

Efficient Density Control for 3D Gaussian Splatting

Xiaobin Deng Changyu Diao* Min Li Ruohan Yu Duanqing Xu*

Zhejiang University

{dengxiaobin, dcy, lmin, yuruohan, xdq}@zju.edu.cn

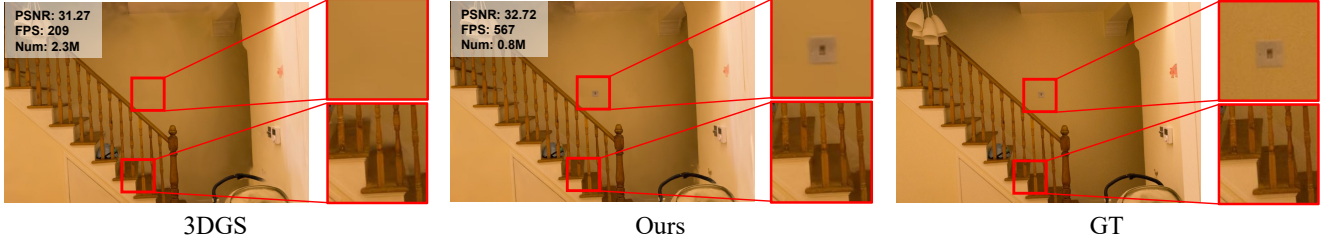


Figure 1. By improving adaptive density control of 3DGS [8], our EDC achieves superior rendering quality while using only one-third of Gaussians. Benefiting from a more efficient long-axis split operation, we successfully recover details of the switch on the wall and the trim line along the stairs in the playroom [6] scene.

Abstract

3D Gaussian Splatting (3DGS) excels in novel view synthesis, balancing advanced rendering quality with real-time performance. However, in trained scenes, a large number of Gaussians with low opacity significantly increase rendering costs. This issue arises due to flaws in the split and clone operations during the densification process, which lead to extensive Gaussian overlap and subsequent opacity reduction. To enhance the efficiency of Gaussian utilization, we improve the adaptive density control of 3DGS. First, we introduce a more efficient long-axis split operation to replace the original clone and split, which mitigates Gaussian overlap and improves densification efficiency. Second, we propose a simple adaptive pruning technique to reduce the number of low-opacity Gaussians. Finally, by dynamically lowering the splitting threshold and applying importance weighting, the efficiency of Gaussian utilization is further improved. We evaluate our proposed method on various challenging real-world datasets. Experimental results show that our Efficient Density Control (EDC) can enhance both the rendering speed and quality.

1. Introduction

Novel view synthesis (NVS) is a classical problem in computer vision, with widespread applications in virtual reality, cultural heritage preservation, autonomous driving, and other fields. Neural Radiance Field (NeRF) [15] introduced

the use of neural networks to learn the structure and features of a scene, requiring only multi-view 2D images as training data to synthesize high-quality novel views. However, NeRF suffers from long synthesis times for individual views [5, 17], making real-time rendering challenging.

Recently, 3D Gaussian Splatting (3DGS) [8] has attracted attention due to its explicit representation and real-time rendering performance. 3DGS represents a scene using a large number of 3D Gaussian ellipsoids. The properties of these Gaussians include position, size, shape, opacity, and color, all of which can be optimized through differentiable rendering. 3DGS generates an initial set of Gaussians from the sparse points obtained through Structure from Motion (SfM) [18], and subsequently refines the scene representation by increasing Gaussian density via adaptive density control.

The rendering and storage overhead of 3DGS is proportional to the number of Gaussians. A key issue is how to achieve the same or even better rendering quality with fewer Gaussians. By analyzing the number distribution of Gaussians across different opacity intervals, we observe an excessive proportion of Gaussians with low opacity. For instance, in the bicycle scene, more than half of the Gaussians have opacity values less than 0.1 (see Figure 2). The rendering contribution of each Gaussian is proportional to its opacity. Gaussians with low opacity contribute minimally to the rendering process, yet they significantly increase both rendering and storage costs. We find that flaws in the clone and split operations during the densification process result in an abnormal proportion of low-opacity Gaussians. The split operation replaces a large Gaussian with two smaller

*Corresponding authors.

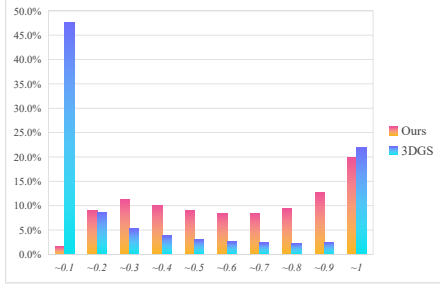


Figure 2. We compared the proportions of Gaussians within different opacity ranges in the bicycle [1] scene, trained using 3DGS and our proposed method.

ones, where the positions of the smaller Gaussians are determined through probabilistic sampling. This leads to the split Gaussians overlapping with a certain probability. The clone operation simply replicates a small Gaussian and hopes the original Gaussian will move away from the clone through parameter updates. However, the extent of each parameter update is limited and gradually decreases as training progresses. As a result, the Gaussian and its clone are likely to overlap. To prevent the opacity in the overlapping regions from becoming too large, the opacities of the overlapping Gaussians are gradually reduced during optimization, resulting in a high proportion of low-opacity Gaussians in the scene.

To improve the efficiency of Gaussian utilization, we enhance the adaptive density control of 3DGS. First, we replace the original clone and split with long-axis split. Specifically, we use the longest axis of the Gaussian as the splitting dimension. The spacing of the child Gaussians is controlled to avoid overlap. We adjust the shape and opacity of the child Gaussians to minimize changes in the shape and density distribution of the covered area before and after splitting, thus maximizing the efficiency of the splitting process. Second, to further reduce the proportion of low-opacity Gaussians, we propose a simple adaptive pruning method. Finally, by dynamically adjusting the splitting threshold and using importance weighting when computing the average gradient, we further improve the efficiency of Gaussian utilization. Our method not only significantly reduces the number of Gaussians in the scene but also enhances the ability to recover fine details (see Figure 1). On the challenging Mip-NeRF 360 dataset, we reduced the number of Gaussians by 30% while improving the average PSNR from 25.52 to 25.82. Our method is easy to implement and will be open-sourced in the future. Summary of our contributions:

- We found that the clone and split operations used during the densification process of 3DGS have inherent limitations. These limitations result in a large number of low-opacity Gaussians, which in turn cause unnecessary overhead. The proposed long-axis split operation effectively

mitigates this issue, thereby improving densification efficiency.

- We propose a simple adaptive pruning method that significantly reduces the proportion of low-opacity Gaussians without affecting rendering quality.
- The dynamic thresholding and adaptive weighting we introduced further enhance the efficiency of Gaussian utilization.

2. Related Works

3D Gaussian Splatting (3DGS) [8] demonstrates outstanding performance in both rendering quality and speed, representing the current state-of-the-art in novel view synthesis. 3DGS has been widely adopted across fields including dynamic scenes [12, 22], simultaneous localization and mapping (SLAM) [7, 14, 23], 3D content generation [2, 20, 21, 26], autonomous driving [31], and high-fidelity human avatars [10, 13, 16, 19].

Numerous studies have focused on improving 3DGS rendering quality. For instance, Mip-Splatting [27] introduces a 3D smoothing filter and a 2D mipmap filter to eliminate aliasing artifacts present in 3DGS during scaling. To mitigate the impact of defocus blur on reconstruction quality, Deblurring 3DGS [11] applies a small multi-layer perceptron (MLP) to the covariance matrix, learning spatially varying blur effects. GaussianPro [3] leverages optimized depth and normal maps to guide densification, filling gaps in areas initialized via structure-from-motion (SfM) [18]. Spec-Gaussian [24] employs anisotropic spherical Gaussian appearance fields for Gaussian color modeling, enhancing 3DGS rendering quality in complex scenes with specular and anisotropic surfaces. Notably, all these enhancements rely on the original density control and could benefit from our proposed work.

3DGS increases Gaussians in a scene through a basic density control mechanism, yet the optimized scene still exhibits blurred regions that are challenging to refine merely by adding more Gaussians. Mini-splatting [4] addresses this by generating depth maps for trained scenes to reinitialize the sparse points, and identifies blurred Gaussians with large rendering areas during training, splitting them as needed. Pixel-GS [30] addresses blur by using the average gradient weighted by the pixel area covered by Gaussians in each view. AbsGS [25] and GOF [28] attribute blur in reconstructions to conflicts in gradient direction across pixels when computing Gaussian coordinate gradients. This conflict leads to larger Gaussians, which represent blur, receiving insufficient average gradients. To resolve this, they compute Gaussian coordinate gradients by taking the modulus of pixel coordinate gradients before summing. The above studies propose improvements for initialization and densification criteria but do not address the densification operation itself. In our work, we analyze and enhance the

densification operation in adaptive density control, resolving issues related to Gaussian overlap and improving densification efficiency.

3. Methods

3.1. Preliminaries

3DGS defines the scene as a set of anisotropic 3D Gaussian primitives:

$$G(x) = \exp\left(-\frac{1}{2}(x)^T \Sigma^{-1}(x)\right), \quad (1)$$

where Σ is the 3D covariance matrix and x represents the position relative to the Gaussian mean coordinates. To ensure the semi-definiteness of the covariance matrix, 3DGS reparameterizes it as a combination of a rotation matrix R and a scaling matrix S :

$$\Sigma = RSS^T R^T. \quad (2)$$

The scaling matrix S can be represented using a 3D vector s , while the rotation matrix R is obtained from the quaternion q . To render an image from a specified viewpoint, the color of each pixel p is obtained by blending N ordered Gaussians $\{G_i \mid i = 1, \dots, N\}$ that cover pixel p , with the following formula:

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

where α_i is the value obtained by projecting G_i onto p and multiplying by the opacity of G_i , while c_i represents the color of G_i , expressed by SH coefficients.

3DGS initializes the scene using sparse points generated by SfM, and then increases the number and density of Gaussians in the scene through adaptive density control. Specifically, 3DGS calculates the cumulative average view-space positional gradients of Gaussians every 100 iterations, with each iteration training a single viewpoint. The formula for calculating the average gradient is as follows:

$$\frac{\sum_{k=1}^{M^i} \sqrt{\left(\frac{\partial L_k}{\partial \mu_{k,x}^i}\right)^2 + \left(\frac{\partial L_k}{\partial \mu_{k,y}^i}\right)^2}}{M^i} > \tau_{\text{pos}}, \quad (4)$$

where M^i represents the number of viewpoints in which the Gaussian participates during a cycle, τ_{pos} is the given densification threshold, $\frac{\partial L_k}{\partial \mu_{k,x}^i}$ and $\frac{\partial L_k}{\partial \mu_{k,y}^i}$ represent the gradients of the Gaussian with respect to the x and y for the current viewpoint, obtained by summing the gradients of each pixel with respect to the coordinates:

$$\frac{\partial L_k}{\partial \mu_{k,x}^i} = \sum_{j=1}^m \frac{\partial L_j}{\partial \mu_{i,x}}, \quad \frac{\partial L_k}{\partial \mu_{k,y}^i} = \sum_{j=1}^m \frac{\partial L_j}{\partial \mu_{i,y}}. \quad (5)$$

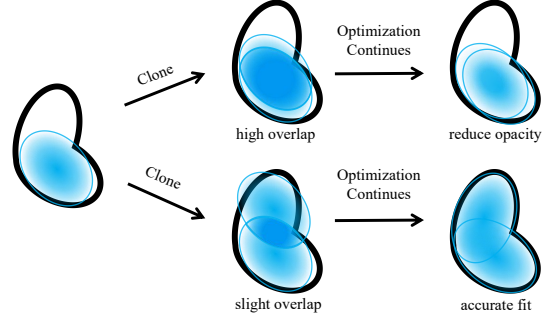


Figure 3. We analyze two potential scenarios arising from the clone operation. In the ideal case (illustrated in the figure below), the two Gaussians exhibit minimal overlap, allowing them to independently adjust their positions and shapes during subsequent optimization to better fit the scene. In contrast, when the two Gaussians experience significant overlap (as shown in the figure above), they receive similar gradients, making independent optimization challenging. As a result, their opacity is reduced to prevent the opacity in the overlapping region from becoming excessively large.

Gaussians with average gradients exceeding a predefined threshold undergo densification using either clone or split, depending on their size.

Clone: A small Gaussian is copied, with the offspring being identical to the parent, resulting in an overlap between the two. In the subsequent parameter updates, the parent’s parameters are optimized, causing its position to change. However, since the offspring does not inherit the parent’s gradient during the copying process, its parameters remain unchanged. Ideally, before the next iteration, the parent and offspring should separate from their overlapping state.

Split: One large Gaussian is replaced by two smaller Gaussians, which have the same shape, opacity, and color. Each small Gaussian is scaled to $1/1.6$ of the parent’s size. The coordinates of the two small Gaussians are generated through Gaussian sampling, using the parent’s coordinates and covariance matrix as parameters.

3.2. Limitations of clone and split

Limitation of clone operation: The clone operation aims to increase the number of small Gaussians that cannot cover fine-scale structures (see Figure 3). The overlap between the original Gaussian and the cloned Gaussian depends on how much the position of the original Gaussian changes in the current iteration. Ideally, the two Gaussians will have a small overlap, allowing their positions and shapes to be optimized independently. However, the extent of position change for the original Gaussian during cloning is determined by both the magnitude of its position gradient in the current view and the position learning rate. As optimization progresses, both the position gradient received by the

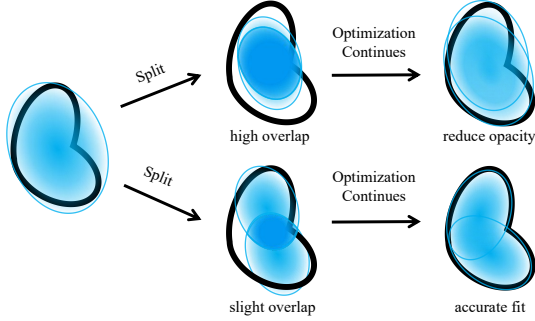


Figure 4. We analyze the performance of the split operation under varying levels of overlap. When the overlap is minimal (as shown in the figure below), the two split Gaussians independently adjust their positions and shapes during subsequent optimization to better fit the scene. In contrast, when the overlap is significant (as shown in the figure above), the Gaussians receive similar gradients, making it challenging for them to be optimized independently. Consequently, their opacity is reduced to prevent the opacity in the overlapping region from becoming excessively large.

Gaussian and the position learning rate gradually decrease. Additionally, under the same viewpoint, different Gaussians receive varying gradients, and some may require less movement during the iteration with the clone operation. Consequently, the clone operation cannot avoid overlap between the original and cloned Gaussians. Due to receiving similar gradients, highly overlapping Gaussians are difficult to separate quickly through parameter updates. For Gaussians that remain overlapped for an extended period, the scene tends to lower their opacity to prevent excessive opacity in the overlapping regions.

Limitation of split operation: The split operation aims to replace large Gaussians that excessively cover fine-scale structures with two smaller Gaussians (see Figure 4). Ideally, the split Gaussians should gradually fit the complex geometric structures. However, split Gaussians generated by independent and identically distributed Gaussian sampling have a spacing that follows a zero-mean Gaussian distribution, which results in a probability of significant overlap.

During the training process, the shape of the Gaussians gradually approaches the target geometric structure, and the same holds for the over-reconstructed regions. Since the two smaller Gaussians maintain the same shape as the original, the region they cover does not perfectly align with the original Gaussian’s shape, resulting in a deviation from the target geometric structure (see Figure 5). This geometric deviation can slow down the convergence speed during the optimization process and reduce the final rendering quality.

Additionally, probabilistic sampling introduces extra uncertainty, leading to greater fluctuations in the training results.

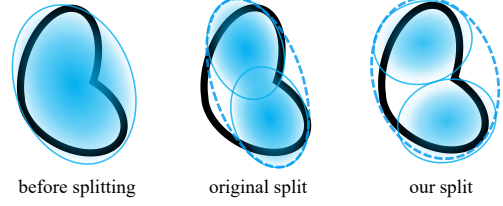


Figure 5. The original split method does not alter the shape of the sub-bodies, resulting in a shape formed by the child Gaussians that differs from the original shape of the parent Gaussian. In contrast, our method shortens the child Gaussians along their longest axis, ensuring that the shape of the covered region before and after the split remains approximately the same, thereby maximizing the densification efficiency.

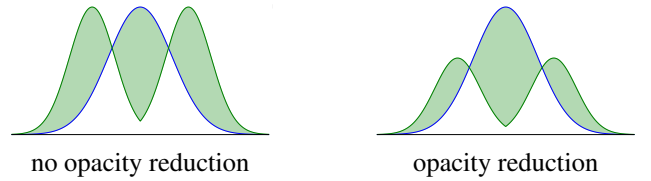


Figure 6. After splitting, the density of the corresponding region changes from a unimodal Gaussian distribution (blue curve) to a bimodal Gaussian distribution (green curve). If the opacity is not altered after the split (left), the change in the density distribution before and after the split is significant. By appropriately reducing the opacity of the split Gaussians (right), the variation in the density distribution can be reduced. The green shaded area represents the difference in density distribution before and after splitting.

3.3. Long-axis Split

To better describe the differences between the long-axis split operation and the original split operation, we will present the explanation in three parts.

Position and Shape: If we consider a Gaussian as a point, the split operation is essentially performed along one dimension. Among the three axes of a Gaussian, we use its longest axis as the splitting dimension. The split Gaussians are positioned symmetrically on both sides of the original Gaussian’s longest axis. To maximize the utilization of the shape information from the original Gaussian, we carefully adjust the shape of the child Gaussians to ensure that the overall shape remains consistent before and after splitting. Specifically, the radius along the longest axis of each child Gaussian is halved, while the radii along the other axes are shortened to 85% of their original values. This adjustment also helps mitigate the issue of excessively elongated Gaussians. Additionally, we set the spacing to the maximum radius of the original Gaussian to prevent overlap (see Figure 5).

Opacity: The split operation transforms the density distribution of the corresponding region from a single-center to a dual-center distribution (see Figure 6). To reduce the

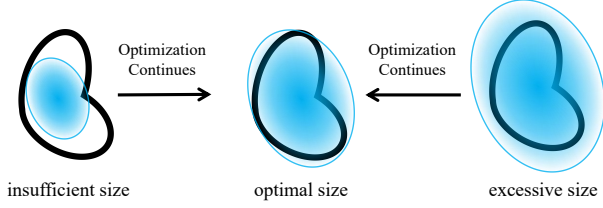


Figure 7. Through the process of backpropagation optimization, the parameters are driven to converge towards a single optimum point. Whether in cases of overfitting or underfitting, the sizes of the Gaussians tend to stabilize at a single consistent value.

impact of density distribution changes before and after splitting, we lower the opacity of each child Gaussian to 60% of the original Gaussian. This adjustment could improve the final rendering quality. Detailed experimental results can be found in Section 4.5.

Split Only: 3DGS uses clone and split to address under-reconstruction and over-reconstruction. However, during optimization, the size of Gaussians tends to converge to a value that balances under-reconstruction and over-reconstruction to minimize loss (see Figure 7). Therefore, we use only the long-axis split as the densification method.

3.4. Adaptive Pruning

3DGS performs an opacity reset operation every 3000 iterations. After resetting and restoring opacity, the opacity of the Gaussians may change, potentially resulting in some Gaussians having low opacity. Furthermore, during the normal densification and optimization process, the opacity of Gaussians with lower rendering priority may gradually decrease. Therefore, in addition to the optimization of the densification operation, additional methods are needed to limit the number of low-opacity Gaussians.

We use adaptive pruning to limit the number of low-opacity Gaussians. Specifically, after the rapid densification, starting from the 6Kth iteration, we prune Gaussians with opacity less than 0.1 every 3000 iterations. This pruning operation continues until the training ends. The over-coverage of Gaussians can reduce overall opacity, potentially causing the opacity of the split Gaussians to increase. As a result, early pruning may negatively affect the final rendering quality. Therefore, we opt to perform pruning after the rapid densification. We set the interval between two pruning operations to 3000 iterations, giving the scene enough time to adapt to the pruning effect. On the other hand, pruning and opacity resetting are executed simultaneously, reducing training fluctuations. Experimental results (as shown in Section 4.5) demonstrate that our adaptive pruning surpasses one-time pruning operation.

3.5. Dynamic Thresholds

During the densification process, the number of Gaussians rapidly reaches a peak. Rapid densification may cause newly split Gaussians to be further split before they have been fully optimized, even though optimization would eliminate the need for further splitting. We choose to dynamically lower the splitting threshold, allowing the scene to first densify the areas with higher errors and then gradually optimize the details.

Specifically, for a given splitting threshold τ_{pos} , we set the initial threshold to $2\tau_{\text{pos}}$. At the 4K, 7K, and 10K iterations, the splitting threshold is set to $1.5\tau_{\text{pos}}$, $1.2\tau_{\text{pos}}$, and τ_{pos} , respectively. Additionally, we pause the split operation for 1000 iterations before lowering the threshold, giving the scene sufficient time to optimize the previously densified areas.

3.6. Importance Weighting

When using the average gradient across views as the basis for splitting, high-frequency regions that appear only in a few specific views receive larger average gradients, making them more likely to densify. Generally, objects that appear consistently across multiple views are considered more important. Therefore, we use the frequency of a Gaussian’s appearance across training views as a weight, ensuring that important regions are allocated more Gaussians to improve rendering quality.

3DGS uses the average view coordinate gradient as the basis for splitting, but due to the existence of gradient conflicts [25], it struggles to handle blurry regions with low mean and large variance. Following the method in [25, 28], we calculate the average view gradient by first taking the modulus of the pixel gradients and then summing them. We determine whether a Gaussian should split using the following formula:

$$\frac{\sum_{k=1}^{M^i} \sqrt{\left(\frac{\partial L_k}{\partial \mu_{k,x}^i}\right)^2 + \left(\frac{\partial L_k}{\partial \mu_{k,y}^i}\right)^2}}{M^i} \left(1 + \frac{M^i}{M} \lambda\right) > \tau_{\text{pos}}, \quad (6)$$

where λ is a given weight used to adjust the magnitude of the importance weighting, M is the total number of training iterations within a period.. The formulas for calculating $\frac{\partial L_k}{\partial \mu_{k,x}^i}$ and $\frac{\partial L_k}{\partial \mu_{k,y}^i}$ are as follows:

$$\frac{\partial L_k}{\partial \mu_{k,x}^i} = \sum_{j=1}^m \left| \frac{\partial L_j}{\partial \mu_{i,x}} \right|, \quad \frac{\partial L_k}{\partial \mu_{k,y}^i} = \sum_{j=1}^m \left| \frac{\partial L_j}{\partial \mu_{i,y}} \right|. \quad (7)$$

4. Experiments

4.1. Datasets and metrics

We evaluated our method on real-world scenes from the Mip-NeRF 360 [1], Tanks and Temples [9], and Deep

Dataset Method—Metric	Mip-NeRF360						Deep Blending						Tanks&Temples					
	SSIM [†]	PSNR [†]	LPIPS [‡]	GS Num	FPS	Train	SSIM [†]	PSNR [†]	LPIPS [‡]	GS Num	FPS	Train	SSIM [†]	PSNR [†]	LPIPS [‡]	GS Num	FPS	Train
Plenoxels*	0.626	23.08	0.463	-	6.79	25m49s	0.795	23.06	0.510	-	11.2	27m49s	0.719	21.08	0.379	-	13.0	25m5s
INGP-Big*	0.699	25.59	0.331	-	9.43	7m30s	0.817	24.96	0.390	-	2.79	8m	0.745	21.92	0.305	-	14.4	6m59s
M-NeRF360*	0.792	27.69	0.237	-	0.06	48h	0.901	29.40	0.245	-	0.09	48h	0.759	22.22	0.257	-	0.14	48h
3DGS	0.815	27.48	0.215	3337659	169.03	21m27s	0.904	29.57	0.244	2832494	177.86	19m52s	0.847	23.68	0.177	1847041	227.85	11m39s
BaseLine	0.820	27.52	0.198	3194225	162.94	22m30s	0.905	29.49	0.243	2054043	231.94	17m40s	0.856	23.83	0.164	1128961	270.53	10m30s
Ours-s	0.826	27.68	0.206	1274741	387.32	13m50s	0.912	30.0	0.237	629380	661.82	11m43s	0.857	23.90	0.172	557700	663.86	6m24s
Ours-m	0.833	27.82	0.184	2261543	279.51	18m4s	0.911	30.1	0.229	1150167	477.89	13m48s	0.864	24.02	0.151	980690	458.42	8m12s

Table 1. Quantitative results on the Mip-NeRF 360, Deep Blending, and Tanks and Temples datasets. Cells are highlighted as follows: **best**, **second best**, and **third best**. Data for Plenoxels [5], INGP-Big [17], and M-NeRF360 [1] are sourced from the 3DGS [8] paper and marked with an asterisk (*). Since our hardware differs from that used in the 3DGS paper, FPS and Train metrics here provide only an approximate reference. Results without an asterisk are retrained by us.

Blending [6] datasets. We selected all nine scenes from the Mip-NeRF 360 dataset, including five outdoor scenes and four indoor scenes. For the Tanks and Temples dataset, we chose the train and truck scenes, and for the Deep Blending dataset, we selected the drjohnson and playroom scenes. As with 3DGS, in each experiment, every 8th image was used as the validation set. We report peak signal-to-noise ratio (PSNR), structural similarity (SSIM), and perceptual metric (LPIPS) from [29] as quality evaluation metrics. All three scores were calculated using the methods provided by 3DGS.

4.2. Implementation

We built our code on top of the open-source 3DGS codebase. We use a modified version of 3DGS, where the method for calculating the average gradient has been altered [25, 28], as the **baseline** for comparison. Some hyperparameters in 3DGS are not optimal for the baseline configuration, so we adjusted certain hyperparameters to improve its performance. Specifically, we slightly reduced the position learning rate, increased the opacity reset threshold from 0.01 to 0.1, and extended the opacity reset operation until the end of training. To mitigate potential floating artifacts in some scenes, we pruned Gaussians with opacity below 0.02 during the warm-up phase of training. We set the importance weighting coefficient λ to 0.3. The baseline uses a densification threshold of 0.0005, while our approach uses thresholds of 0.00035 and 0.0006. Apart from the densification threshold, our approach shares the same hyperparameters as the baseline. We also reproduced the original 3DGS as a reference. For all configurations, the training iterations were set to 30K. All experiments were performed using a single 4090 GPU.

4.3. Main Results

Quantitative Analysis: In the first set of experiments, we demonstrate the overall improvements of our method, as shown in Table 1. For the sake of completeness, we also present the results of three NeRF improvements from [8] for comparison. The standard version of our method achieved the best rendering quality across all three datasets, while the compact version also achieved the second-best quality.

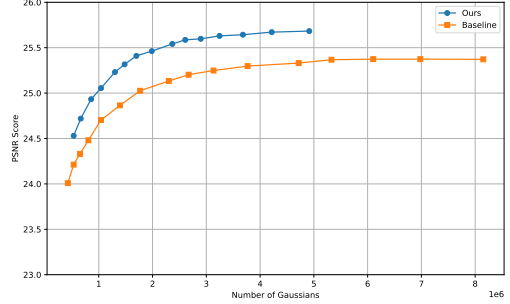


Figure 8. Comparison of rendering quality between our approach and the baseline under different numbers of Gaussians in the bicycle scene.

The baseline addresses the issue of reconstruction blur and outperforms 3DGS. Compared to the baseline, our method significantly enhances rendering quality while using substantially fewer Gaussians. Notably, in the Deep Blending dataset, although the baseline reduced the Gaussian count by 0.8 million compared to 3DGS, it also decreased rendering quality. This is because the two indoor scenes in the Deep Blending dataset were better initialized, resulting in less reconstruction blur. Our compact version, which surpasses the baseline across all three metrics, uses only 30% of the Gaussians, demonstrating a significant improvement in Gaussian utilization efficiency. Thanks to the reduced Gaussian count, our method also achieved substantial improvements in rendering and training speed over the baseline. Compared to the baseline, our approach achieves a greater improvement in rendering speed than the reduction in the number of Gaussians, which is particularly evident on the Tanks and Temples dataset. This is because when the proportion of low-opacity Gaussians is too high, the average number of Gaussians contributing to the rendering of a single pixel increases.

We further tested the rendering performance of our method and the baseline at various Gaussian counts in the bicycle scene (see Figure 8). In each range, our method significantly outperformed the baseline. As the Gaussian count increased, our method exhibited faster quality improvements and reached its peak performance earlier. This demonstrates that our method offers comprehensive en-

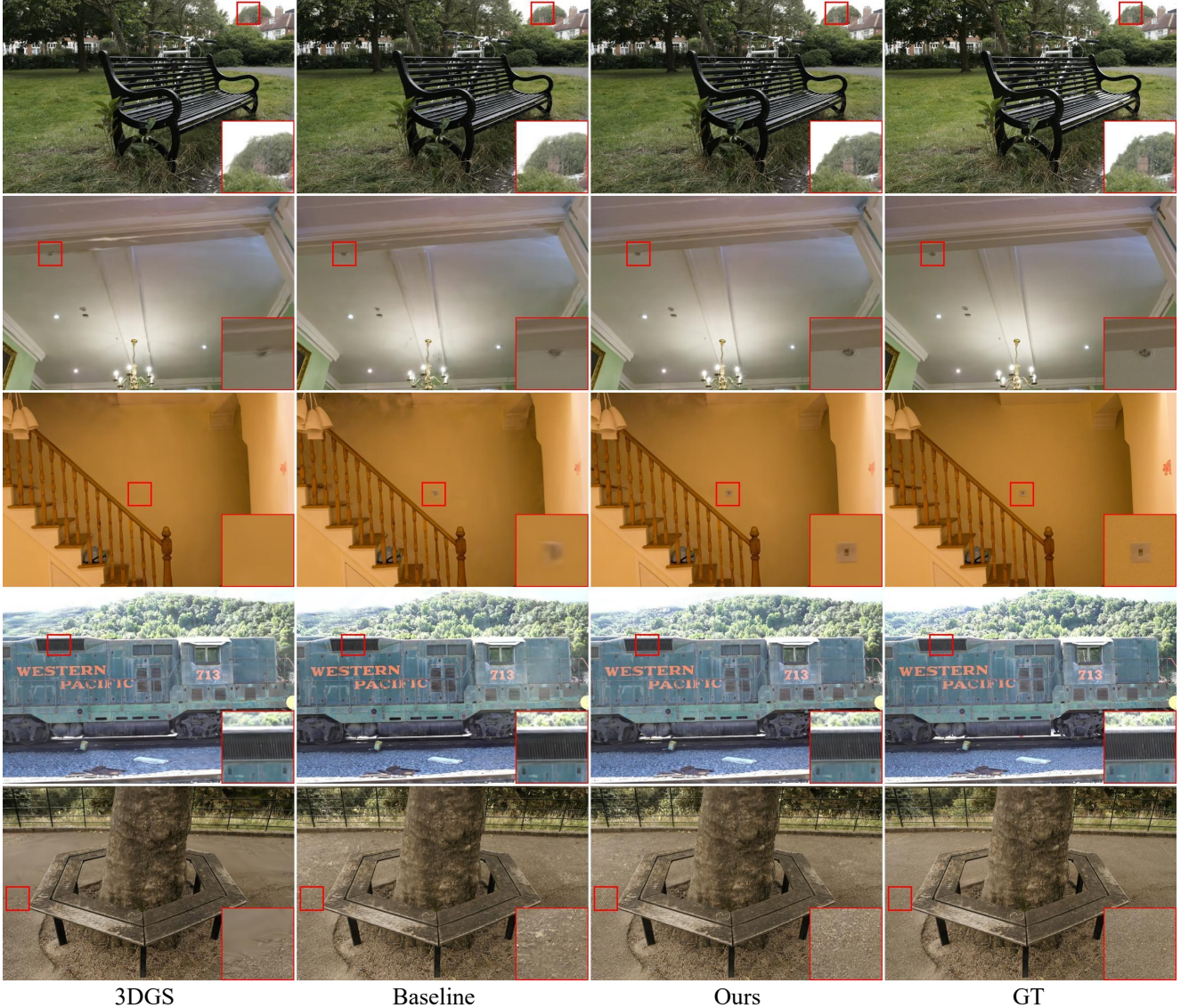


Figure 9. Qualitative analysis of the improvements brought by our approach across multiple scenes. Compared with the baseline and 3DGS, our approach performs better in detailed areas, distant views, and linear regions. Notably, our approach uses significantly fewer Gaussians.

hancements in Gaussian utilization efficiency.

The PSNR scores for each scene are shown in Table 2 and Table 3. Our method shows varying degrees of improvement across both indoor and outdoor scenes. Note that in the flowers and treehill scenes, the insufficient similarity between training and testing viewpoints results in some test viewpoints failing to reconstruct properly, which lowers the average PSNR.

Qualitative Analysis: In Figure 9, we present visual improvements of our method compared to the baseline and 3DGS, with key areas highlighted. Compared to the baseline, our method provides better detail restoration. For example, details such as the wall switch in playroom (line

3), the ceiling light in drjohnson (line 2), and the ground textures in treehill (line 5) are more accurately rendered. This is because our adaptive pruning method allows us to use a smaller splitting threshold to reconstruct these high-frequency details without creating excessive low-opacity Gaussians.

Additionally, our method can also reconstruct some distant regions that are challenging for both 3DGS and the baseline, such as distant streetlights in bicycle (line 1) and trees on the left mountain in train (line 4). These distant features appear infrequently in the training views, making them hard to optimize with multi-view data. By maximizing the use of the parent Gaussian’s shape information, our

Dataset Method—Scene	Mip-NeRF360				Tanks&Temples	
	bicycle	flowers	garden	stump	treehill	train
3DGS	25.21	21.54	27.36	26.54	22.50	21.96
Baseline	25.37	21.37	27.41	26.73	22.10	22.13
Ours-s	25.53	21.72	27.58	27.16	22.53	22.07
Ours-m	25.67	21.73	27.79	27.29	22.33	22.21

Table 2. PSNR scores for outdoor scenes from the Mip-NeRF 360 and Tanks and Temples datasets.

Dataset Method—Scene	Mip-NeRF360				Deep Blending	
	bonsai	counter	kitchen	room	drjohnson	playroom
3DGS	32.24	29.02	31.47	31.45	29.12	30.02
Baseline	32.14	29.10	31.85	31.63	28.93	30.05
Ours-s	32.04	29.00	31.69	31.94	29.55	30.44
Ours-m	32.38	29.16	31.98	32.09	29.63	30.56

Table 3. PSNR scores for indoor scenes from the Mip-NeRF 360 and Deep Blending datasets.

Method—Metric	SSIM [†]	PSNR [†]	LPIPS [‡]	GS Num	FPS	Train
No importance weighting	0.8315	27.76	0.189	2119999	294.12	17m42s
No importance weighting, $\tau_{\text{pos}} = 0.00032$	0.8323	27.78	0.186	2323504	285.26	18m9s
Using a fixed threshold	0.8322	27.78	0.180	2680163	246.52	21m57s
Using original clone & split	0.8232	27.57	0.196	2116298	238.95	19m7s
No pruning with original clone & split	0.8245	27.62	0.189	4772142	117.02	27m1s
No pruning	0.8324	27.80	0.183	3346824	192.92	21m58s
One-time pruning at 15K	0.8327	27.79	0.182	2604252	237.89	19m37s
No opacity reduction after splitting	0.8300	27.68	0.186	2312122	272.19	18m10s
Ours Full	0.8332	27.82	0.184	2261543	278.51	18m4s

Table 4. Results of the ablation study on the Mip-NeRF 360 dataset.

long-axis split enables faster fitting of split Gaussians, successfully recovering these details.

Furthermore, our method excels in handling large, linear areas, such as the ceiling region in drjohnson (line 2) and the vent louver on the train (line 4). This is because we choose the longest axis as the splitting dimension, which allows the elongated Gaussians aligned with straight lines to remain aligned after splitting.

4.4. Ablation Experiments

We tested the impact of each improvement (including the reduction of child Gaussian opacity) on the Mip-NeRF 360 dataset. Unless specified, a uniform splitting threshold of 0.00035 is used, with the detailed results shown in Table 4.

Long-Axis Splits: Compared to the original clone and split, our long-axis split can more efficiently fit scene details (see Figure 10). In the absence of adaptive pruning, long-axis split significantly reduces the number of Gaussians. This is because the long-axis split avoids Gaussian overlap, thereby preventing the generation of a large number of redundant low-opacity Gaussians. The experiment also demonstrates that reducing the opacity of child Gaussians after splitting can improve rendering quality.

Adaptive Pruning: By restricting the minimum opacity of Gaussians, the adaptive pruning operation significantly reduces the number of Gaussians, while even improving rendering quality. The degree of pruning depends on the proportion of low-opacity Gaussians in the scene. Compared to a one-time pruning operation, our approach is more



Clone & Split

Long-axis Split

Figure 10. Qualitative comparison of two densification operations in the treehill scene.

efficient and produces better rendering quality.

Dynamic Thresholding: Dynamic thresholding reduces the number of Gaussians, however, due to a more reasonable densification order and more ample optimization time, it achieves better results in both PSNR and SSIM.

Importance Weighting: By using importance weighting, quality can be improved with only a small increase in the number of Gaussians. Compared to using a smaller splitting threshold, importance weighting is more efficient.

4.5. Limitation

Pruning Gaussians based solely on opacity is effective for most scenes. However, in scenes requiring precise reconstruction of translucent objects, it may impact rendering quality. More complex pruning operations can be used to replace our adaptive pruning, depending on the specific requirements.

5. Conclusion

3DGS generates a large number of low-opacity Gaussians during the densification process, leading to additional overhead. We found that this is primarily due to the limitations of the clone and split operations, which result in overlapping Gaussians after densification. To address this issue, we propose a more efficient long-axis split that effectively eliminates the overlap problem and significantly improves rendering quality. Additionally, we introduce improvements such as adaptive pruning, dynamic thresholding, and importance weighting to further enhance the efficiency of Gaussian utilization. Overall, by improving adaptive density control, our EDC achieves state-of-the-art performance in both rendering quality and speed.

References

- [1] 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. [2](#), [5](#), [6](#)
- [2] Zilong Chen, Feng Wang, Yikai Wang, and Huaping Liu. Text-to-3d using gaussian splatting. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 21401–21412, 2024. [2](#)
- [3] Kai Cheng, Xiaoxiao Long, Kaizhi Yang, Yao Yao, Wei Yin, Yuexin Ma, Wenping Wang, and Xuejin Chen. Gaussianpro: 3d gaussian splatting with progressive propagation. In *Forty-first International Conference on Machine Learning*, 2024. [2](#)
- [4] Guangchi Fang and Bing Wang. Mini-splatting: Representing scenes with a constrained number of gaussians. *arXiv preprint arXiv:2403.14166*, 2024. [2](#)
- [5] Sara Fridovich-Keil, Alex Yu, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. Plenoxels: Radiance fields without neural networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 5501–5510, 2022. [1](#), [6](#)
- [6] 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. [1](#), [6](#)
- [7] Nikhil Keetha, Jay Karhade, Krishna Murthy Jatavallabhula, Gengshan Yang, Sebastian Scherer, Deva Ramanan, and Jonathon Luiten. Splatam: Splat track & map 3d gaussians for dense rgb-d slam. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 21357–21366, 2024. [2](#)
- [8] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Trans. Graph.*, 42(4):139–1, 2023. [1](#), [2](#), [6](#)
- [9] 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. [5](#)
- [10] Muhammed Kocabas, Jen-Hao Rick Chang, James Gabriel, Oncel Tuzel, and Anurag Ranjan. Hugs: Human gaussian splats. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 505–515, 2024. [2](#)
- [11] Byeonghyeon Lee, Howoong Lee, Xiangyu Sun, Usman Ali, and Eunbyung Park. Deblurring 3d gaussian splatting. *arXiv preprint arXiv:2401.00834*, 2024. [2](#)
- [12] Youtian Lin, Zuozhuo Dai, Siyu Zhu, and Yao Yao. Gaussian-flow: 4d reconstruction with dynamic 3d gaussian particle. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 21136–21145, 2024. [2](#)
- [13] Xian Liu, Xiaohang Zhan, Jiaxiang Tang, Ying Shan, Gang Zeng, Dahua Lin, Xihui Liu, and Ziwei Liu. Humangaussian: Text-driven 3d human generation with gaussian splatting. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6646–6657, 2024. [2](#)
- [14] Hidenobu Matsuki, Riku Murai, Paul HJ Kelly, and Andrew J Davison. Gaussian splatting slam. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 18039–18048, 2024. [2](#)
- [15] 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. [1](#)
- [16] Arthur Moreau, Jifei Song, Helisa Dharmo, Richard Shaw, Yiren Zhou, and Eduardo Pérez-Pellitero. Human gaussian splatting: Real-time rendering of animatable avatars. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 788–798, 2024. [2](#)
- [17] 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. [1](#), [6](#)
- [18] 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. [1](#), [2](#)
- [19] Zhijing Shao, Zhaolong Wang, Zhuang Li, Duotun Wang, Xiangru Lin, Yu Zhang, Mingming Fan, and Zeyu Wang. Splattingavatar: Realistic real-time human avatars with mesh-embedded gaussian splatting. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1606–1616, 2024. [2](#)
- [20] 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. [2](#)
- [21] Alexander Vilesov, Pradyumna Chari, and Achuta Kadambi. Cg3d: Compositional generation for text-to-3d via gaussian splatting. *arXiv preprint arXiv:2311.17907*, 2023. [2](#)
- [22] Guanjun Wu, Taoran Yi, Jiemin Fang, Lingxi Xie, Xiaopeng Zhang, Wei Wei, Wenyu Liu, Qi Tian, and Xinggang Wang. 4d gaussian splatting for real-time dynamic scene rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 20310–20320, 2024. [2](#)
- [23] Chi Yan, Delin Qu, Dan Xu, Bin Zhao, Zhigang Wang, Dong Wang, and Xuelong Li. Gs-slam: Dense visual slam with 3d gaussian splatting. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 19595–19604, 2024. [2](#)
- [24] Ziyi Yang, Xinyu Gao, Yangtian Sun, Yihua Huang, Xiaoyang Lyu, Wen Zhou, Shaohui Jiao, Xiaojuan Qi, and Xiaogang Jin. Spec-gaussian: Anisotropic view-dependent appearance for 3d gaussian splatting. *arXiv preprint arXiv:2402.15870*, 2024. [2](#)
- [25] Zongxin Ye, Wenyu Li, Sidun Liu, Peng Qiao, and Yong Dou. Absgs: Recovering fine details in 3d gaussian splatting. In *Proceedings of the 32nd ACM International Conference on Multimedia*, pages 1053–1061, 2024. [2](#), [5](#), [6](#)

- [26] Taoran Yi, Jiemin Fang, Guanjun Wu, Lingxi Xie, Xiaopeng Zhang, Wenyu Liu, Qi Tian, and Xinggang Wang. Gaussian-dreamer: Fast generation from text to 3d gaussian splatting with point cloud priors. *arXiv preprint arXiv:2310.08529*, 2023. [2](#)
- [27] Zehao Yu, Anpei Chen, Binbin Huang, Torsten Sattler, and Andreas Geiger. Mip-splatting: Alias-free 3d gaussian splatting. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 19447–19456, 2024. [2](#)
- [28] Zehao Yu, Torsten Sattler, and Andreas Geiger. Gaussian opacity fields: Efficient and compact surface reconstruction in unbounded scenes. *arXiv preprint arXiv:2404.10772*, 2024. [2](#), [5](#), [6](#)
- [29] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 586–595, 2018. [6](#)
- [30] Zheng Zhang, Wenbo Hu, Yixing Lao, Tong He, and Hengshuang Zhao. Pixel-gs: Density control with pixel-aware gradient for 3d gaussian splatting. *arXiv preprint arXiv:2403.15530*, 2024. [2](#)
- [31] Xiaoyu Zhou, Zhiwei Lin, Xiaojun Shan, Yongtao Wang, Deqing Sun, and Ming-Hsuan Yang. Drivinggaussian: Composite gaussian splatting for surrounding dynamic autonomous driving scenes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 21634–21643, 2024. [2](#)