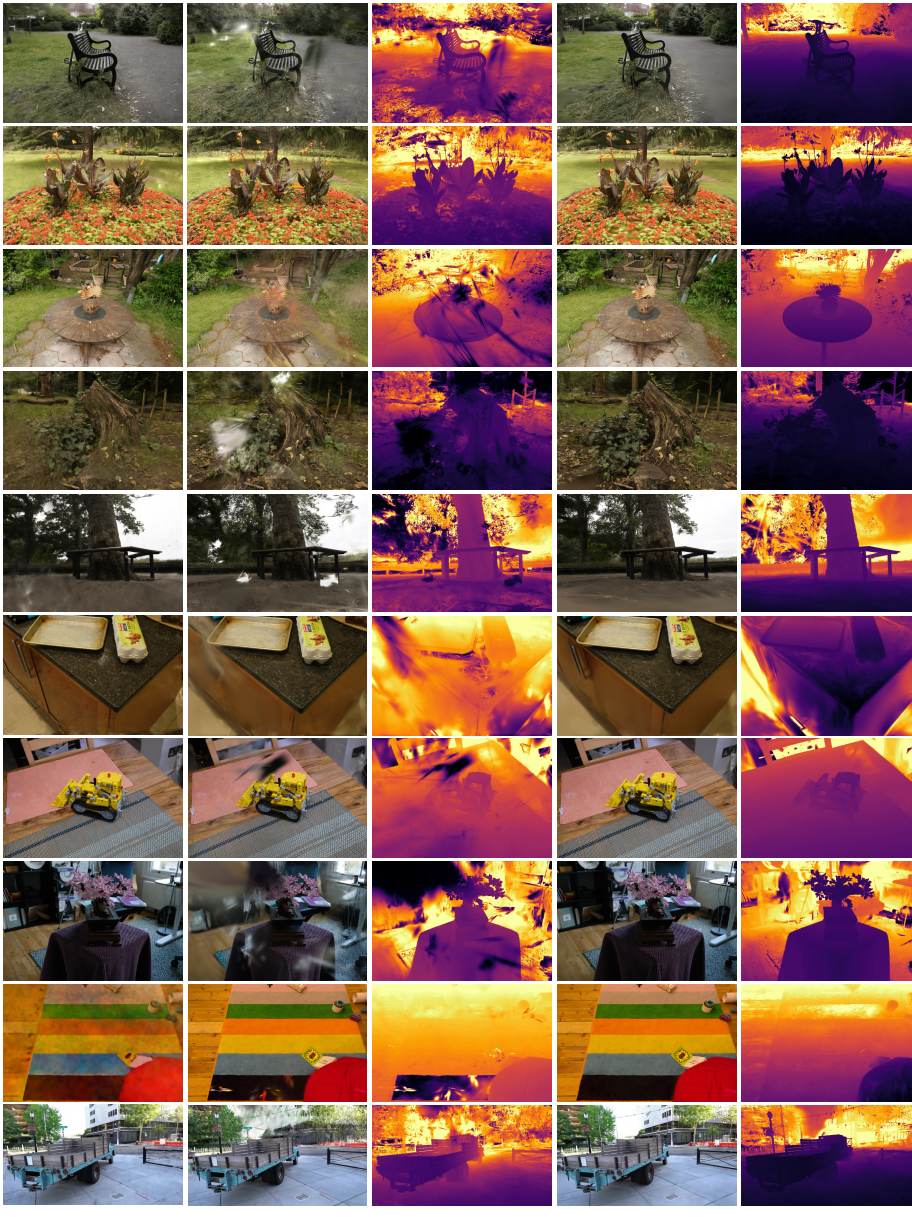
Although **RAIN-GS** robustly guides 3D Gaussians to model the scene even from randomly initialized point clouds, our novel strategy is not without limitations. As our strategy prioritizes learning the coarse approximation, it sometimes fails to detect the need for further densification to capture high-frequency 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 clouds, 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.

(a) Ours (b) 3DGS (c) GT image

Fig. 14: Visualization of failure cases of learning the coarse approximation first. 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 of 3DGS trained with our strategy in Figure 15.



(a) INGP (b) 3DGS (DSV) (c) Depth of (b) (d) Ours (e) Depth of (d)

Fig. 15: Additional qualitative results.

F Additional results of 3DGS in sparse view settings

In Table 9, Table 10, and Figure 16, we show additional quantitative and qualitative results comparing 3DGS (DSV) and 3DGS trained with our strategy in the sparse view setting in the Mip-NeRF360 dataset [4], Tanks&Temples [14], and Deep Blending [10] dataset. 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.

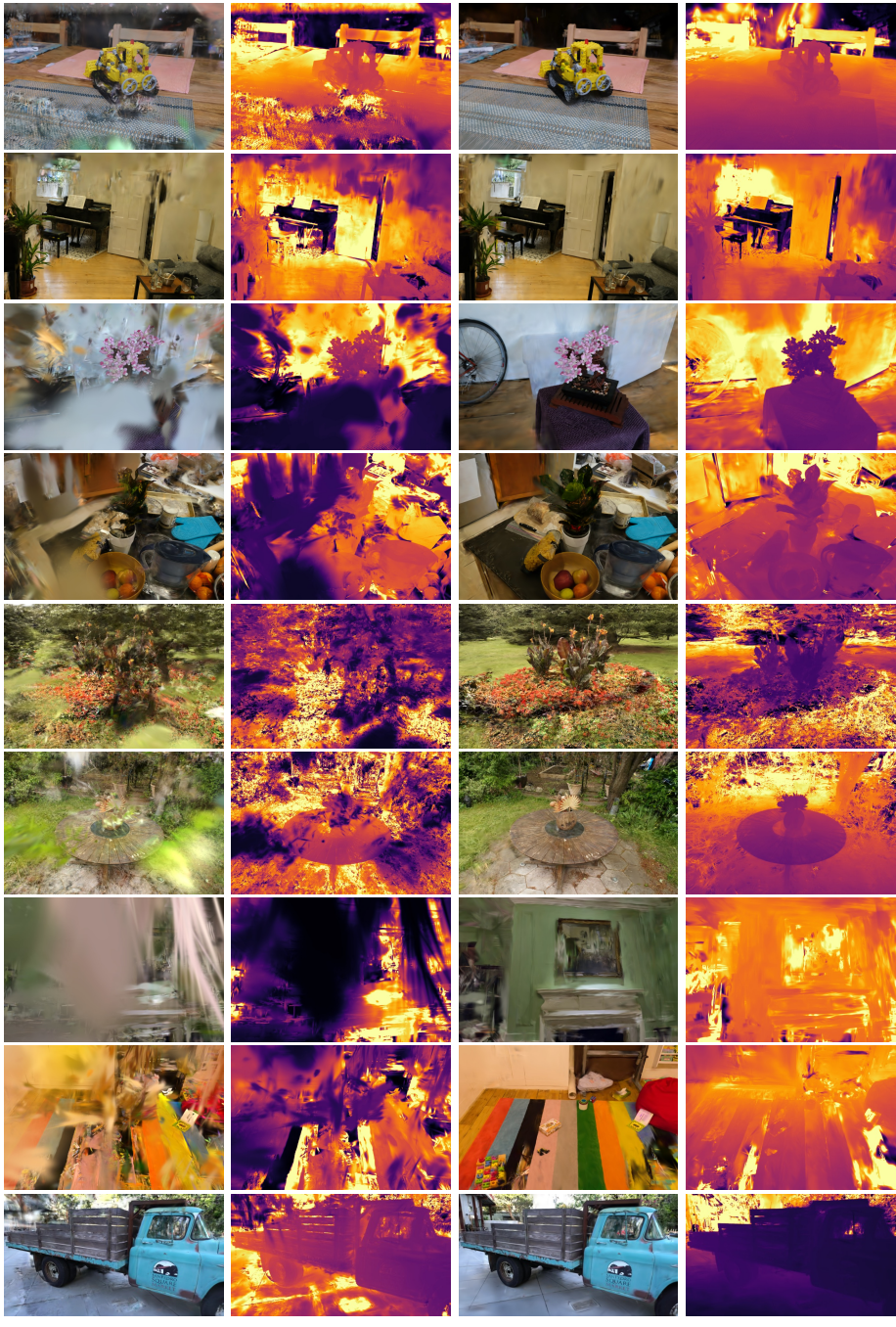
Table 9: Quantitative comparison on Mip-NeRF360 dataset in sparse-view setting. We compare our method with the random initialization method (DSV) described in the original 3DGS [13]. We report PSNR, SSIM, LPIPS.

Outdoor Scene															
	bicycle			flowers			garden			stump			treehill		
Method	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM ↑	LPIPS↓	PSNR↑	SSIM ↑	LPIPS↓
3DGS (DSV)	11.787	0.170	0.637	10.948	0.118	0.648	16.752	0.377	0.444	15.239	**0.194**	0.602	12.675	0.206	0.599
Ours	**13.107**	**0.181**	**0.626**	**12.413**	**0.181**	**0.584**	**18.310**	**0.469**	**0.416**	**15.448**	0.191	**0.594**	**14.460**	**0.249**	**0.580**

Indoor Scene															
	room			counter			kitchen			bonsai			average		
Method	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM ↑	LPIPS↓	PSNR↑	SSIM ↑	LPIPS↓
3DGS (DSV)	18.864	0.702	0.418	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	**20.536**	**0.745**	**0.404**	**18.557**	**0.666**	**0.432**	**22.053**	**0.745**	**0.332**	**20.421**	**0.705**	**0.422**	**17.256**	**0.459**	**0.488**

Table 10: Quantitative comparison on Tanks&Temples and Deep Blending dataset in sparse-view setting. We compare our method with the random initialization method (DSV) described in the original 3DGS [13]. We report PSNR, SSIM, LPIPS.

	Tanks&Temples						Deep Blending					
	Truck			Train			DrJohnson			Playroom		
Method	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM ↑	LPIPS↓
3DGS (DSV)	14.803	0.534	0.434	14.139	**0.506**	**0.478**	13.614	0.578	0.595	12.868	0.640	0.579
Ours	**16.387**	**0.591**	**0.408**	**14.268**	0.489	0.489	**17.013**	**0.644**	**0.511**	**16.158**	**0.693**	**0.498**



(a) 3DGS (DSV) (b) Depth of (a) (c) Ours (d) Depth of (c)

Fig. 16: Qualitative results in sparse view settings.