This CVPR paper is the Open Access version, provided by the Computer Vision Foundation.
Except for this watermark, it is identical to the accepted version;
the final published version of the proceedings is available on IEEE Xplore.

MaskGaussian: Adaptive 3D Gaussian Representation from Probabilistic Masks

Yifei Liu[1,2⋆] Zhihang Zhong[1†] Yifan Zhan[1,3] Sheng Xu[2] Xiao Sun[1,†]
[1]Shanghai AI Laboratory [2]Beihang University [3]The University of Tokyo

Abstract

While 3D Gaussian Splatting (3DGS) has demonstrated remarkable performance in novel view synthesis and real-time rendering, the high memory consumption due to the use of millions of Gaussians limits its practicality. To mitigate this issue, improvements have been made by pruning unnecessary Gaussians, either through a hand-crafted criterion or by using learned masks. However, these methods deterministically remove Gaussians based on a snapshot of the pruning moment, leading to sub-optimized reconstruction performance from a long-term perspective. To address this issue, we introduce MaskGaussian, which models Gaussians as probabilistic entities rather than permanently removing them, and utilize them according to their probability of existence. To achieve this, we propose a ***masked-rasterization*** *technique that enables unused yet probabilistically existing Gaussians to receive gradients, allowing for dynamic assessment of their contribution to the evolving scene and adjustment of their probability of existence. Hence, the importance of Gaussians iteratively changes and the pruned Gaussians are selected diversely. Extensive experiments demonstrate the superiority of the proposed method in achieving better rendering quality with fewer Gaussians than previous pruning methods, pruning over 60% of Gaussians on average with only a 0.02 PSNR decline. Our code can be found at:* https://github.com/kaikai23/MaskGaussian

1. Introduction

Novel view synthesis (NVS) aims to generate photorealistic images of a 3D scene from unobserved views, and has emerged as a crucial area in computer vision and graphics. Central to the main progress is the Neural Radiance Field (NeRF) [24], a method that utilizes multi-layer perceptron (MLP) to represent 3D scenes as continuous volumetric functions, enabling high-fidelity image generation from a collection of 2D images. Despite its impressive results, NeRF's computational demands, particularly the need for expensive ray point sampling and MLP inferences, make it less practical for scenarios where real-time rendering is critical, such as virtual and augmented reality.

In this context, 3D Gaussian Splatting (3DGS) [14] introduces an explicit Gaussian unit-based representation, achieving both photorealistic and real-time rendering through highly parallel GPU kernels. Although explicit Gaussian units can be splatted [40] efficiently for fast rasterization, the densification and optimization processes tend to generate redundant Gaussians [5, 6, 17, 28], resulting in even millions of points for a single indoor scene. This not only reduces training and rendering speed, which could otherwise be faster, but also leads to significant memory consumption.

Recent methods attempt to address this problem by pruning redundant Gaussian points with two main approaches. The first computes a hand-crafted importance score [5, 6, 28] per Gaussian and removes those below a preset threshold. The calculation of the importance score typically requires a scan of all training images, therefore limiting the pruning to be performed only once or twice during training. The other stream uses a learnable mask [4, 17, 34] and multiplies them by Gaussian attributes to receive a gradient. While this allows for gradual removal of Gaussians with masks, the rendered scene keeps relying on the same subset of Gaussians: if a Gaussian is not removed, it persists present from the beginning until the current iteration; once a Gaussian is removed, it is permanently excluded. Pruning with such deterministic generation of masks does not account for the scene's evolution after pruning, potentially leading to the removal of Gaussians that may appear to contribute little at the current iteration, but are vital and difficult to recover in later stages of training (Fig. 3). This yields suboptimized reconstructions, where finer details or small objects are removed (Fig. 2).

In this work, we instead treat Gaussians as being probabilistically existing and utilize them by sampling based on their probability of existence. By rendering solely the sampled Gaussians, we naturally enable the dynamic evolution of the sampled Gaussians alongside the rendered scene, as both the sampling and rendering processes are differentiable. However, only rendering with sampled Gaussians is

⋆ Work done during his internship at Shanghai AI Laboratory.
† denotes co-corresponding authors.

insufficient, as it prevents unsampled Gaussians from receiving gradients and adjusting their probability of existence. Consequently, as the scene evolves with each iteration, unsampled Gaussians remain unchanged and risk becoming outdated, leading to an inaccurate evaluation of their potential contribution when eventually sampled in later iterations.

Previous methods that prune Gaussians by multiplying masks with scales or opacities [4, 17, 34] fail to address this challenge because setting opacity or scale to zero results in a Gaussian with a zero density value α. This zero α causes the Gaussian to be filtered out before rasterization, hindering it from receiving any gradient updates. These masked Gaussians, as not being able to update their probability of existence, maintain a low probability of being used, and are soon outdated, leading to their early removal when sampled and assessed in subsequent iterations. This results in a drastic decrease in rendering quality (Tab. 7). To address this challenge, we propose *masked-rasterization*, in which masks are applied during the blending process to allow masked Gaussians to participate in the rasterization and receive gradients, enabling them to adjust their probability of existence even when not sampled. Specifically, we apply the mask in two places: transmittance attenuation, and color accumulation. We demonstrate that unsampled Gaussians can receive meaningful gradients, guiding the update based on their virtual contribution.

We present the pipeline of MaskGaussian as follows: we formulate the Gaussian pruning based on the probability of existence and sample a mask for each Gaussian to represent its presence or absence according to this probability. Then, we splat all Gaussians in the standard way without any interference from the mask. Subsequently, both present and absent Gaussians can pass through the α filter and enter masked-rasterization, where masks are applied to the transmittance attenuation and color accumulation. This rasterization produces the rendered images and allows the gradient to backpropagate all the way to the masks, regardless of whether they indicate presence or absence.

To summarize, our contributions are three-fold:

- To make pruning adaptive to the dynamically evolving scene, we model Gaussians as probabilistic entities rather than permanently removing them. To effectively assist the learning of Gaussian existence probabilistically, we find it essential for masked Gaussians to receive gradient updates through the mask. We propose *masked-rasterization* to accomplish this.
- We provide the mathematical derivation and analysis for the mask gradient and open source the CUDA implementation of *masked-rasterization*. Through extensive experiments on various datasets, we show *masked-rasterization* substantially outperforms the approach of multiplying masks with Gaussian attributes, such as opacity and scale, offering a superior alternative for 3DGS mask application in future work.
- Comprehensive experiments on three real-world datasets demonstrate the effectiveness of our pruning framework and each component. We achieve pruning ratio of 62.4%, 67.7% and 75.3%, and rendering speedups of 2.05×, 2.19×, and 3.16× on Mip-NeRF360 [1], Tanks & Temples [15] and Deep Blending [11], while only sacrificing 0.02 PSNR for the first two datasets and even improve the rendering quality on Deep Blending.

2. Related Work

Rendering with Radiance Field. By representing a 3D scene as an implicit radiance field from a specific viewpoint, Neural Radiance Field [24] (NeRF) has brought new vitality to the task of novel view synthesis. Subsequent work has focused on improving the rendering accuracy and speed of the MLP-based radiance field representations, leading to the emergence of hybrid representations based on voxel grids [9, 12, 18, 33, 35], hash tables [25], and triplanes [2, 3, 30]. In contrast to implicit formulation of radiance field in NeRF, recent advance in 3D Gaussian Splatting [14] (3DGS) utilizes explicit Gaussian units to model distributions of the radiance field, achieving high-quality and real-time rendering. However, despite these advantages, 3DGS suffers from redundant memory consumption due to its explicit structure and is more prone to overfitting because of the lack of smoothness bias in neural network.

Efficient Gaussian Splatting. 3D Gaussians [14] are sparse and unorganized, posing a significant challenge in spawning new Gaussians, which result in redundant Gaussian primitives and parameters. To improve the storage efficiency, codebooks [5, 17, 26, 27, 29] have been used to compress the redundancy in Gaussian parameters, and pruning [4–6, 10, 17, 28, 38] has been used for reducing the number of primitives. Besides pure compression, Mini-Splatting [6] and Taming 3DGS [22] aim to improve the densification rules to better generate new Gaussians and avoid the generation of primitive redundancy. In another line of research, Scaffold-GS [20] explores spatial efficiency by introducing anchor points to distribute local 3D Gaussians based on encoded features. HAC [4] further explores the spatial efficiency on top of Scaffold-GS by modeling the context between anchors via a structured hash grid. Our work lies in the field of pruning, which is a pure compression technique, and aims at better utilizing the mask to more effectively remove primitive redundancy, without altering the 3DGS pipeline such as representations (anchors) and the densification scheme.

In Gaussian Pruning, LightGaussian [5] proposes an importance score per Gaussian defined by the sum of its ray contribution counts over all training views, multiplied by

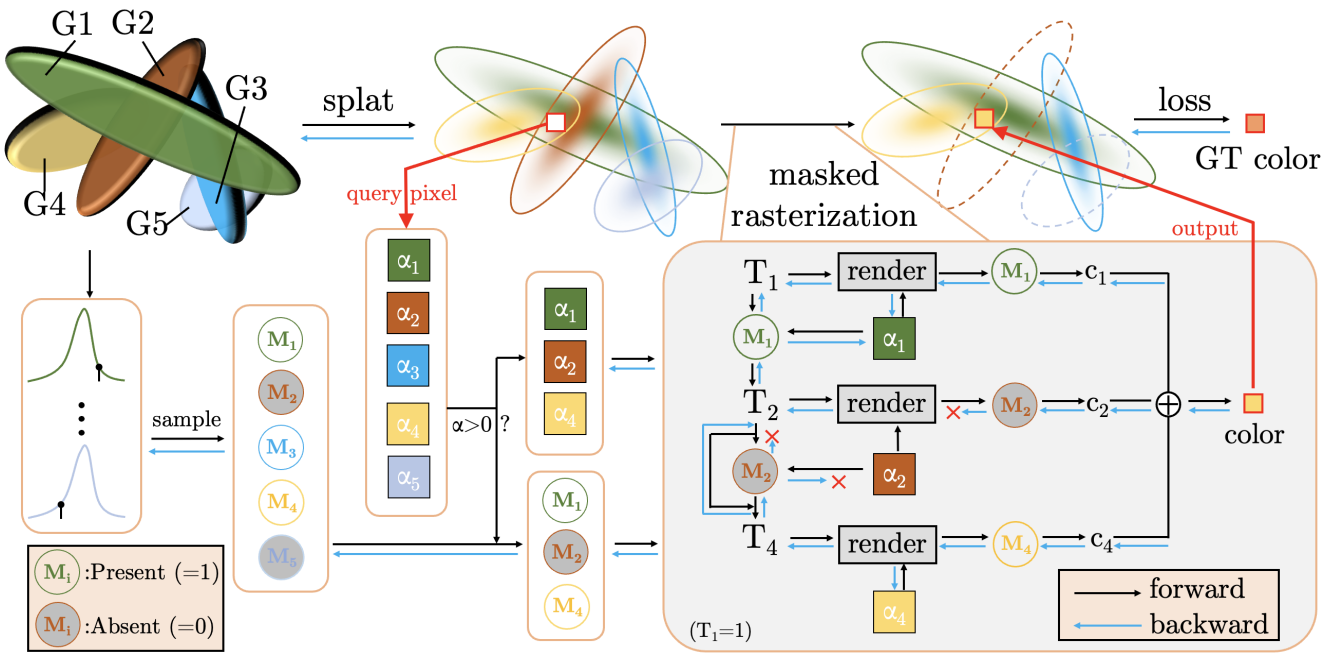

Figure 1. **Overview of MaskGaussian.** We illustrate our pipeline with five Gaussians, G_1 through G_5, where G_2 and G_5 are not sampled and masked. First, all Gaussians are splatted in the standard manner, and differentiable masks are sampled from their existence distributions. For each query pixel, a splat G_i has α_i computed from normal attributes (center, scale, rotation). Splats with zero α_i are filtered out, and the remaining splats and their masks are passed into the masked-rasterization. We apply the masks in two places: the transmittance evolution for T_i and the color rendering for c_i, as detailed in Eq 3 and Eq. 4. A masked splat G_i (e.g., i=2 in this figure) does not receive a gradient for α_i, and thus does not update its normal attributes, but it receives a gradient for mask m_i and updates its existence probability.

the product of its opacity, transmittance and scales. RadSplat [28] scans all training views to find the maximum value of the product of alpha and transmittance for each Gaussian, and uses this as its importance score. Mini-Splatting [6] also calculates the product of alpha and transmittance, but uses the sum instead of max value. Compact3DGS [17] is the first to introduce masks for pruning Gaussians, and multiplies them by Gaussian opacity and scales. LP-3DGS [38] combines masks with importance scores to avoid thresholds in score-based pruning, at the price of rending all training views at each mask training iteration to compute the importance score.

Despite the successful practical usage of aforementioned pruning methods, we observe that they only reflect Gaussian's importance on the current static scene. After the pruning, the Gaussians will move and adjust to make up for the pruned parts, but it is unknown how well they can compensate for and improve the pruned scene, and such a question is beyond the design of previous pruning methods. In this work, we demonstrate that considering this information is feasible by treating pruning as a stochastic sampling, assisted by a *masked-rasterization* technique that can flow gradients to unsampled Gaussians. Our work builds upon Compact3DGS and extends its pruning approach from deterministic to probabilistic, while introducing a novel and more effective method for applying masks.

Dynamic Pruning with Masks Masks have been used to prune redundant components in various fields, including removing image spatial area in CNNs [7], redundant tokens in vision transformers [16, 19, 23, 31, 32], and LLMs [8, 21, 37, 39]. One key to their success is making the mask to receive gradients simultaneously from active and inactive tokens/patches, thus balancing the benefit and cost of mask activation. Inspired by them, we propose *masked-rasterization* to let masked Gaussians receive gradients.

3. Method

3.1. Background

3D Gaussian Splatting [14] uses a set of explicit Gaussian points to represent the 3D scene. A Gaussian $\mathcal{G}^{3D}$ has the following attributes: point center $\mathbf{p} \in \mathbb{R}^3$, opacity $o \in \mathbb{R}^1$, scales $\mathbf{s} \in \mathbb{R}^3$, rotation represented as a quaternion $\mathbf{q} \in \mathbb{R}^4$ and view-dependent spherical harmonics coefficients (SH). When rendering a 2D image from the 3D representation, Gaussians are splatted [40] according to a local affine transformation to be a 2D Gaussian $\mathcal{G}^{2D}(\mathbf{x}) = Splat(\mathcal{G}^{3D}(\mathbf{p}, \mathbf{s}, \mathbf{q})) = e^{-\frac{1}{2}(\mathbf{x}-\overline{\mathbf{p}})^T \overline{\Sigma}(\mathbf{x}-\overline{\mathbf{p}})}$, where $\overline{\mathbf{p}}$ is the projected 2D center, $\overline{\Sigma}$ is the projected 2D covariance ma-

trix, and $\mathbf{x}$ is the evaluated pixel. We omit the pixel subscript x for simplicity and only consider the rendering of one pixel. For this pixel, the splatted density $\alpha_i = o_i \cdot \mathcal{G}_i^{2D}$ is computed for every Gaussian, where i represents the i-th Gaussian in the depth order. Color of Gaussians $\mathbf{c}_i \in \mathbb{R}^3$ is determined by the projection view during splatting. Then the color for pixel $\mathbf{x}$ is rendered from the first to the last Gaussian using the following equation:

$$\mathbf{c}(\mathbf{x}) = \sum_{i=1}^{N} \mathbf{c}_i \cdot \alpha_i \cdot T_i, \tag{1}$$

$$T_{i+1} = (1 - \alpha_i) \cdot T_i, \tag{2}$$

where T_i is the transmittance initialized as $T_1 = 1$.

A Gaussian has theoretically infinite size to affect every pixel, but the influence is negligible for far away pixels. To avoid unnecessary computations in practice, only Gaussians intersecting with the tile (16x16 grid) and with $\alpha > 0$ will participate in the above rendering process.

3.2. Method Overview

We show the overview of our pipeline in Fig. 1. Our method is designed to learn a mask distribution for every Gaussian. Sampling from this mask distribution, we can generate a binary mask to indicate the Gaussian's presence or absence. Then, all Gaussians are splatted without any attribute masked out, and can compute valid α values and normally pass the α-filtering. Gaussians that pass the filtering, including both present and absent ones, will participate in the masked-rasterization together with their masks. In the masked-rasterization, we use the masks in the transmittance evolution and color accumulation such that masked Gaussians will not affect the rendered result and act as pruned, but can receive gradients from zeroed-out color contributions, thereby facilitating a precise evaluation of their contribution to the adapted scene. Specifically, if the present Gaussians effectively compensate for the absence of masked Gaussians, the contributions of masked Gaussians will be small or negative, and their sampling probability will stay low. Otherwise, masked Gaussians will receive positive gradients proportional to their potential contributions, and will be more likely to be sampled in subsequent iterations. Thanks to the masked-rasterization process, masked Gaussians always know their potential influence on the scene, even when the scene's point organization is dynamically changing.

In the example shown in Fig.1, the desired color is closer to the masked Gaussian G_2 than to the color behind it, leading G_2 to increase its probability of existence for use in subsequent iterations, as will be discussed in Sec. 3.4.

3.3. Masked Rasterization: Forward

To prune less important Gaussians, we add masks that can be optimized together with other Gaussian attributes to evaluate the contribution of Gaussians. We take mask generation as a two-category sampling process. Specifically, we assign 2 learnable mask scores for each Gaussian, and apply Gumbel-Softmax[13] to sample one differentiable category out of the 2 scores, denoted as $\mathcal{M}_i \in \{0, 1\}$. By refraining from applying the mask to Gaussian attributes, we retain the integrity of the splatted α, allowing it to bypass filtering and participate fully in rasterization. Then, we integrate masks directly within the rasterization framework as in Eq. 3 and Eq. 4, effectively decoupling Gaussian presence from attributes such as opacity and shape.

$$\mathbf{c}(\mathbf{x}) = \sum_{i=1}^{N} \mathcal{M}_i \cdot \mathbf{c}_i \cdot \alpha_i \cdot T_i, \tag{3}$$

$$T_{i+1} = \mathcal{M}_i \cdot (1 - \alpha_i) \cdot T_i + (1 - \mathcal{M}_i) \cdot T_i. \tag{4}$$

The mask is applied to both color accumulation and transmittance attenuation processes. When $\mathcal{M}_i = 1$, the Gaussian normally contributes to the color and consumes the transmittance based on its α_i. When $\mathcal{M}_i = 0$, the Gaussian's contribution to color is masked out, and its transmittance consumption is skipped. This formulation ensures the forward rasterization result is correct when handling the absence of masked Gaussians. It is important to note that masked Gaussian still engages in the computation of forward pass, and would receive meaningful gradients, as described next.

3.4. Masked Rasterization: Backward

To illustrate the gradient formulation, we define $\mathbf{b}_{i+1}$ as the color rendered behind the i-th Gaussian, i.e., from the (i+1)-th Gaussian to the last Gaussian, as shown in Eq. 5.

$$\mathbf{b}_{i+1} = \sum_{j=i+1}^{N} \mathcal{M}_j \cdot \mathbf{c}_j \cdot \alpha_j \cdot T_j, \tag{5}$$

where $\mathbf{c}_j$, α_j, T_j are the color, splatted density and transmittance of the j-th Gaussian. Then, the mask's gradient is given as follows, and we present the proof in the Appendix:

$$\frac{\partial L}{\partial \mathcal{M}_i} = \alpha_i \cdot T_i \cdot \frac{\partial L}{\partial \mathbf{c(x)}} \cdot (\mathbf{c}_i - \mathbf{b}_{i+1}), \tag{6}$$

where L is the total loss defined later in Eq. 8, $\mathcal{M}_i$ is the binary mask indicating presence or absence, $\mathbf{c(x)}$ is the final output color for pixel $\mathbf{x}$, and $\mathbf{c}_i$ and $\mathbf{b}_{i+1}$ are defined as above.

This gradient equation can be understood by viewing it in two parts. The first part is its weight to the color $\alpha_i \cdot T_i$, which determines the extent of its influence on the color,

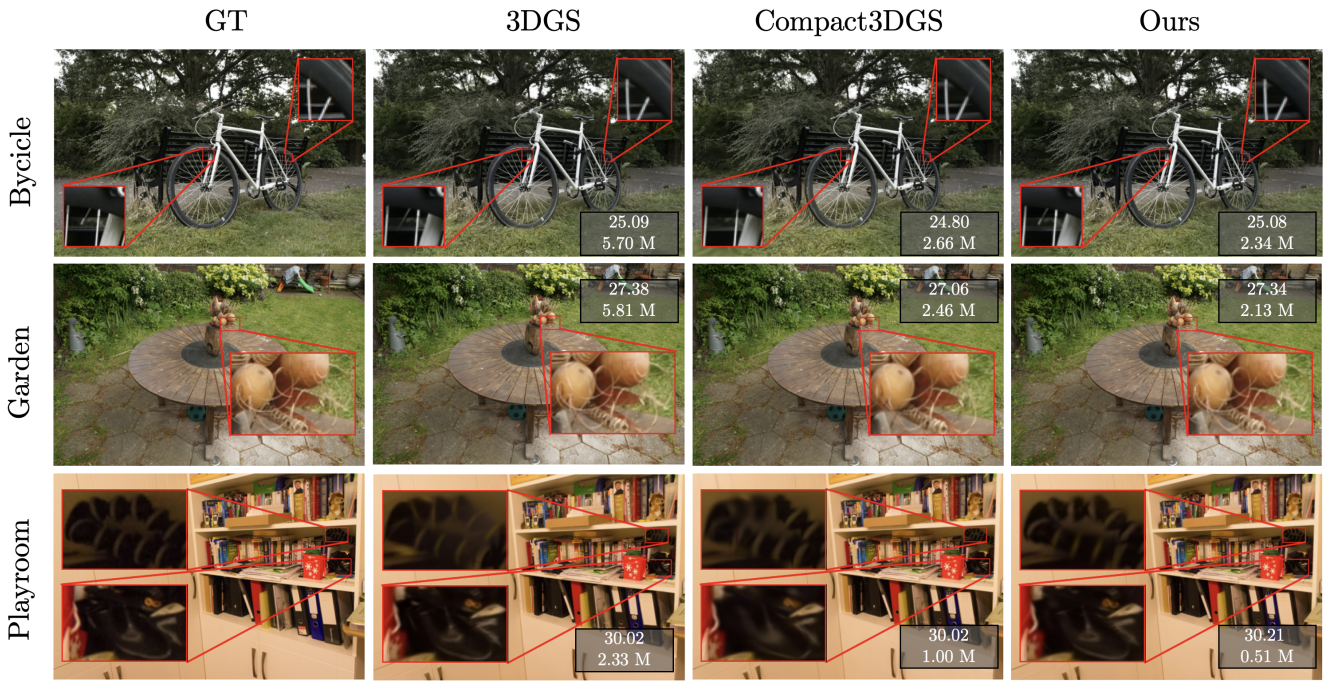

Figure 2. **Qualitative comparison of MaskGaussian with 3DGS and Compact3DGS.** The results show PSNR and the number of Gaussians used (in millions) for each method. While Compact3DGS struggles to accurately represent small and transparent objects, such as the inflation nozzle of the bicycle tire and the penetrating spoke (row 1), withered tendrils of the plant (row 2), and light reflection on the bag (row 3), our method successfully identifies and preserves these fine details.

and therefore also the extent of the gradient. The second part is $\frac{\partial L}{\partial \mathbf{c}(\mathbf{x})} \cdot (\mathbf{c}_i - \mathbf{b}_{i+1})$, where $\frac{\partial L}{\partial \mathbf{c}(\mathbf{x})}$ represents the wanted optimization direction for the color output, and $(\mathbf{c}_i - \mathbf{b}_{i+1})$ represents the benefits of using the i-th Gaussians's color over not using it (thus using $\mathbf{b}_{i+1}$, the color behind it). For example, if the dot product between $\frac{\partial L}{\partial \mathbf{c}(\mathbf{x})}$ and $(\mathbf{c}_i - \mathbf{b}_{i+1})$ is positive, it means using this Gaussian is contributive, and $\mathcal{M}_i$ will receive positive gradient to increase the probability of existence for the i-th Gaussian, even if it is currently not sampled and masked.

Interestingly, our formulation's gradient already encompasses $\alpha_i \cdot T_i$, the importance criterion for score-based prunings [6, 28]. In addition, our formulation captures the influence between the wanted colors and the color of the masked Gaussians, which score-based methods cannot measure and overlook. Compared to approaches that multiply masks with Gaussian scales and opacity [4, 17, 34], we do not inherently bind the gradients of masks with the scales and opacity, which may put fine Gaussians at disadvantage when they receive negative gradients for becoming smaller. And our approach exclusively enables masked Gaussians to receive gradients to update its mask distribution.

3.5. Training and Pruning

We use squared loss to constrain the average number of Gaussians, as defined below, and find it empirically superior to L1 loss:

$$L_m = \left(\frac{1}{N} \sum_{i=1}^{N} \mathcal{M}_i \right)^2, \tag{7}$$

$$L = L_{render} + \lambda_m \cdot L_m. \tag{8}$$

The loss term L_m is then weighted by a balancing hyperparameter λ_m and added to the original rendering loss. We show comparisons of using different λ_m in different phases.

To prune low-probability Gaussians with near-zero sampling likelihood, we sample each Gaussian 10 times and remove those that are never sampled. This pruning procedure is applied at every densification step, and every 1000 iterations after densification.

4. Experiment

4.1. Experimental Settings

Dataset and Metrics. We evaluate our method on three real-world datasets: Mip-NeRF360 [1], which includes five unbounded outdoor and four indoor scenes; two outdoor scenes from the Tanks & Temples dataset [15]; and two indoor scenes from the Deep Blending Dataset [11]. We report metrics including the peak signal-to-noise ratio (PSNR), structural similarity (SSIM), perceptual similarity

Table 1. Quantitative evaluation of our method compared to previous work, computed over three datasets. Results marked with dagger † have been directly adopted from the original paper, all others were obtained in our own experiments. MaskGaussian uses λ_m=0.1 from 19,000 to 20,000 iterations. # GS represents the number of Gaussians in millions. The best, and second best results are highlighted.

Dataset	Mip-NeRF360					Tanks&Temples					Deep Blending				
Metrics	PSNR↑	SSIM↑	LPIPS↓	#GS↓	FPS↑	PSNR↑	SSIM↑	LPIPS↓	#GS↓	FPS↑	PSNR↑	SSIM↑	LPIPS↓	#GS↓	FPS↑
3DGS †	27.21	0.815	0.214	-	-	23.14	0.841	0.183	-	-	29.41	0.903	0.243	-	-
3DGS	27.45	0.811	0.223	3.204	187.8	23.74	0.848	0.176	1.825	254.5	29.53	0.903	0.243	2.815	201.3
Compact3DGS	27.32	0.805	0.233	1.533	281.1	23.61	0.846	0.180	0.960	358.9	29.58	0.903	0.248	1.310	366.2
RadSplat	27.45	0.811	0.223	2.184	247.8	23.61	0.847	0.178	1.053	396.4	29.55	0.903	0.244	1.515	345.3
MaskGaussian-α	27.43	0.811	0.227	1.205	384.7	23.72	0.847	0.181	0.590	558.3	29.69	0.907	0.244	0.694	637.1

Table 2. Ablation study: comparing our method with Compact3DGS. Ours-β has the same training setting as Compact3DGS, and Ours-γ has stronger penalty on the number of Gaussians. Unit: #GS (Million), time (mm:ss), size (MB).

Dataset	Mip-NeRF360						Tanks&Temples						Deep Blending					
Metrics	PSNR↑	SSIM↑	LPIPS↓	#GS↓	time↓	size↓	PSNR↑	SSIM↑	LPIPS↓	#GS↓	time↓	size↓	PSNR↑	SSIM↑	LPIPS↓	#GS↓	time↓	size↓
Compact3DGS	27.32	0.805	0.233	1.533	19:19	362.7	23.61	0.846	0.180	0.960	10:22	228.3	29.58	0.903	0.248	1.310	16:23	311.1
Ours-β	27.44	0.811	0.226	1.520	19:44	362.5	23.66	0.846	0.180	0.740	09:52	175.4	29.76	0.907	0.244	0.913	15:47	215.2
Ours-γ	27.42	0.809	0.228	1.171	17:52	274.2	23.59	0.845	0.183	0.549	08:54	129.8	29.74	0.907	0.247	0.570	14:08	134.9

as measured by LPIPS [36], the number of Gaussians used, and rendering speed in FPS.

Compared Baselines. In addition to the original 3DGS [14], we compare our method with two pruning methods that use hand-crafted importance scores—RadSplat [28] and LightGaussian [5]—and one method that uses learned masks to prune Gaussians, Compact3DGS [17]. For a fair evaluation, we focus solely on these methods' pruning components, excluding unrelated elements such as NeRF initialization, Gaussian attribute vector quantization, and Spherical Harmonics distillation.

Implementation Details. We modify the CUDA rasterization code of 3DGS [14] to integrate our masking into the α-blending process and conduct all experiments including performance evaluation on an NVIDIA RTX 4090. All models are trained for 30,000 iterations, unless stated otherwise. We use 3 settings for hyperparameter λ_m, Ours-α: λ_m=0.1 from 19,000 to 20,000 iterations; Ours-β: λ_m=0.0005 from 0 to 30,000 iterations; Ours-γ: λ_m=0.001 from 0 to 30,000 iterations.

4.2. Experimental Results

Quantitative Results. We summarize the performance of various methods in Tab. 1. Please refer to the Appendix for detailed metrics for each scene. Our MaskGaussian achieves pruning ratios of 62.4%, 67.7%, and 75.3% on Mip-NeRF360 [1], Tanks & Temples [15], and Deep Blending [11], respectively, while nearly preserving image quality metrics. Notably, our method achieves even higher fidelity than the original 3DGS on Deep Blending, which we attribute to the regularization effect of pruning, helping to mitigate overfitting. By efficiently representing the scene through Gaussian pruning, we achieve rendering speedups of 2.05×, 2.19×, and 3.16× across the three datasets.

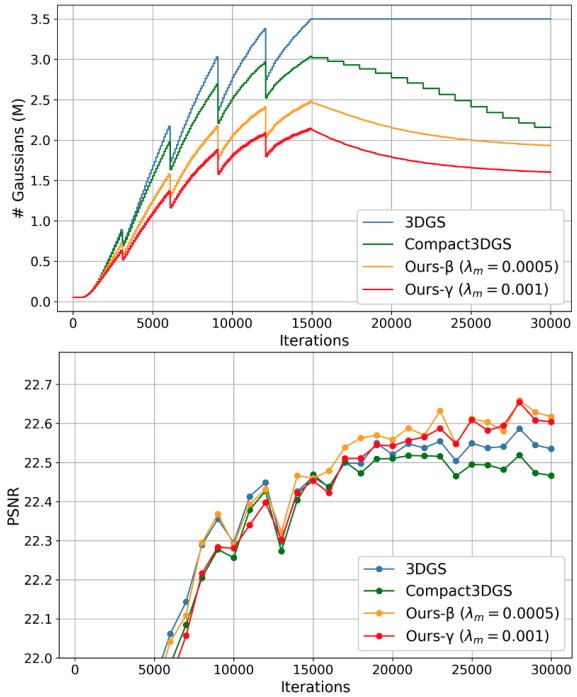

Figure 3. Test PSNR and number of Gaussians used when applying mask over the entire training duration. Ours-β has the same λ_m as Compact3DGS.

Qualitative Results. The qualitative results are shown in Fig. 2. Unlike Compact3DGS, which puts small and transparent Gaussians at a disadvantage, our method effectively

Table 3. Peak GPU Memory Requirement. Our method can prune more Gaussians and preserve equal or better quality, thus saving the most GPU memory.

Scenes	counter	drjohnson	room	truck	flowers
3DGS	5.41GB	9.03GB	5.55GB	5.70GB	7.79GB
RadSplat	4.77GB	8.98GB	4.76GB	5.18GB	7.38GB
Compact3DGS	4.99GB	7.40GB	5.16GB	4.90GB	6.30GB
Ours-β	3.94GB	7.99GB	3.84GB	4.48GB	6.26GB
Ours-γ	3.53GB	6.45GB	3.65GB	4.41GB	5.94GB

identifies and reconstructs them. For instance, in the bicycle scene (row 1 in Fig. 2), Compact3DGS incorrectly places the spoke penetrating the tire and omits the inflation nozzle, while our method accurately captures both. Similarly, in the garden scene (row 2 in Fig. 2), Compact3DGS fails to detect the many withered tendrils, whereas our method correctly identifies their presence.

Post-Training Results. Our method is designed for training from scratch but can also be applied to an existing 3DGS for fine-tuning and Gaussian pruning. We compare our model with LightGaussian [5], finetuning both for 5,000 iterations on a 3DGS trained for 30,000 iterations. As shown in Tab. 5, our model achieves superior performance in the post-training setting, particularly in the LPIPS metric.

Integration with Taming-3DGS. We demonstrate that our method can be seamlessly integrated with state-of-the-art 3DGS optimization frameworks, such as Taming-3DGS [22]. As shown in Tab. 4, using masks to prune Gaussians is a more effective approach to improving efficiency compared to directly controlling the target number of Gaussians in Taming-3DGS.

Table 4. Integration with Taming-3DGS. "Budget" refers to Taming-3DGS's hyperparameter that controls the target number of primitives. Scene: Garden. The difference in training time is negligible compared to the reference 3DGS and is therefore not highlighted.

Target Budget (M)		0.7	2.1	3.5	4.9
Taming-3DGS	PSNR	26.736	27.350	27.481	27.617
	#GS (M)	0.700	2.100	3.500	4.873
	training time	04:47	07:28	10:37	13:47
Taming-3DGS + Compact3DGS	PSNR	26.694	27.318	27.477	27.551
	#GS (M)	0.595	1.623	2.440	3.207
	training time	04:51	07:25	10:06	12:57
Taming-3DGS + Ours	PSNR	26.717	27.391	27.522	27.616
	#GS (M)	0.583	1.581	2.401	3.179
	training time	4:58	7:31	10:14	13:10
3DGS (as ref.)	PSNR: 27.38	#GS (M): 5.81		training time: 32:32	

4.3. Ablation Study

Comparison with Compact3DGS. Compact3DGS [17] multiplies the masks with Gaussian scales and opacities, and applies masks throughout the entire training phase. For comparison, we similarly apply masks over the full training duration, with results presented in Tab. 2. We also show their training progress in Fig. 3.

By using the default mask learning rate and loss hyperparameter λ_m from Compact3DGS, our model-β demonstrates a significant improvement in rendering quality while utilizing slightly fewer Gaussians on Mip-NeRF360 (row 2 in Tab. 2 left). Moreover, we use significantly fewer Gaussians on Tanks & Temples and Deep Blending and also surpass the rendering quality of Compact3DGS by a large margin (row 2 in Tab. 2 middle and right).

To further reveal the effectiveness of our model on Mip-NeRF360, we increase the λ_m to place a stronger penalty on the number of Gaussians and derive model-γ. This adjustment makes the number of used Gaussians of our model distinguishable from Compact3DGS, and demonstrates that we exceed their performance on both rendering quality and Gaussian counts on all three datasets(row 2, 3 in Tab. 2).

Furthermore, as demonstrated in Fig. 3, our method consistently utilizes fewer Gaussians than Compact3DGS throughout the entire training phase, resulting in lower GPU memory consumption as shown in Tab. 3. This reduction is already significant during the densification stages of training, where the number of Gaussians tends to increase at a lower level than Compact3DGS, even as the model's performance, measured by PSNR, remains competitive during this phase and superior afterward.

Effect of Gumbel Softmax. Compact3DGS uses a Straight-Through Estimator (STE) to back-propagate gradients through binary masks and produce deterministic masks using a predefined threshold, while we use Gumbel Softmax to stochastically sample from the mask distribution.

We ablate the effect of Gumbel Softmax by replacing it with the Straight-Through Estimator (STE) and using the pruning threshold from Compact3DGS. Tab. 6 shows the results. Unlike Gumbel Softmax, using STE produces deterministic masks. While a masked Gaussian can still receive gradients and potentially reappear due to masked-rasterization, the scene does not resample Gaussians with mask scores above the threshold and offers no alternative representation for the removed portions. This leads to convergence in a suboptimal state, where Gaussians with lower threshold scores are pruned rapidly, causing an apparent decrease in rendering quality, as shown in row 1 in Tab. 6.

To evaluate the two components' performance at a similar number of primitives, we increase λ_m to encourage fewer Gaussians. As shown in Tab. 6 row 3, Gumbel Softmax with increased λ_m achieves a better trade-off than STE.

Table 5. Post-Training results. The baseline is a normal 3DGS model trained for 30000 iterations, and both LightGaussian and MaskGaussian finetune 5000 iterations on it. #GS is in millions.

Dataset	Mip-NeRF360				Tanks&Temples				Deep Blending			
Metrics	PSNR↑	SSIM↑	LPIPS↓	#GS↓	PSNR↑	SSIM↑	LPIPS↓	#GS↓	PSNR↑	SSIM↑	LPIPS↓	#GS↓
Baseline 3DGS	27.50	0.811	0.224	3.188	23.62	0.847	0.176	1.841	29.54	0.904	0.244	2.820
LightGaussian	27.50	0.809	0.232	1.084	23.75	0.843	0.187	0.626	29.55	0.902	0.250	0.959
MaskGaussian	27.49	0.811	0.228	1.052	23.75	0.847	0.178	0.579	29.71	0.905	0.246	0.667

Table 6. Ablation study of Gumbel Softmax vs.Straight-Through Estimator (STE) on our model. Mask is applied for the full training duration with λ_m=0.0005 and larger λ_m=0.001. #GS in millions.

Dataset	Mip-NeRF360			
Metrics	PSNR↑	SSIM↑	LPIPS↓	#GS↓
STE	27.30	0.808	0.232	1.026
Gumbel Softmax	27.44	0.811	0.226	1.520
Gumbel Softmax + larger λ_m	27.42	0.809	0.228	1.171

Table 7. Ablation study of masked-rasterization on our model. Mask is applied for the full training duration with λ_m=0.0005, smaller λ_m=0.0001, and larger λ_m=0.001. #GS in millions.

Dataset	Mip-NeRF360			
Metrics	PSNR↑	SSIM↑	LPIPS↓	#GS↓
Mask × opacity + smaller λ_m	27.37	0.808	0.229	1.844
Mask × opacity	27.05	0.801	0.245	0.866
Masked-Raster.	27.44	0.811	0.226	1.520
Masked-Raster. + larger λ_m	27.42	0.809	0.228	1.171

Effect of Masked Rasterization. To ablate the effectiveness of the proposed masked-rasterization, we replace it by multiplying mask with opacity and compare their difference. Tab. 7 demonstrates that directly multiplying the mask with opacity results in a significant reduction in the number of Gaussians, leading to a noticeable deterioration in rendering quality. We explain this as follows: in the sampling scenario, multiplying the mask with opacity prevents gradients from propagating to unsampled Gaussians, causing them to fall behind in the optimization process. As a result, even if these Gaussians are sampled again later, their contribution remains misaligned with those that have already been optimized with the scene, leading to a "death spiral" where they are eventually discarded.

We restore the number of Gaussian points in the mask-opacity multiplication approach by relaxing the penalization parameter λ_m, while applying a stronger penalization with a larger λ_m for masked-rasterization. As shown in rows 1 and row 4 of Tab. 7, masked-rasterization significantly outperforms the alternative approach in terms of both PSNR and the number of points, resulting in more effective pruning, and validating the usefulness of receiving gradients for unsampled Gaussians.

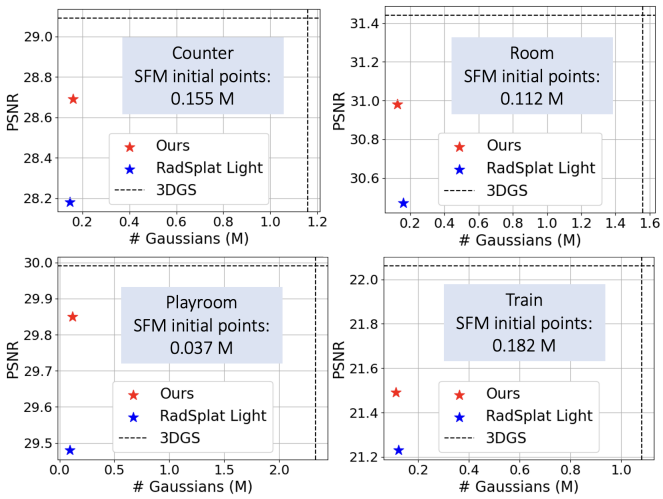

Figure 4. Performance comparison with RadSplat Light in scenarios with a very small number of Gaussians, similar to the initial set of SFM points. λ_m is increased and applied after densification.

Comparison with RadSplat-Light. RadSplat [28] offers a lightweight version by increasing pruning threshold. We increase mask penalty λ_m to match its pruning extent. Fig. 4 shows that even under extreme pruning conditions with a number of Gaussians similar to the initial SFM points, our method more effectively preserves the rendering quality, making it more adaptable to resource-constrained settings.

5. Conclusions

In this paper, we investigate into the problem of pruning 3D Gaussians in a scene by viewing it as a dynamically evolving point organization and modeling pruning as a problem of updating Gaussian's probability of existence. We assist this type of formulation by implementing a *masked-rasterization* technique, which flows gradient through unsampled Gaussians and helps evaluate their virtual contribution to the scene to adjust their existence probability accordingly. Extensive experimental results and analysis demonstrate the effectiveness of this approach and the superiority of applying masks in rasterization process over multiplying them with Gaussian attributes, with 62.4%, 67.7%, and 75.3% Gaussians pruned on Mip-NeRF360 [1], Tanks & Temples [15], and Deep Blending [11] , respectively, with negligible performance degradation.

References

[1] Jonathan T. Barron, Ben Mildenhall, Dor Verbin, Pratul P. Srinivasan, and Peter Hedman. Mip-nerf 360: Unbounded anti-aliased neural radiance fields. *CVPR*, 2022. 2, 5, 6, 8

[2] Eric R Chan, Connor Z Lin, Matthew A Chan, Koki Nagano, Boxiao Pan, Shalini De Mello, Orazio Gallo, Leonidas J Guibas, Jonathan Tremblay, Sameh Khamis, et al. Efficient Geometry-aware 3D Generative Adversarial Networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16123–16133, 2022. 2

[3] Anpei Chen, Zexiang Xu, Andreas Geiger, Jingyi Yu, and Hao Su. Tensorf: Tensorial Radiance Fields. In *European Conference on Computer Vision*, pages 333–350. Springer, 2022. 2

[4] Yihang Chen, Qianyi Wu, Weiyao Lin, Mehrtash Harandi, and Jianfei Cai. Hac: Hash-grid assisted context for 3d gaussian splatting compression. In *European Conference on Computer Vision*, 2024. 1, 2, 5

[5] Zhiwen Fan, Kevin Wang, Kairun Wen, Zehao Zhu, Dejia Xu, and Zhangyang Wang. Lightgaussian: Unbounded 3d gaussian compression with 15x reduction and 200+ fps, 2023. 1, 2, 6, 7

[6] Guangchi Fang and Bing Wang. Mini-splatting: Representing scenes with a constrained number of gaussians. *arXiv preprint arXiv:2403.14166*, 2024. 1, 2, 3, 5

[7] Michael Figurnov, Maxwell D Collins, Yukun Zhu, Li Zhang, Jonathan Huang, Dmitry Vetrov, and Ruslan Salakhutdinov. Spatially adaptive computation time for residual networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1039–1048, 2017. 3

[8] Elias Frantar and Dan Alistarh. SparseGPT: Massive language models can be accurately pruned in one-shot. *arXiv preprint arXiv:2301.00774*, 2023. 3

[9] 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. 2

[10] Alex Hanson, Allen Tu, Vasu Singla, Mayuka Jayawardhana, Matthias Zwicker, and Tom Goldstein. Pup 3d-gs: Principled uncertainty pruning for 3d gaussian splatting. *arXiv*, 2024. 2

[11] Peter Hedman, Julien Philip, True Price, Jan-Michael Frahm, George Drettakis, and Gabriel Brostow. Deep blending for free-viewpoint image-based rendering. 37(6):257:1–257:15, 2018. 2, 5, 6, 8

[12] Tao Hu, Shu Liu, Yilun Chen, Tiancheng Shen, and Jiaya Jia. EfficientNeRF: Efficient Neural Radiance Fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12902–12911, 2022. 2

[13] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016. 4

[14] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics*, 42 (4), 2023. 1, 2, 3, 6

[15] Arno Knapitsch, Jaesik Park, Qian-Yi Zhou, and Vladlen Koltun. Tanks and temples: Benchmarking large-scale scene reconstruction. *ACM Transactions on Graphics*, 36(4), 2017. 2, 5, 6, 8

[16] Zhenglun Kong, Peiyan Dong, Xiaolong Ma, Xin Meng, Wei Niu, Mengshu Sun, Xuan Shen, Geng Yuan, Bin Ren, Hao Tang, et al. Spvit: Enabling faster vision transformers via latency-aware soft token pruning. In *European conference on computer vision*, pages 620–640. Springer, 2022. 3

[17] Joo Chan Lee, Daniel Rho, Xiangyu Sun, Jong Hwan Ko, and Eunbyung Park. Compact 3d gaussian representation for radiance field. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 21719–21728, 2024. 1, 2, 3, 5, 6, 7

[18] Lingjie Liu, Jiatao Gu, Kyaw Zaw Lin, Tat-Seng Chua, and Christian Theobalt. Neural Sparse Voxel Fields. *Advances in Neural Information Processing Systems*, 33:15651–15663, 2020. 2

[19] Yifei Liu, Mathias Gehrig, Nico Messikommer, Marco Cannici, and Davide Scaramuzza. Revisiting token pruning for object detection and instance segmentation. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 2658–2668, 2024. 3

[20] Tao Lu, Mulin Yu, Linning Xu, Yuanbo Xiangli, Limin Wang, Dahua Lin, and Bo Dai. Scaffold-gs: Structured 3d gaussians for view-adaptive rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 20654–20664, 2024. 2

[21] Xinyin Ma, Gongfan Fang, and Xinchao Wang. Llm-pruner: On the structural pruning of large language models. In *Advances in Neural Information Processing Systems*, 2023. 3

[22] Mallick and Goel, Bernhard Kerbl, Francisco Vicente Carrasco, Markus Steinberger, and Fernando De La Torre. Taming 3dgs: High-quality radiance fields with limited resources. In *SIGGRAPH Asia 2024 Conference Papers*, 2024. 2, 7

[23] Lingchen Meng, Hengduo Li, Bor-Chun Chen, Shiyi Lan, Zuxuan Wu, Yu-Gang Jiang, and Ser-Nam Lim. Adavit: Adaptive vision transformers for efficient image recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12309–12318, 2022. 3

[24] 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. In *ECCV*, 2020. 1, 2

[25] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant Neural Graphics Primitives with a Multiresolution Hash Encoding. *ACM Transactions on Graphics (ToG)*, 41(4):1–15, 2022. 2

[26] KL Navaneet, Kossar Pourahmadi Meibodi, Soroush Abbasi Koohpayegani, and Hamed Pirsiavash. Compgs: Smaller and faster gaussian splatting with vector quantization. *ECCV*, 2024. 2

[27] Simon Niedermayr, Josef Stumpfegger, and Rüdiger Westermann. Compressed 3d gaussian splatting for accelerated novel view synthesis. In *Proceedings of the IEEE/CVF*

Conference on Computer Vision and Pattern Recognition (CVPR), pages 10349–10358, 2024. 2

[28] Michael Niemeyer, Fabian Manhardt, Marie-Julie Rakotosaona, Michael Oechsle, Daniel Duckworth, Rama Gosula, Keisuke Tateno, John Bates, Dominik Kaeser, and Federico Tombari. Radsplat: Radiance field-informed gaussian splatting for robust real-time rendering with 900+ fps. *arXiv.org*, 2024. 1, 2, 3, 5, 6, 8

[29] Panagiotis Papantonakis, Georgios Kopanas, Bernhard Kerbl, Alexandre Lanvin, and George Drettakis. Reducing the memory footprint of 3d gaussian splatting. *Proceedings of the ACM on Computer Graphics and Interactive Techniques*, 7(1), 2024. 2

[30] Songyou Peng, Michael Niemeyer, Lars Mescheder, Marc Pollefeys, and Andreas Geiger. Convolutional Occupancy Networks. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part III 16*, pages 523–540. Springer, 2020. 2

[31] Yongming Rao, Wenliang Zhao, Benlin Liu, Jiwen Lu, Jie Zhou, and Cho-Jui Hsieh. Dynamicvit: Efficient vision transformers with dynamic token sparsification. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021. 3

[32] Michael S Ryoo, AJ Piergiovanni, Anurag Arnab, Mostafa Dehghani, and Anelia Angelova. Tokenlearner: What can 8 learned tokens do for images and videos? *arXiv preprint arXiv:2106.11297*, 2021. 3

[33] Cheng Sun, Min Sun, and Hwann-Tzong Chen. Direct Voxel Grid Optimization: Super-fast COnvergence for Radiance Fields Reconstruction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5459–5469, 2022. 2

[34] Henan Wang, Hanxin Zhu, Tianyu He, Runsen Feng, Jiajun Deng, Jiang Bian, and Zhibo Chen. End-to-end rate-distortion optimized 3d gaussian representation. *arXiv preprint arXiv:2406.01597*, 2024. 1, 2, 5

[35] Alex Yu, Ruilong Li, Matthew Tancik, Hao Li, Ren Ng, and Angjoo Kanazawa. Plenoctrees for Real-time Rendering of Neural Radiance Fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5752–5761, 2021. 2

[36] 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

[37] Yuxin Zhang, Mingbao Lin, Zhihang Lin, Yiting Luo, Ke Li, Fei Chao, Yongjian Wu, and Rongrong Ji. Learning best combination for efficient n: M sparsity. *Advances in Neural Information Processing Systems*, 35:941–953, 2022. 3

[38] Zhaoliang Zhang, Tianchen Song, Yongjae Lee, Li Yang, Cheng Peng, Rama Chellappa, and Deliang Fan. LP-3DGS: Learning to prune 3d gaussian splatting. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. 2, 3

[39] Aojun Zhou, Yukun Ma, Junnan Zhu, Jianbo Liu, Zhijie Zhang, Kun Yuan, Wenxiu Sun, and Hongsheng Li. Learning n: m fine-grained structured sparse neural networks from scratch. *arXiv preprint arXiv:2102.04010*, 2021. 3

[40] M. Zwicker, H. Pfister, J. van Baar, and M. Gross. Ewa volume splatting. In *Proceedings Visualization, 2001. VIS '01.*, pages 29–538, 2001. 1, 3