Learning Differentiable Hierarchies in 3D Gaussian Splatting

Youqi Pan[1] Wugen Zhou[1] Hongbin Zha[1, 2]

[1] State Key Laboratory of GAI, School of IST, Peking University
[2] School of Artificial Intelligence and Computer Science, Anqing Normal University

panyouqi@stu.pku.edu.cn zhouwugen@pku.edu.cn zha@cis.pku.edu.cn

Abstract

Although 3D Gaussian Splatting (3DGS) has achieved impressive performance in real-time rendering, its unordered Gaussians make level-of-detail (LoD) construction and model compression highly challenging, limiting its applicability in customized scenarios. In this work, we propose a learning-based Gaussian hierarchy representation that ranks Gaussians by their contribution to the scene, enabling flexible LoD representations across arbitrary Gaussian counts. We first introduce a unified, continuous formulation and metric for Gaussian hierarchy. Then, we introduce a hierarchy-based modulated rendering method built upon a Differentiable Decreasing Step Function, which enables efficient hierarchy learning while maintaining approximately equivalent rendering. Moreover, we develop a PDF-Guided Active-Region Sampling strategy that encourages the learned hierarchy to become widely distributed within its value range. Our method requires no additional training stages and produces Gaussian hierarchies within training time comparable to classical 3DGS. Experiments on multiple datasets show that our approach achieves performance comparable to or surpassing state-of-the-art methods in both LoD rendering and model pruning.

1. Introduction

3D Gaussian Splatting (3DGS) [8], as an emerging scene representation method, has gained increasing popularity due to its strong novel-view synthesis capability and real-time rendering performance. In many applications, customized scene representations are required — for example, low-power devices demand compact models for real-time rendering, while VR/AR systems require foveated rendering that dynamically adjusts spatial resolution according to human visual characteristics.

However, 3DGS-based representations often contain millions of Gaussians, resulting in massive storage overhead. Moreover, the unordered and unstructured nature of Gaussians makes it highly challenging to perform customized model pruning or adaptation. To meet the varying demands of different tasks in terms of model size and rendering fidelity, it is crucial to incorporate hierarchical information into the originally unstructured representation.

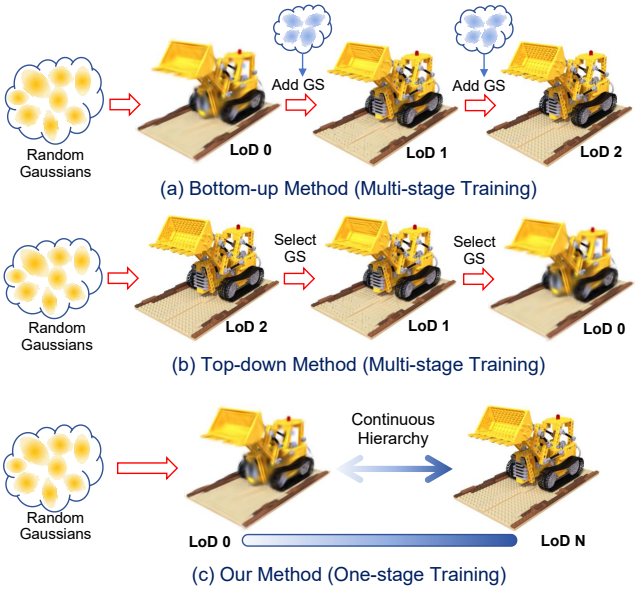

Figure 1. Frameworks of different hierarchical 3DGS methods. (a) Bottom-up methods start from lower LoD and gradually add more Gaussians to construct higher LoDs. (b) Top-down methods begin with the highest LoD and progressively select Gaussian subsets to build lower LoDs. (c) Our method, based on continuous hierarchy learning, constructs all LoD levels within a single training stage.

Existing approaches typically rely on manually designed data structures [9, 12, 26], spatial hierarchies [14, 15, 24, 30], or implicit neural networks [17, 27] to represent scene hierarchy, achieving notable results in Level-of-Detail (LoD) rendering. However, these methods modify the original 3DGS formulation, making them incompatible with the standard representation and training pipeline.

A straightforward idea is to divide Gaussians into several levels and train each level separately, thereby achiev-

ing LoD representation within the classical 3DGS framework. As shown in Fig. 1(a), bottom-up methods [26, 28] first train basic-level Gaussians and then incrementally add higher-level ones to construct a hierarchical structure. In contrast, Fig. 1(b) shows top-down methods [19], which first train all Gaussians and then refine selected subsets. However, these methods rely on manually predefined hierarchical partitions and require separate training for each stage, which limits hierarchical scalability, degrades overall rendering quality, and significantly increases training time compared with standard 3DGS.

In this work, we propose a learning-based Gaussian hierarchy representation, as illustrated in Fig. 1(c). It can be seamlessly integrated into the classical 3DGS framework, enabling the model to preserve high rendering quality using hierarchy-guided Gaussian subsets.

Specifically, we introduce a lightweight hierarchy feature (*H-feat*) for each Gaussian, which is jointly optimized with other Gaussian attributes via gradient backpropagation. From this feature, we derive a continuous, bounded scalar (*H-exp*) that reflects each Gaussian's hierarchy.

To facilitate effective learning of $\mathbf{H}^{\text{feat}}$ and improve rendering quality across hierarchy levels, we propose a training-time rendering strategy based on the Differentiable Decreasing Step Function (DDSF Rendering). It partitions Gaussians into three regions: saturated, active, and cutoff. Gaussians in the saturated region contribute almost fully to rendering, those in the cutoff region contribute negligibly, and those in the active region are adaptively modulated. Within the active region, rendering errors produce amplified gradients on H^{exp}, effectively separating Gaussians according to their contribution. Furthermore, we design a PDF-Guided Active-Region Sampling strategy that selects active regions based on the probability density of H^{exp}. This strategy disperses H^{exp} values and promotes a more uniform distribution across the hierarchy range.

Experiments on multiple datasets demonstrate that our method achieves effective hierarchical representation of 3DGS and delivers near– or even state-of-the-art performance in both Level-of-Detail rendering and model pruning, with only a marginal increase in training time compared to classical 3DGS.

Our contributions can be summarized as follows:

- We propose a learning-based Gaussian hierarchy representation that integrates seamlessly into the classical 3DGS framework, enabling hierarchical organization of Gaussians for high-quality rendering using hierarchy-guided Gaussian subsets.
- We introduce a differentiable hierarchical rendering strategy for training, using a Differentiable Decreasing Step Function and PDF-Guided Active-Region Sampling for effective hierarchy learning.
- Our approach achieves near- or state-of-the-art performance on both Level-of-Detail (LoD) and model pruning tasks, while maintaining a training time comparable to classical 3DGS method.

2. Related Work

2.1. Level-of-Detail in 3D Gaussian Splatting

3D Gaussians, like point clouds, are inherently unstructured. While early LoD methods for point clouds [3, 25, 29] offer some inspiration, the larger number of parameters and higher expressiveness of 3D Gaussians make LoD construction in 3DGS particularly challenging.

One way to address the unstructured nature of classical 3DGS is to introduce additional data structures or spatial hierarchies. FLoD [26] and LODGE [12] implement LoD rendering using manually designed structures and hierarchical criteria. Scaffold-GS [17] introduces multi-level encoding via MLPs. Methods like Octree-GS [24] and LOD-GS [27] leverage explicit representations such as octree voxels or triangle meshes to construct multi-resolution Gaussian representations indirectly. However, these approaches require additional structures or MLPs, preventing seamless integration into the classical 3DGS pipeline.

To keep generality, PRoGS [32] introduces a progressive rendering technique, sorting existing scene splats. LapisGS [28] constructs a multi-layer model that incrementally adds detail, supporting seamless switching of progressive rendering resolution. CLOD [19] learns a continuous LoD model during training, allowing rendering with only the first N Gaussians at inference without modifying the Gaussian structures. These approaches retain the classical 3DGS structure while achieving LoD rendering through manually specified hierarchy divisions. However, such partitioning often requires separate training for different LoD, leading to significantly increased training time.

In contrast, our method leverages a learned hierarchy representation to enable high-quality LoD rendering without manual partitioning or multi-stage training.

2.2. Model Pruning for 3D Gaussian Splatting

To achieve Gaussian model compression, existing methods either apply compressed encodings to the Gaussian representation [20, 21, 23] or directly prune redundant Gaussians [6]. One approach for Gaussian pruning is to score the importance of each Gaussian and retain only the important ones. LightGaussian [4] defines the importance score by summing the number of rays a Gaussian contributes to, multiplied by the product of its opacity and scale. RadSplat [22] defines the importance score as the maximum of the product of each Gaussian's alpha and transmittance. Another approach is pruning via mask training. Compact3DGS [13] introduces masks for Gaussian pruning for the first time. LP-3DGS [31] combines masks with importance scores.

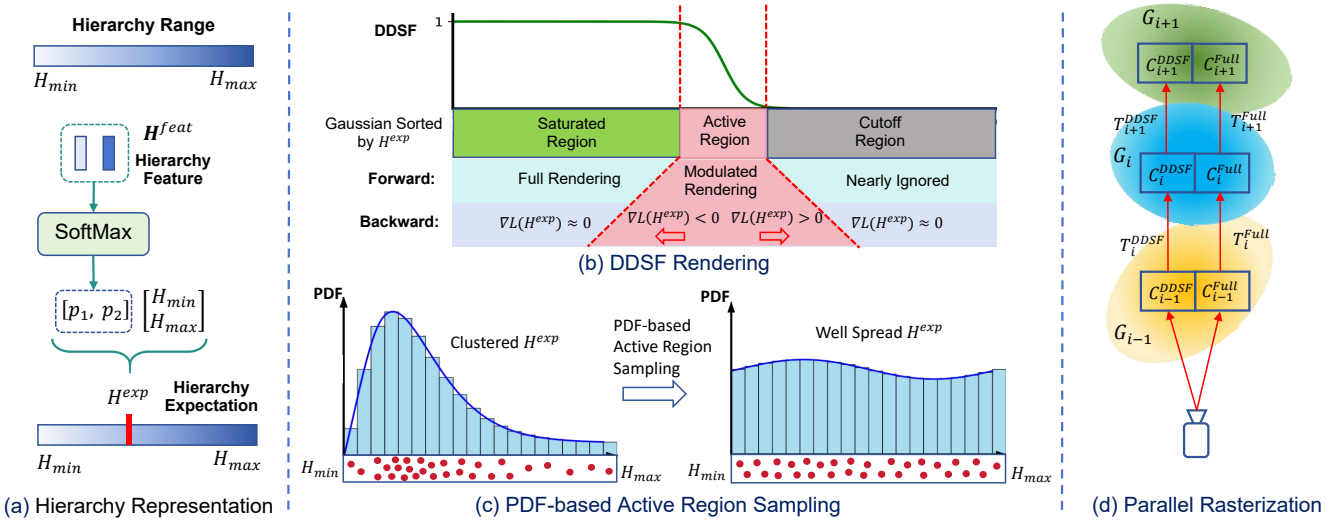

Figure 2. Gaussian Hierarchy Representation and Training Strategy. (a) We construct a continuous and bounded hierarchy range and assign each Gaussian an unconstrained hierarchy feature, from which its hierarchy value H^{exp} is obtained via a soft expectation. (b) During training, we adopt a differentiable decreasing step function for rendering, ensuring that Gaussians within the activation region receive effective gradients for hierarchy optimization. (c) Our PDF-guided active-region sampling strategy enables comprehensive hierarchy learning across all Gaussians and encourages H^{exp} to spread broadly within the predefined range. (d) During training, DDSF rendering shares the same rasterization pipeline with classical rendering, and both pixel colors are computed in parallel.

MaskGaussian [16] learns whether to retain it via differentiable probabilistic sampling during rendering. In addition, some methods improve densification rules [5, 18] or leverage hash grids [2] to enhance spatial efficiency.

In fact, LoD representation can also be regarded as a form of pruning. Our learning-based hierarchy representation can be easily adapted to the pruning task with minor modifications, achieving competitive results.

3. Method

3.1. Overview

Traditional 3DGS [8, 10] represents a scene using a set of order-independent explicit Gaussian primitives. Each primitive is characterized by basic attributes including position $\mathbf{x}_c \in \mathbb{R}^3$, opacity $\alpha \in [0, 1]$, scale $\mathbf{S} \in \mathbb{R}^{3\times3}$, rotation $\mathbf{R} \in SO(3)$, and spherical harmonic coefficients. On this basis, we introduce an additional property — the hierarchy feature ($\mathbf{H}^{feat} \in \mathbb{R}^{N\times1}$) — which reflects each Gaussian's contribution to scene representation. H-feat does not alter the Gaussian's shape, and thus our Gaussian primitive formulation remains consistent with the classical 3DGS representation, as shown in Eq. 1:

$$G(\mathbf{x}) = \alpha e^{-(\mathbf{x}-\mathbf{x}_c)^\top \mathbf{\Sigma}^{-1}(\mathbf{x}-\mathbf{x}_c)}, \tag{1}$$

The $\mathbf{H}^{feat}$ is transformed into a bounded and continuous hierarchy expectation $H^{exp} \in [H_{min}, H_{max}]$, which indicates the relative importance of each Gaussian — a smaller value corresponds to a higher contribution to the model.

To enable hierarchy learning, we design a rendering method based on the Differentiable Decreasing Step Function (DDSF Rendering) during training. Essentially, DDSF (σ) serves as a modulation function on the Gaussian opacity, determined by the activation center μ and the H^{exp}, i.e. $\sigma(H^{exp}; \mu) \in [0, 1]$.

The rendering color C in pixel x is shown as Eq. 2

$$C(x) = \sum_{i=1}^{N} T_i(m_i\alpha_i)c_i \tag{2}$$

with modulation coefficient $m_i = \sigma(H_i^{exp}; \mu)$ and transparency $T_i = \prod_{j=1}^{i-1}(1 - m_j\alpha_j)$.

Compared with the classical rendering equation [8], our method multiplies each Gaussian splat's opacity α_i by its corresponding m_i coefficient. Due to the step-like yet smooth nature of the function, DDSF Rendering simultaneously approximates hierarchy-based rendering of Gaussian subsets, preserving the original training objective, and provides large gradients for H^{exp} in the active area near the activation center. This effectively facilitates the learning of the $\mathbf{H}^{feat}$.

Since the active region of DDSF has a very narrow bandwidth, not all Gaussians' $\mathbf{H}^{feat}$ can be effectively learned. To address this, we divide the H^{exp} range $[H_{min}, H_{max}]$ into several intervals. At each training step, we randomly sample activation centers within these intervals according to the current H-exp distribution and its probability density function, ensuring that active regions primarily cover high-density areas. This encourages H^{exp} to eventually approach

a uniform distribution across its range. We refer to this approach as PDF-Guided Active-Region Sampling.

During training, our DDSF rendering shares the same rendering pipeline with the classical rendering and is executed in parallel, resulting in only a minor overhead. After training, the Gaussians are sorted in ascending order according to their H^{exp} values, which reflect their relative contribution to the scene. Selecting a subset of Gaussians below a certain H^{exp} threshold can preserve the majority of the scene's visual fidelity, enabling LoD rendering and effective model pruning.

3.2. Gaussian Hierarchy Representation

Compared to discrete, manually defined levels [19, 26, 28] and handcrafted hierarchy structures [9, 27], we predefine a continuous hierarchy range $[H_{\min}, H_{\max}]$. Our goal is to learn a hierarchy value for each Gaussian within this range.

Directly optimizing a value constrained to a fixed interval is undesirable for parameter training, as it may lead to vanishing or unstable gradients. To circumvent this, we introduce a two-dimensional, unconstrained hierarchy feature, $\mathbf{H}^{\text{feat}} = [h_1, h_2]$, which is mapped to $[H_{\min}, H_{\max}]$ via a softmax followed by a weighted expectation, as in Eq. 3:

$$\begin{bmatrix} p_1 \\ p_2 \end{bmatrix} = \text{Softmax}(h_1, h_2), \quad H^{\text{exp}} = \begin{bmatrix} p_1 & p_2 \end{bmatrix} \begin{bmatrix} H_{\min} \\ H_{\max} \end{bmatrix} \tag{3}$$

Here, H^{exp} serves as the continuous, differentiable hierarchy value, while the softmax ensures stable gradient propagation by smoothly controlling the relative contribution of each boundary ($H_{\min}$ and $H_{\max}$), as illustrated in Fig. 2(a).

Note that in our setting, a smaller H^{exp} for a Gaussian indicates higher importance.

3.3. DDSF Rendering

Implementing a differentiable hierarchical 3DGS framework involves two coupled training objectives. First, we aim to optimize the hierarchy of each Gaussian based on its contribution to the rendering. Second, we aim to optimize the rendering of Gaussian subsets according to the learned hierarchy. Manually designing Gaussian importance indices [32] or employing multi-stage training [19, 28] cannot simultaneously satisfy both objectives.

This motivates the need to differentiably modulate the classical rendering process using hierarchy information. On one hand, this ensures that a Gaussian's contribution to the rendered output is reflected in the gradient of the rendering loss with respect to the hierarchy, i.e., H^{exp}. On the other hand, its rendering function approximately equivalent to selecting a subset of Gaussians according to a hierarchy threshold.

To this end, we propose a rendering strategy based on the Differentiable Decreasing Step Function (DDSF, σ), as shown in Fig. 2(b). The DDSF approximates a reversed step function around its activation center μ, smoothly transitioning from 1 to 0 while remaining fully differentiable. Applying DDSF to the hierarchy expectation, $m = \sigma(H^{\text{exp}}; \mu)$, naturally separates Gaussians into three regions: (i) the saturated region, where $H^{\text{exp}} \ll \mu$ and $m \approx 1$; (ii) the active region, a narrow interval around μ where $0 < m < 1$; and (iii) the cutoff region, where $H^{\text{exp}} \gg \mu$ and $m \approx 0$.

We use m as a modulation coefficient, multiplying it with the Gaussian splat's opacity during rendering. In the following, we analyze its effect on hierarchy-guided Gaussian subset rendering and Gaussian hierarchy learning, considering both forward and backward propagation.

Forward propagation. According to Eq. 2, for Gaussians in the *active* region, their contribution to the color of a pixel x is given by $C_i(x) = T_i\,(m_i\alpha_i)\,c_i$, where T_i denotes the accumulated transmittance up to the i-th Gaussian, and $m_i = \sigma(H_i^{\text{exp}}; \mu)$. After passing through the i-th Gaussian, the transmittance is updated as $T_{i+1} = T_i\,(1 - m_i\alpha_i)$.

For Gaussians in the *saturated* region, the contribution reduces to $C_i(x) \approx T_i\,\alpha_i\,c_i, \quad T_{i+1} \approx T_i\,(1-\alpha_i)$, which is consistent with classical rendering. And Gaussians in the *cutoff* region yield $C_i(x) \approx 0$ and $T_{i+1} \approx T_i$, meaning they effectively do not participate in rendering.

Since the active region is narrow, the DDSF-based rendering can be seen as an approximation to rendering only the subset of Gaussians with $H^{\text{exp}} < \mu$.

Backward propagation. After obtaining the gradient of the loss with respect to the pixel color, $\frac{\partial L(x)}{\partial C(x)}$, we can compute the gradient with respect to the Gaussian hierarchy value H_i^{exp} using the chain rule:

$$\frac{\partial L(x)}{\partial H_i^{\text{exp}}} = \frac{\partial L(x)}{\partial C(x)} \cdot \frac{\partial C(x)}{\partial m_i} \cdot \frac{\partial \sigma(H_i^{\text{exp}}; \mu)}{\partial H_i^{\text{exp}}}, \tag{4}$$

Expanding $\frac{\partial C(x)}{\partial m_i}$ based on Eq. 2, we have

$$\frac{\partial C(x)}{\partial m_i} = \alpha_i \left(T_i c_i - \sum_{k>i} \frac{T_k m_k \alpha_k c_k}{1 - m_i \alpha_i} \right), \tag{5}$$

where $k > i$ means Gaussians k that are rendered after the i-th Gaussian along the ray.

We observe that for Gaussians whose H_i^{exp} lies in the saturated or cutoff region, the derivative $\frac{\partial \sigma(H_i^{\text{exp}}; \mu)}{\partial H_i^{\text{exp}}} \approx 0$, indicating that the rendering loss provides almost no gradient signal for updating their hierarchy values. In contrast, for Gaussians located in the narrow active region around μ, the steep transition of $\sigma(\cdot)$ yields significantly larger gradient magnitudes, resulting in substantial amplification of the propagated gradients.

More importantly, for different Gaussians with similar H_i^{exp} values within the active region, their gradients are approximately proportional to $\frac{\partial L(x)}{\partial C(x)} \cdot \frac{\partial C(x)}{\partial m_i}$, as derived in

Eq. 4 and Eq. 5. This term reflects each Gaussian's contribution to the rendered color. When the coefficient is positive, a larger magnitude leads to a stronger negative gradient (since $\frac{\partial \sigma(H_i^{\text{exp}};\mu)}{\partial H_i^{\text{exp}}} < 0$), causing H_i^{exp} to decrease. This indicates that **Gaussians with greater rendering contributions are driven toward lower H^{exp} values**, naturally forming an importance-ordered hierarchy.

To summarize, DDSF rendering approximates hierarchy-based Gaussian subset rendering during the forward pass, ensuring consistent optimization objective. During back-propagation, the loss induces significant gradients on the H^{exp} values within the active region, with magnitudes proportional to each Gaussian's rendering contribution, thereby enabling effective learning of the Gaussian hierarchy.

3.4. PDF-Guided Active-Region Sampling

As discussed above, during training, only Gaussians whose H^{exp} values lie within the active region receive meaningful gradient updates. To ensure comprehensive optimization, the active region should shift across iterations to cover a broad range of hierarchy values.

However, in practice, the H^{exp} distribution within $[H_{\min}, H_{\max}]$ is typically non-uniform. Owing to initialization bias and training dynamics, many Gaussians' H^{exp} cluster around a narrow subrange. Consequently, uniform random selection of the active region can lead to imbalanced learning, where certain Gaussians are over-optimized while others remain under-trained.

To address the issue, we propose to sample the active region according to the probability density function (PDF) of the current H^{exp} distribution, as shown in Fig. 2(c).

Specifically, we uniformly divide the interval $[H_{\min}, H_{\max}]$ into B bins, each with width $\Delta H = (H_{\max} - H_{\min})/B$.

Let $\text{hist}(b)$ denote the number of Gaussians whose hierarchy values H_i^{exp} fall into the b-th bin:

$$\text{hist}(b) = \sum_{i=1}^{N} \mathbf{1}\Big(H_{\min} + (b-1)\Delta H \le H_i^{\text{exp}} < H_{\min} + b\Delta H\Big), \quad b = 1, \ldots, B. \tag{6}$$

The empirical probability density function $p(H)$ is then obtained by normalizing the histogram counts:

$$p(H_b) = \frac{\text{hist}(b)}{\sum_{b'=1}^{B} \text{hist}(b')}. \tag{7}$$

During each training iteration, the activation center μ of DDSF is sampled from this discrete density:

$$\mu \sim \text{Multinomial}(p(H)). \tag{8}$$

Compared to random selection, the PDF-Guided Active-Region Sampling strategy focuses more on regions where H^{exp} values are densely concentrated, ensuring a more effective separation of Gaussian hierarchies within each active region. Over time, this leads to a more balanced optimization, allowing the hierarchy distribution to gradually expand and approach uniformity.

3.5. Training and Implementation Details

Parallel Rasterization. During training, we perform DDSF rendering on hierarchical Gaussian subsets, while simultaneously rendering all Gaussians using the standard method. Both renderings share the same rasterization pipeline and are executed in parallel, as shown in Fig. 2(d).

Specifically, we modify the original CUDA implementation of the rasterization kernel. The thread cluster size is increased from $block_x \times block_y$ to $block_x \times block_y \times 2$, such that each pixel is processed by two threads: one applies the modulation coefficient m to the input Gaussian splat's α (for DDSF rendering), while the other multiplies α by 1 (for full Gaussian rendering). Two images are output for DDSF rendering (I^{DDSF}) and full rendering (I^{full}), respectively.

Since the two renderings share all computation except for the color and transparency, the additional time overhead is minimal, ensuring efficient training.

Loss Function. We follow the classical 3DGS formulation [8, 10], where the rendered image and the ground-truth image from the same viewpoint are supervised using an $\mathcal{L}_1$ and SSIM loss:

$$\mathcal{L}_{\text{ren}} = (1-\lambda)\,\mathcal{L}_1(I_{\text{ren}}, I_{\text{gt}}) + \lambda\,\text{SSIM}(I_{\text{ren}}, I_{\text{gt}}) \tag{9}$$

where $\lambda = 0.2$ in practice. This loss is applied to both DDSF and full renderings.

Besides, we observed that the hierarchy values of all Gaussians tend to collectively drift toward $H_{\min}$ during training. This can be intuitively understood as Gaussians being biased toward higher importance (i.e., smaller H^{exp}), or equivalently, that the probability of a Gaussian contributing positively to rendering (i.e. Eq. 5) is higher than that of contributing negatively. To alleviate this bias, we introduce hierarchy balance loss, as defined in Eq. 10. Its gradient towards H^{exp} is proportional to the distance from $H_{\max}$, thus applying a stronger corrective force as H^{exp} moves closer to $H_{\min}$:

$$\mathcal{L}_{\text{bal}} = w_3 \frac{1}{N} \sum_{i=1}^{N} (H_{\max} - H_i^{\text{exp}})^2, \tag{10}$$

The overall loss is the weighted combination of all terms, as shown in Eq. 11:

$$\mathcal{L} = w_1\,\mathcal{L}_{\text{ren}}^{\text{full}} + w_2\,\mathcal{L}_{\text{ren}}^{\text{DDSF}} + w_3\,\mathcal{L}_{\text{bal}} \tag{11}$$

where $w_1 = 1.0$, $w_2 = 0.01$ and $w_3 = 0.001$.

Gaussian Pruning Implementation. Although hierarchy can be interpreted as a flexible form of pruning, achieving

Table 1. Comparison of LoD rendering across different datasets and splat percentages. TrainT denotes the average training time (min:s).

splats %	Method	Mip-NeRF360				Tanks&Temples				Deep Blending			
		PSNR↑	SSIM↑	LPIPS↓	TrainT↓	PSNR↑	SSIM↑	LPIPS↓	TrainT↓	PSNR↑	SSIM↑	LPIPS↓	TrainT↓
25	3DGS-MCMC [10]	18.18	0.61	0.36	-	15.57	0.57	0.37	-	19.60	0.69	0.44	-
	PRoGS [32]	25.28	0.79	0.19	-	22.19	0.79	0.19	-	25.68	0.83	0.21	-
	LapisGS [28]	28.01	0.86	0.15	-	22.72	0.84	0.17	-	26.33	0.84	0.21	-
	CLOD [19]	28.26	0.87	0.14	-	23.65	0.84	0.15	-	26.86	0.84	0.20	-
	Ours	28.35	0.86	0.13	-	23.64	0.84	0.14	-	26.79	0.85	0.21	-
50	3DGS-MCMC [10]	23.45	0.77	0.21	-	19.96	0.74	0.21	-	24.46	0.79	0.28	-
	PRoGS [32]	28.57	0.86	0.13	-	23.93	0.84	0.13	-	26.99	0.84	0.19	-
	LapisGS [28]	28.06	0.87	0.14	-	23.51	0.85	0.16	-	26.06	0.83	0.21	-
	CLOD [19]	29.51	0.88	0.12	-	24.01	0.85	0.14	-	26.87	0.85	0.20	-
	Ours	29.47	0.89	0.11	-	24.20	0.86	0.12	-	26.92	0.85	0.19	-
75	3DGS-MCMC [10]	27.26	0.85	0.14	-	22.72	0.82	0.14	-	26.33	0.83	0.21	-
	PRoGS [32]	29.65	0.88	0.11	-	24.17	0.86	0.12	-	27.01	0.85	0.19	-
	LapisGS [28]	28.14	0.87	0.14	-	23.76	0.84	0.16	-	26.76	0.85	0.21	-
	CLOD [19]	29.60	0.88	0.12	-	24.11	0.86	0.13	-	26.87	0.85	0.19	-
	Ours	29.68	0.90	0.11	-	24.26	0.87	0.12	-	26.94	0.86	0.19	-
100	3DGS-MCMC [10]	29.73	0.89	0.11	23:53	24.21	0.86	0.12	12:37	27.07	0.85	0.19	20:19
	LapisGS [28]	28.21	0.87	0.13	78:20	24.00	0.85	0.15	37:56	26.87	0.85	0.20	63:34
	CLOD [19]	29.59	0.89	0.12	59:34	24.12	0.86	0.13	32:39	26.86	0.85	0.19	51:44
	Ours	29.74	0.90	0.10	26:41	24.27	0.87	0.12	13:48	27.02	0.86	0.18	21:51

Note: All methods use the same total number of Gaussians. PRoGS is a post-processing ranking of a converged 3DGS-MCMC model; thus the 100% case is identical to 3DGS-MCMC and omitted.

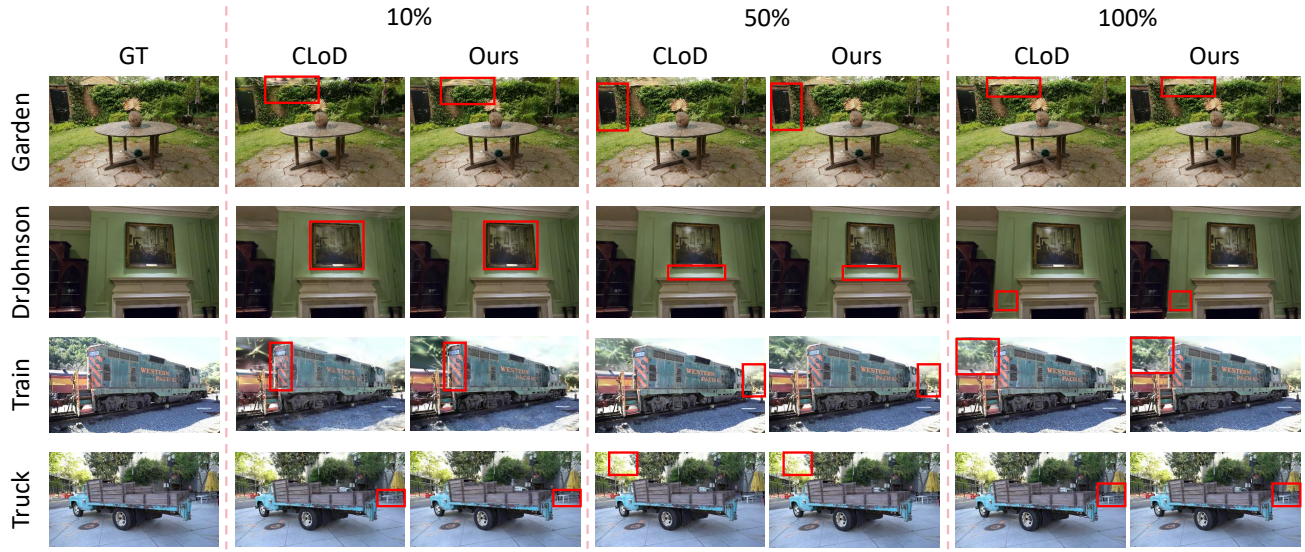

Figure 3. Comparison of rendered images at different splatting ratios (10%, 50%, and 100%).

more aggressive model compression requires adjusting the training objectives in the later stages. Specifically, we attenuate the global hierarchy optimization while emphasizing the rendering optimization of Gaussians within a specified hierarchy threshold. Meanwhile, Gaussians near the threshold continue to be effectively distinguished according to their hierarchy, resulting in a more compact model.

In our settings, after 20,000 training iterations, we select the top 60% of splats as the hierarchy threshold. The activation center of the DDSF is fixed near this threshold, the balance loss weight before the threshold is increased to $w_3 = 0.02$, and the DDSF rendering error weight is set to $w_2 = 0.5$.

Other Implementation Details. In deployment, we set $H_{\min} = 0$ and $H_{\max} = 10$, resulting in a H^{exp} value range of 10. We adopt a sigmoid function $\sigma(\cdot)$ as the decreasing step function and set $\beta = 10$, for which the gradient reaches its maximum value of 2.5 at the activation center. The corresponding activation region ($\sigma \in [0.05, 0.95]$) has a bandwidth of approximately 0.59, which provides a good

balance between sharpness and smoothness.

When computing the probability density function (PDF), we use $B = 50$ bins. The initial learning rate for the hierarchy feature $\mathbf{H}^{\text{feat}}$ is set to 0.001. All other training parameters are kept consistent with 3DGS-MCMC [10].

4. Experiments

4.1. Experiment Setup

We evaluate both LoD rendering and model pruning performance on several public datasets, including *MipNeRF 360* [1], *Tank & Temples* [11], and *Deep Blending* [7]. Our workstation is equipped with an Intel i9-14900K CPU and an NVIDIA RTX 5090 (32 GB) GPU.

4.2. Level-of-Detail Rendering

We perform partial sampling of Gaussians according to the learned hierarchy to evaluate their rendering performance.

We select PRoGS [32] (reordering of pre-trained Gaussian primitives), LapisGS [28] (a bottom-up training strategy), and CLOD [19] (a top-down training strategy) as comparison methods. To ensure a fair comparison, all methods adopt the training strategy of 3DGS-MCMC [10] and use the same total number of Gaussians.

Following the original settings of each method, LapisGS is trained for 30,000 iterations at each sampling level, while CLOD includes an additional 30,000 iterations of random sampling training, resulting in a total of 60,000 iterations. All other methods are trained for 30,000 iterations.

Rendering Performance Comparison. Tab. 1 compares the average rendering performance across three datasets when retaining different proportions of Gaussians. We observe that our method and other clod methods significantly outperform the original 3DGS-MCMC under partial-Gaussian rendering (25%–75%), and in most cases, our method achieves the best results.

Under the 100% rendering setting, our method still surpasses 3DGS-MCMC on two datasets, whereas CLOD exhibits a noticeable performance drop. This can also be observed in Fig. 4, where we compare how PSNR varies with splat percentage across different sequences. It can be seen that the performance of our method steadily improves as the splat ratio increases, whereas CLOD often exhibits oscillations or even degradation in the 60%–100% range.

Qualitative Comparison. Qualitative comparisons between CLOD and ours are presented in Fig. 3. As highlighted by the red boxes, our method preserves more structural details and complex scene information—such as paintings, and distant trees—especially under low splat ratios.

Training Time Comparison. As shown by the *average training time* in Tab. 1, our method increases training time only slightly compared to 3DGS-MCMC (by roughly 10%), while LapisGS and CLOD require multi-stage LoD-training iterations, making their average training time more than twice that of ours.

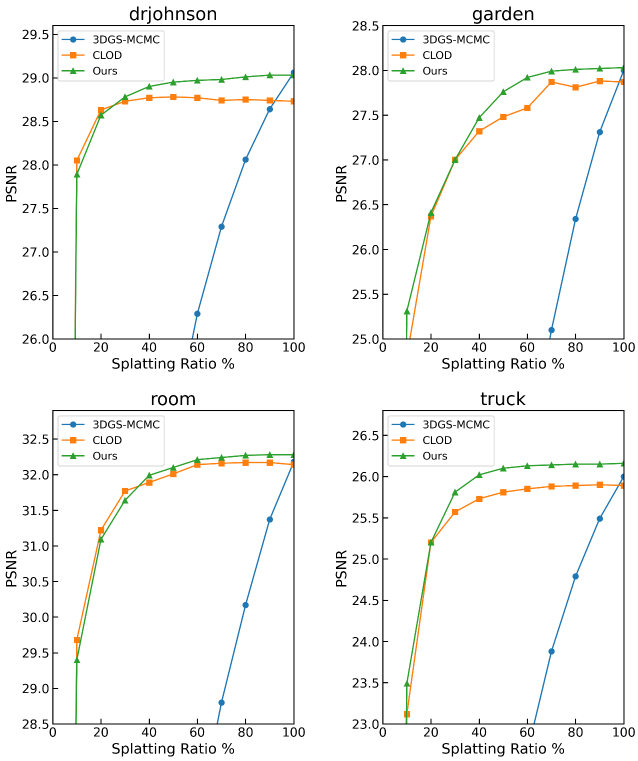

Figure 4. PSNR comparison of different methods at varying splatting ratios.

The experimental results demonstrate that our method excels at achieving LoD with any number of Gaussians, while requiring significantly less training time, particularly CLOD, which also adopts a continuous LoD representation. This can be attributed to two main reasons: (1) In CLOD, the second stage (training with a sampled subset of Gaussians) causes forgetting in the full Gaussian model, leading to degraded performance when rendering 100% of the Gaussians. In contrast, our method uses a single training stage that simultaneously maintains both the full model and hierarchical rendering performance. (2) CLOD fixes the hierarchy according to the insertion order of Gaussians, which cannot be modified, whereas our method learns a flexible, data-driven hierarchy, resulting in superior performance.

4.3. Gaussian Pruning

Our hierarchy representation itself can serve as a basis for pruning. With the adjustments described in Sec. 3.5, more aggressive pruning can be achieved. We compare our method with current state-of-the-art approaches, and the results are shown in Tab. 2.

To fairly demonstrate our contributions, we train models based on both the classic 3DGS [8] and 3DGS-MCMC [10]. The latter allows precise control over the total number of

Table 2. Comparison of Gaussian pruning performance on Mip-NeRF360 across different methods. #GS denotes the number of Gaussians (in millions).

Method	PSNR↑	SSIM↑	LPIPS↓	#GS↓
3DGS [8]	27.45	0.811	0.223	3.204
Compact3DGS [13]	27.32	0.805	0.233	1.533
RadSplat [22]	27.45	0.811	0.223	2.184
MaskGaussian-α [16]	27.43	0.811	0.227	1.205
Ours (3DGS, pruning)	27.37	0.810	0.231	1.534
3DGS-MCMC [10]	29.73	0.891	0.107	3.093
Ours (MCMC, 60% LoD)	29.59	0.890	0.113	1.856
Ours (MCMC, pruning)	29.67	0.895	0.111	1.622

Gaussians, so we additionally report results for both its 60% LoD model and a dedicated pruning model.

It can be observed that under the classical 3DGS framework, our pruning model is highly competitive: even with roughly half the number of Gaussians, the rendering performance decreases only slightly. The pruning model based on 3DGS-MCMC achieves even stronger results, outperforming the corresponding unadjusted LoD model in both rendering quality and total Gaussian count, demonstrating the effectiveness of our pruning strategy.

Table 3. Ablation of sigmoid function on the `truck` sequence. All results are averaged over splatting ratios of 25%, 50%, 75%, and 100%.

Method	PSNR↑	SSIM↑	LPIPS↓
Sigmoid (β=10, Default)	**25.24**	**0.88**	**0.09**
Sigmoid (w/o $\mathcal{L}_{\text{bal}}$)	21.76	0.77	0.22
Sigmoid (β=0.1)	22.49	0.78	0.17
Sigmoid (β=1.0)	24.86	0.86	0.11
Sigmoid (β=100)	24.11	0.84	0.12

Table 4. Ablation of active-region sampling strategy on the `truck` sequence. All results are averaged over splatting ratios of 25%, 50%, 75%, and 100%.

Method	PSNR↑	SSIM↑	LPIPS↓
PDF Sampling (Default)	**25.24**	**0.88**	**0.09**
Random Sampling	24.65	0.86	0.11
Quantile Sampling	24.23	0.85	0.13

4.4. Ablation Study

To validate the effectiveness of our contributions, we conduct an ablation study on the balance loss, the choice of the Sigmoid slope β, and the active-region sampling strategy.

Ablation of Sigmoid coefficient and Balance Loss. We evaluate the rendering results of a variant without the balance loss, as well as variants with the Sigmoid slope β set to 0.1, 1.0, and 100.0, as shown in Tab. 3. The results show that the balance loss is crucial; without it, the overall H^{exp} of Gaussians tends to drift toward H_{min}, which undermines the learning of a correct hierarchy. The choice of β is also critical: if β is too small, the active region becomes excessively large, impairing rendering equivalence and reducing gradient magnitude for updating H^{exp}; if β is too large, H^{exp} changes too abruptly, disrupting hierarchy learning.

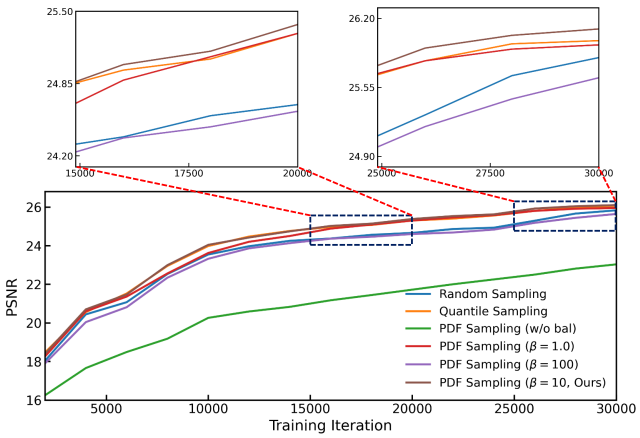

Figure 5. Ablation of PSNR growth during training of 50% splats.

Ablation of Active-Region Sampling Strategy. As shown in Tab. 4, compared to random sampling within the hierarchy range or sampling based on Gaussian H^{exp} quantiles, PDF-based sampling ensures both sampling effectiveness and guidance of the distribution.

We further track the training dynamics of each ablation, as shown in Fig. 5. From the PSNR trajectories over training iterations, we observe that a smaller β and Quantile Sampling exhibit rapid improvement in the mid-training stage but plateau later; in contrast, a larger β and Random Sampling show the opposite behavior. Our adopted strategy maintains a consistent lead with steady growth throughout the entire training process, confirming its effectiveness.

5. Conclusion

In this work, we propose a learning-based hierarchical 3DGS that constructs a differentiable Gaussian hierarchy representation, enabling the training of 3DGS models with hierarchy without manual intervention. We introduce a training and rendering method based on a differentiable decreasing step function, which optimizes the local Gaussian hierarchy while ensuring that rendering using hierarchy-based Gaussian subsets remains approximately equivalent. Additionally, we propose a PDF-based active-region sampling strategy to enable more effective and balanced optimization of the Gaussian hierarchy. Experiments demonstrate that our method achieves LoD and model pruning performance at or beyond the current sota, with only a marginal increase in training time.

Acknowledgement

We gratefully acknowledge the anonymous reviewers and area chairs for their valuable comments and suggestions. This work is supported by the NFS, China (U22A2061), and 230601GP0004.

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. 7, 3

[2] 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*, pages 422–438. Springer, 2024. 3

[3] Carsten Dachsbacher, Christian Vogelgsang, and Marc Stamminger. Sequential point trees. *ACM Transactions on Graphics (TOG)*, 22(3):657–662, 2003. 2

[4] Zhiwen Fan, Kevin Wang, Kairun Wen, Zehao Zhu, Dejia Xu, Zhangyang Wang, et al. Lightgaussian: Unbounded 3d gaussian compression with 15x reduction and 200+ fps. *Advances in neural information processing systems*, 37: 140138–140158, 2024. 2

[5] Guangchi Fang and Bing Wang. Mini-splatting: Representing scenes with a constrained number of gaussians. In *European Conference on Computer Vision*, pages 165–181. Springer, 2024. 3

[6] Alex Hanson, Allen Tu, Vasu Singla, Mayuka Jayawardhana, Matthias Zwicker, and Tom Goldstein. Pup 3d-gs: Principled uncertainty pruning for 3d gaussian splatting. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 5949–5958, 2025. 2

[7] 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. 7

[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, 3, 5, 7, 8

[9] Bernhard Kerbl, Andreas Meuleman, Georgios Kopanas, Michael Wimmer, Alexandre Lanvin, and George Drettakis. A hierarchical 3d gaussian representation for real-time rendering of very large datasets. *ACM Transactions on Graphics (TOG)*, 43(4):1–15, 2024. 1, 4, 3

[10] Shakiba Kheradmand, Daniel Rebain, Gopal Sharma, Weiwei Sun, Yang-Che Tseng, Hossam Isack, Abhishek Kar, Andrea Tagliasacchi, and Kwang Moo Yi. 3d gaussian splatting as markov chain monte carlo. *Advances in Neural Information Processing Systems*, 37:80965–80986, 2024. 3, 5, 6, 7, 8, 2

[11] 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. 7, 3

[12] Jonas Kulhanek, Marie-Julie Rakotosaona, Fabian Manhardt, Christina Tsalicoglou, Michael Niemeyer, Torsten Sattler, Songyou Peng, and Federico Tombari. Lodge: Level-of-detail large-scale gaussian splatting with efficient rendering. *arXiv preprint arXiv:2505.23158*, 2025. 1, 2

[13] 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. 2, 8

[14] Yang Liu, Chuanchen Luo, Lue Fan, Naiyan Wang, Junran Peng, and Zhaoxiang Zhang. Citygaussian: Real-time high-quality large-scale scene rendering with gaussians. In *European Conference on Computer Vision*, pages 265–282. Springer, 2024. 1

[15] Yang Liu, Chuanchen Luo, Zhongkai Mao, Junran Peng, and Zhaoxiang Zhang. Citygaussianv2: Efficient and geometrically accurate reconstruction for large-scale scenes. *arXiv preprint arXiv:2411.00771*, 2024. 1

[16] Yifei Liu, Zhihang Zhong, Yifan Zhan, Sheng Xu, and Xiao Sun. Maskgaussian: Adaptive 3d gaussian representation from probabilistic masks. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 681–690, 2025. 3, 8

[17] 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. 1, 2

[18] Saswat Subhajyoti Mallick, Rahul Goel, Bernhard Kerbl, Markus Steinberger, Francisco Vicente Carrasco, and Fernando De La Torre. Taming 3dgs: High-quality radiance fields with limited resources. In *SIGGRAPH Asia 2024 Conference Papers*, pages 1–11, 2024. 3

[19] Nicholas Milef, Dario Seyb, Todd Keeler, Thu Nguyen-Phuoc, A Božič, Sushant Kondguli, and Carl Marshall. Learning fast 3d gaussian splatting rendering using continuous level of detail. In *Computer Graphics Forum*, page e70069. Wiley Online Library, 2025. 2, 4, 6, 7

[20] KL Navaneet, Kossar Pourahmadi Meibodi, Soroush Abbasi Koohpayegani, and Hamed Pirsiavash. Compgs: Smaller and faster gaussian splatting with vector quantization. In *European Conference on Computer Vision*, pages 330–349. Springer, 2024. 2

[21] 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*, pages 10349–10358, 2024. 2

[22] 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. In *International Conference on 3D Vision 2025*, 2025. 2, 8

[23] 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):1–17, 2024. 2

[24] Kerui Ren, Lihan Jiang, Tao Lu, Mulin Yu, Linning Xu, Zhangkai Ni, and Bo Dai. Octree-gs: Towards consistent real-time rendering with lod-structured 3d gaussians. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–15, 2025. 1, 2, 3

[25] Szymon Rusinkiewicz and Marc Levoy. Qsplat: A multiresolution point rendering system for large meshes. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 343–352, 2000. 2

[26] Yunji Seo, Young Sun Choi, Hyun Seung Son, and Youngjung Uh. Flod: Integrating flexible level of detail into 3d gaussian splatting for customizable rendering. *arXiv preprint arXiv:2408.12894*, 2024. 1, 2, 4

[27] Jianxiong Shen, Yue Qian, and Xiaohang Zhan. Lod-gs: Achieving levels of detail using scalable gaussian soup. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 671–680, 2025. 1, 2, 4

[28] Yuang Shi, Géraldine Morin, Simone Gasparini, and Wei Tsang Ooi. Lapisgs: Layered progressive 3d gaussian splatting for adaptive streaming. In *2025 International Conference on 3D Vision (3DV)*, pages 991–1000. IEEE, 2025. 2, 4, 6, 7

[29] Marc Stamminger and George Drettakis. Interactive sampling and rendering for complex and procedural geometry. In *Eurographics Workshop on Rendering Techniques*, pages 151–162. Springer, 2001. 2

[30] Yufei Wang, Zhihao Li, Lanqing Guo, Wenhan Yang, Alex Kot, and Bihan Wen. ContextGS : Compact 3d gaussian splatting with anchor level context model. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. 1

[31] Zhaoliang Zhang, Tianchen Song, Yongjae Lee, Li Yang, Cheng Peng, Rama Chellappa, and Deliang Fan. Lp-3dgs: Learning to prune 3d gaussian splatting. *Advances in Neural Information Processing Systems*, 37:122434–122457, 2024. 2

[32] Brent Zoomers, Maarten Wijnants, Ivan Molenaers, Joni Vanherck, Jeroen Put, and Nick Michiels. Progs: Progressive rendering of gaussian splats. In *2025 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pages 3118–3127. IEEE, 2025. 2, 4, 6, 7, 1