

GoDe: Gaussians on Demand for Progressive LoD and Scalable Compression

Francesco Di Sario^{1,2*}, Riccardo Renzulli¹, Marco Grangetto¹,
Akihiro Sugimoto², Enzo Tartaglione^{3*}

¹Computer Science, University of Turin, Corso Svizzera, Turin, 10149, Italy.

²National Institute of Informatics, 2 Chome-1-2 Hitotsubashi, Tokyo, 101-8430, Japan.

³LTCI, Télécom Paris, Institut Polytechnique de Paris, 19 Pl. Marguerite Perey, Paris, 91120, France.

*Corresponding author(s). E-mail(s): francesco.disario@unito.it;
enzo.tartaglione@telecom-paris.fr;

Contributing authors: riccardo.renzulli@unito.it; marco.grangetto@unito.it;
sugimoto@nii.ac.jp;

Abstract

Recent progress in compressing explicit radiance field representations, and in particular 3D Gaussian Splatting, has substantially reduced memory consumption while further improving real-time rendering performance. Despite these advances, existing approaches are *inherently single-rate*: each target compression level is obtained through a separately optimized model, resulting in a collection of fixed operating points rather than a truly scalable representation. This limitation restricts practical deployment in scenarios where *memory, bandwidth, or computational budgets vary* across devices or over time. We argue that scalable compression should be formulated as an *intrinsic property of the representation itself*. We show that trained explicit radiance models naturally exhibit a *structured distribution of information*, which can be revealed using generic optimization signals already present during training. In particular, the *aggregated gradient sensitivity* of primitives provides a simple and model-agnostic criterion to organize representation elements from coarse structure to progressively finer refinements. Building on this observation, we introduce GoDe (Gaussians on Demand), a general framework for *scalable compression and progressive level-of-detail control*, instantiated on 3D Gaussian Splatting. Starting from a *single trained model*, GoDe reorganizes Gaussian primitives into a *fixed progressive hierarchy* that supports multiple discrete rate–distortion operating points *without retraining or per-level fine-tuning*. A *single quantization-aware fine-tuning stage* ensures consistent behavior across all levels under low-precision storage. Extensive experiments on standard benchmarks and across multiple 3D Gaussian Splatting backbones demonstrate that GoDe achieves rate–distortion performance comparable to state-of-the-art single-rate methods, while *uniquely enabling true scalable compression and adaptive rendering within a single unified representation*.

Keywords: Radiance Fields, 3D Gaussian Splatting, Scalable Compression, Novel View Synthesis

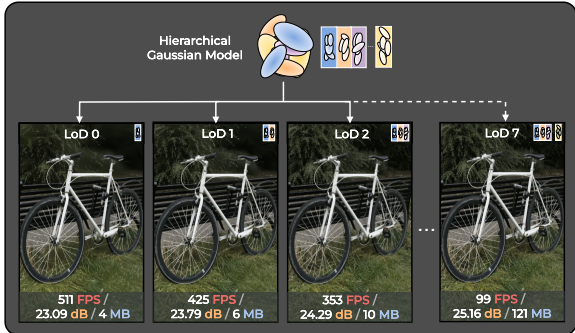


Fig. 1: Our method builds a progressive Gaussian hierarchy, enabling adaptive levels of detail (LoD). Combined with compression, it supports multiple rates (eight in the figure).

1 Introduction

Neural Radiance Fields (NeRFs) have enabled high-fidelity novel view synthesis by representing scenes as continuous volumetric functions optimized from sparse multi-view imagery. Despite their expressive power, NeRF-based approaches typically incur high training and rendering costs, which limit their applicability in real-time and resource-constrained scenarios. Recent advances in explicit radiance field representations, most notably 3D Gaussian Splatting (3DGS), have addressed these limitations by replacing volumetric integration with rasterization-based formulations, enabling real-time rendering with competitive reconstruction quality. As 3DGS models scale to complex, high-resolution scenes; however, their memory footprint grows rapidly, often requiring millions of Gaussian primitives. This growth imposes practical constraints on deployment, including GPU memory limits, storage requirements, and transmission costs. Consequently, a substantial body of recent work has focused on compressing 3DGS models through pruning, quantization, and learned compact representations. These approaches have demonstrated that significant reductions in model size are possible, often with limited impact on reconstruction quality and, in many cases, with further improvements in rendering performance.

Despite their effectiveness, existing compression approaches share a fundamental limitation:

they are inherently single-rate. Each target memory or performance budget is achieved by training or fine-tuning a separate model, producing a set of independent operating points on the rate–distortion curve. While effective in isolation, this formulation does not provide scalability in the strict sense. In realistic deployment scenarios, available memory, bandwidth, and rendering budgets may vary across devices, users, or even dynamically over time. Under such conditions, maintaining, storing, or transmitting multiple specialized models becomes impractical. This observation motivates a different formulation of the problem. Rather than asking how to compress a 3D Gaussian Splatting model to a specific target budget, we ask whether a single trained model can be reorganized to support multiple operating points within a unified representation. In this sense, scalable compression refers to the ability of a single representation to expose multiple rate–distortion configurations without retraining or structural modification. In this work, we show that trained explicit radiance models already exhibit a structured distribution of information that can be exploited for this purpose. Specifically, we observe that the optimization dynamics of 3DGS naturally induce an ordering of Gaussian primitives from coarse, globally important structure to progressively finer, localized refinements. This structure can be revealed using generic optimization signals already present in trained models. In particular, the aggregated gradient sensitivity of each Gaussian, computed across all its parameters, provides a stable and model-agnostic measure of its contribution to reconstruction quality. Based on this insight, we introduce GoDe (**G**aussians **o**n **D**emand), a framework for scalable compression and progressive level-of-detail control in 3D Gaussian Splatting. Starting from a single pretrained model, GoDe reorganizes Gaussian primitives into a fixed progressive hierarchy of discrete levels of detail. Lower levels form compact coarse approximations of the scene, while higher levels incrementally refine geometry and appearance. Crucially, all operating points are derived from the same trained model and do not require retraining or per-level fine-tuning at inference time. To ensure consistent behavior across all levels under low-precision storage, we perform a single quantization-aware fine-tuning stage in

which operating points are randomly sampled during optimization. This joint refinement improves reconstruction quality, particularly at lower levels, and reduces artifacts when switching between configurations. After fine-tuning, the hierarchy is compressed into a scalable bitstream that supports progressive decoding and adaptive rendering, enabling the model to dynamically adjust to changing resource constraints.

In summary, our contributions are as follows:

- *Scalable compression from a single training pipeline.* We identify the lack of scalable compression as a fundamental limitation of existing 3D Gaussian Splatting compression methods and propose a framework that derives multiple discrete compression levels from a single pre-trained model, after a single joint fine-tuning stage.
- *Gradient-induced hierarchical organization.* We introduce an iterative, gradient-informed masking strategy that aggregates gradient information across all Gaussian parameters to construct a stable and effective level-of-detail hierarchy.
- *Model-agnostic design.* Our approach relies exclusively on generic optimization signals and standard training techniques, making it applicable to both explicit and neural 3DGS variants without architectural modifications.
- *Practical deployment.* Through quantization-aware fine-tuning, opacity interpolation, and fast entropy coding, GoDe enables efficient storage, progressive decoding, and adaptive rendering across heterogeneous deployment scenarios.

Through extensive experiments on multiple datasets and 3DGS variants, we demonstrate that GoDe achieves rate-distortion performance comparable to, and often exceeding, state-of-the-art single-rate compression methods, while uniquely enabling scalable compression and flexible deployment within a single unified representation.

2 Related Works

2.1 3D Gaussian Splatting Preliminaries

3D Gaussian Splatting (3DGS) [1] is a real-time radiance field representation that models scenes as collections of volumetric Gaussian primitives. Each Gaussian is parameterized by a 3D position

\mathbf{x} , a covariance matrix Σ (decomposed into scaling and rotation), an opacity o , and spherical harmonics (SH) coefficients encoding view-dependent appearance. During rendering, Gaussians are projected into screen space and rasterized using alpha blending. The contribution of a Gaussian at image position \mathbf{x} is given by

$$\alpha = o \cdot G(\mathbf{x}), \quad (1)$$

with

$$G(\mathbf{x}) = \exp\left(-\frac{1}{2}\mathbf{x}^\top \Sigma^{-1}\mathbf{x}\right). \quad (2)$$

The final image is obtained by combining per-view SH-evaluated colors through alpha compositing. Gaussian parameters are typically initialized from sparse Structure-from-Motion reconstructions and optimized via gradient-based learning using photometric and perceptual losses. Thanks to its explicit formulation and rasterization-based rendering, 3DGS achieves real-time performance with competitive reconstruction quality, making it a strong alternative to NeRF-based models for interactive and large-scale applications.

2.2 Level of Detail in 3D Gaussian Splatting

Level-of-detail (LoD) [2, 3] techniques are a long-established concept in computer graphics, where scene complexity is adapted at render time to meet performance constraints. In the context of 3D Gaussian Splatting, recent works regulate the set of rendered primitives using view-dependent criteria such as screen-space coverage, distance to the camera, or hierarchical spatial structures [4–7]. These approaches are effective for improving rendering efficiency in large or dense scenes, but generally assume a fixed underlying representation. As a result, LoD methods in 3DGS primarily focus on rendering-time selection rather than on compression or storage scalability. While they determine which primitives are rendered for a given view, they do not modify the memory footprint of the model nor provide multiple rate-distortion operating points from a single trained representation.

Resolution-based progressive approaches, such as Lapis-GS [8], extend this paradigm by enabling progressive refinement of the rendered content as a function of target image resolution. These methods facilitate bandwidth-aware streaming and

adaptive rendering, but typically rely on multiple optimization stages or independently trained representations for different resolution levels. Consequently, the training cost scales with the number of supported levels, limiting practical scalability. Our approach is different to existing LoD strategies, including both view-dependent and resolution-based methods. Instead of adapting rendering complexity or resolution at runtime, we reorganize a trained 3DGS model into a progressive, predefined hierarchy that supports scalable compression and progressive transmission. Progressive levels define the available representation budget, while LoD mechanisms may further select subsets of primitives at render time. We demonstrate this orthogonality and complementarity by applying our framework on top of hierarchical 3DGS models such as Octree-GS [6].

2.3 Compression of 3D Gaussian Splatting Models

Reducing the memory footprint of 3D Gaussian Splatting models is a key requirement for practical deployment, particularly in scenarios involving large scenes, limited GPU memory, or transmission over networks. As a result, compression has recently become an active area of research in the context of 3DGS. Inspired by earlier work on radiance field compression [9], many approaches adapt classical techniques such as pruning, quantization, and entropy coding to Gaussian-based representations. A common design pattern consists of post-training Gaussian pruning, followed by parameter quantization and lossless compression [10–13]. These methods aim to remove redundant primitives and reduce parameter precision while preserving reconstruction quality. Other approaches integrate compression directly into the training process. Examples include learning binary or soft masks over Gaussians, compact parameterizations, or low-dimensional encodings [14–18]. In addition, structured representations such as anchor-based Gaussians [19] or hash-grid-assisted encodings [15, 20] reduce storage by sharing parameters or introducing auxiliary neural components.

Statistical pruning strategies have also been explored. PUP-3DGS [21], for instance, proposes to prune Gaussians based on statistical measures derived from opacity and spatial distribution,

without relying on gradient information. Despite their differences, existing compression methods for 3DGS share a common limitation: they are fundamentally designed for a single operating point on the rate–distortion curve. Each compression level corresponds to a separately optimized model, and adapting to different memory or performance constraints typically requires retraining or fine-tuning additional instances. This single-rate formulation limits practical scalability in scenarios involving heterogeneous hardware, varying bandwidth, or dynamic rendering budgets. It is worth noting that many of these works already employ general optimization tools, such as magnitude-based or gradient-based pruning, quantization-aware training, and entropy coding. However, these techniques are typically applied in isolation to produce a single compact representation. In contrast, our work shows that a careful and unified integration of general, model-agnostic techniques—specifically gradient-based importance estimation, iterative masking, and quantization-aware fine-tuning—can not only match or outperform specialized compression strategies, but also enable scalable compression from a single trained model, yielding multiple discrete operating points within a unified framework.

3 Gaussians on Demand

This section describes *GoDe* (Gaussians on Demand), our framework for scalable compression and progressive level-of-detail in 3D Gaussian Splatting. Given a pretrained 3DGS model, our goal is to derive a *single representation* that can operate at multiple rate–distortion operating points, corresponding to different memory budgets, bitrates, or rendering constraints. Crucially, this is achieved within a single training pipeline: no retraining or per-level fine-tuning is required when switching between levels at inference time.

GoDe achieves this by reorganizing the Gaussians of a trained model into a fixed progressive hierarchy, followed by a single joint fine-tuning stage under quantization constraints. Each level of the hierarchy refines the representation of the previous ones by adding a disjoint set of Gaussians. Once constructed and fine-tuned, the hierarchy is compressed into a scalable bitstream that supports progressive decoding and adaptive rendering. We first describe the construction of the

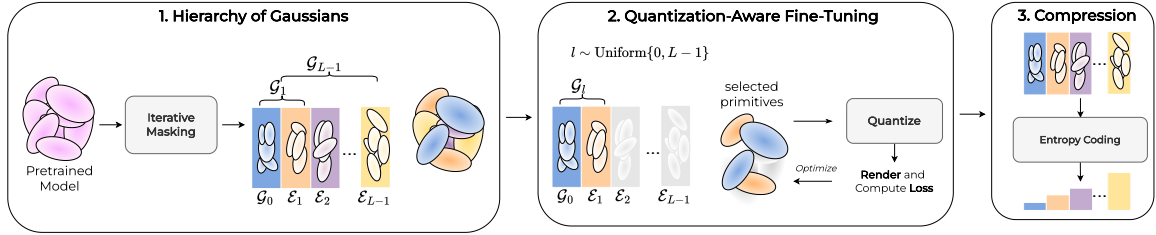


Fig. 2: Overview of GoDe. Starting from a trained 3DGS model, we (1) reorganize Gaussians into a progressive hierarchy using an iterative gradient-informed assignment, (2) perform a single quantization-aware fine-tuning stage that jointly optimizes all levels to ensure consistency under quantization, and (3) compress each level independently to enable scalable decoding and adaptive rendering.

progressive hierarchy (Sec. 3.1), then present the quantization-aware fine-tuning strategy (Sec. 3.2), and finally detail the scalable compression and adaptive rendering mechanism (Sec. 3.3).

3.1 Progressive Hierarchy of Gaussians

Let L denote the total number of levels of detail. Our objective is to reorganize the Gaussians of a trained 3DGS model into a progressive hierarchy that supports scalable compression and level-of-detail control from a single representation. We decompose the full set of Gaussians \mathcal{G} into a base set \mathcal{G}_0 and a sequence of disjoint enhancement sets:

$$\mathcal{G} = (\mathcal{G}_0, \mathcal{E}_1, \dots, \mathcal{E}_{L-1}), \quad (3)$$

where each enhancement set \mathcal{E}_l contains Gaussians that incrementally refine the representation provided by the previous levels. The representation at level l is defined as

$$\mathcal{G}_l = \mathcal{G}_0 \cup \bigcup_{i=1}^l \mathcal{E}_i, \quad (4)$$

with $\mathcal{G}_{L-1} = \mathcal{G}$. All sets are disjoint by construction, ensuring that each Gaussian belongs to exactly one level. Intuitively, lower levels form a compact coarse approximation of the scene, while higher levels progressively add fine-grained details.

Level sizes

We explicitly control the number of Gaussians assigned to each level. Given a minimum number of Gaussians $c_{\min} = |\mathcal{G}_0|$, we define the size of level

l through a geometric progression:

$$|\mathcal{G}_l| = c_{\min} \cdot b^l, \quad (5)$$

where

$$b = \exp\left(\frac{\log(|\mathcal{G}_{L-1}|) - \log(c_{\min})}{L-1}\right). \quad (6)$$

This choice reflects a commonly observed rate-distortion behavior [22]: early Gaussians tend to capture coarse structure and dominant appearance, while additional Gaussians yield progressively smaller refinements. Using a geometric progression provides a simple and effective way to allocate more capacity to coarse levels and progressively less to higher refinement levels. Importantly, this choice is not a requirement of our method. GoDe is not tied to this specific progression and can operate with alternative level size schedules without modifying the hierarchy construction, fine-tuning procedure, or compression pipeline.

Gradient-informed hierarchy construction

To assign Gaussians to levels, we employ an iterative gradient-informed masking procedure. The hierarchy is constructed in a top-down manner, starting from the full model \mathcal{G}_{L-1} and progressively identifying enhancement sets from level $l = L-1$ down to $l = 1$. The remaining Gaussians form the base set \mathcal{G}_0 . At each step l , we accumulate gradients of the reconstruction loss \mathcal{L} with respect to all parameters θ_i of each Gaussian (position, opacity, scaling, rotation, and appearance), while keeping model parameters frozen. For

each Gaussian $i \in \mathcal{G}_l$, we define an importance score

$$s_i = \left\| \frac{\partial \mathcal{L}}{\partial \theta_i} \right\|_2. \quad (7)$$

This aggregated norm provides a first-order estimate of the contribution of a Gaussian to the loss and was found to be more stable than parameter-specific criteria. To construct the enhancement set \mathcal{E}_l , we select

$$k_l = |\mathcal{G}_l| - |\mathcal{G}_{l-1}| \quad (8)$$

Gaussians with the lowest importance scores:

$$\mathcal{E}_l = \text{lowest}_{k_l} \{s_i\}_{i \in \mathcal{G}_l}. \quad (9)$$

These Gaussians are assigned to level l and removed from further consideration. Gradients are reset and the process is repeated on the reduced set. This iterative construction avoids bias introduced by one-shot ranking and yields a stable and well-structured hierarchy.

Efficiency

Although the procedure involves $L-1$ iterations, it is lightweight in practice. Each step requires only a single forward-backward pass over the training views without parameter updates. The total cost of hierarchy construction is negligible compared to full 3DGS training.

3.2 Quantization-Aware Fine-Tuning

Once the hierarchy is constructed, we perform a *single joint fine-tuning stage* to improve reconstruction quality across all levels and to enforce consistency under quantization constraints. This fine-tuning step is executed once and does not depend on the number of levels. We run F fine-tuning iterations. At each iteration, we uniformly sample a level

$$l \sim \text{Uniform}\{0, \dots, L-1\}, \quad (10)$$

instantiate the corresponding representation \mathcal{G}_l (Eq. 4), render using only those Gaussians, and apply a standard 3DGS optimization step to the parameters in \mathcal{G}_l . Random level sampling forces parameters shared across multiple levels—particularly at lower levels—to remain

robust. No additional fine-tuning is required when switching levels at inference time.

Quantization-aware training

During fine-tuning, we simulate quantization for parameters that dominate storage, including appearance coefficients, opacity, scaling, and rotation. Quantization is applied before the forward pass, and gradients are propagated using a straight-through estimator.

Sparsity on appearance coefficients

To further reduce memory usage, we encourage sparsity in appearance coefficients via an ℓ_1 penalty:

$$\mathcal{L}_{\text{app}} = \lambda_{\text{app}} \sum_{i \in \mathcal{G}_l} \|\mathbf{C}_i\|_1. \quad (11)$$

3.3 Scalable Compression and Adaptive Rendering

After fine-tuning, we compress the base level \mathcal{G}_0 and each enhancement set $\{\mathcal{E}_l\}_{l=1}^{L-1}$ independently. This produces a scalable bitstream: decoding \mathcal{G}_0 yields a coarse representation, while progressively appending enhancement layers refines quality. We use Zstandard (ZSTD) with two shared dictionaries trained once on model parameters: one for geometry-related streams and one for appearance streams. Because each enhancement layer is encoded independently and derived from the same trained model, the resulting representation supports true scalable compression rather than a collection of fixed-rate encodings.

Adaptive rendering and smooth level transitions

At render time, the rate-distortion trade-off is controlled by selecting how many levels to decode and render. When budgets tighten, rendering can revert to a lower level without retraining or re-encoding. Switching between adjacent levels may introduce pop-in artifacts due to the sudden activation of enhancement Gaussians. To mitigate this effect, we employ a simple runtime opacity interpolation mechanism: when transitioning from \mathcal{G}_{l-1} to \mathcal{G}_l , Gaussians in the enhancement set \mathcal{E}_l are initially rendered with zero opacity and progressively faded in to their target opacity, while Gaussians belonging to lower levels remain unchanged. This

mechanism is used exclusively at render time and does not affect training, hierarchy construction, or compression.

4 Experiments

4.1 Setup

All experiments were conducted in Python 3.9 using PyTorch 2.1 on a single NVIDIA A40 GPU. Unless otherwise stated, we use the official training and rendering settings of each backbone and keep all evaluation parameters (image resolution, test split, and metric computation) consistent across methods. We set $L = 8$ levels for all configurations, $c_{\min} = 100,000$, and the appearance sparsity weight $\lambda_{\text{app}} = 10^{-2}$ (Eq. 11). The quantization-aware fine-tuning phase consists of $F = 30,000$ iterations. We evaluate GoDe on three representative 3DGS families: (i) *explicit* Gaussians with MCMC-based optimization (MCMC-3DGS [23]), (ii) *neural* Gaussians generated from a compact anchor representation (Scaffold-GS [19]), and (iii) *neural* Gaussians with an explicit dynamic-LoD structure (Octree-GS [6]). This selection allows us to test both model-agnosticity and compatibility with existing LoD mechanisms.

Quantization

For 3DGS/MCMC-3DGS, spherical harmonic coefficients are quantized to 8-bit integers, while positions, opacity, scaling, and rotation are quantized to 16-bit floating-point values. For Scaffold-GS and Octree-GS, we apply the same principle to their parameterization: anchor-point features are quantized to 8-bit integers, and all remaining continuous attributes are quantized to 16-bit floating-point values. For these models, the hierarchy is built by accumulating gradients at the anchor level and progressively masking anchor points, yielding a multi-level representation that can be decoded progressively.

Compression

After fine-tuning, all parameters are compressed using Zstandard (ZSTD), a fast, general-purpose lossless compressor. We use two shared dictionaries (256 kB each): one for geometry-related attributes and one for appearance-related attributes. Dictionaries are learned once and

reused across all hierarchy levels, which is critical for fast decoding and progressive transmission.

4.2 Quantitative Results

We evaluate on the standard 3DGS benchmarks: Mip-NeRF360 [26], Tanks&Temples [27], and DeepBlending [28]. We report PSNR, SSIM, and LPIPS (VGG), together with model size in megabytes (SIZE), primitive count $|\mathcal{G}_{L-1}|$ (W), and average rendering speed over the test set (FPS).

Fairness and reproducibility

To ensure a fair comparison, we retrained and evaluated all methods on the same machine. We followed the hyperparameters and evaluation protocols specified in the original papers when generating rate-distortion curves. We also account for known evaluation discrepancies (e.g., LPIPS normalization) as discussed in [29]. For non-scalable baselines, we report up to three representative operating points, since additional points require training separate models and incur a high computational cost. Specifically, we select operating points by using the public configurations provided by each method (or the closest settings reported in the original paper) and retrain the model for each selected point on our hardware. For GoDe, all L operating points are obtained from a single training pipeline by selecting different progressive levels. The main results are summarized in Tab. 1; additional results are provided in the appendix. In Fig. 3, we plot rate-distortion and PSNR/FPS curves. GoDe covers a wide range of compression rates, from a few megabytes up to approximately sixty megabytes, while preserving competitive reconstruction quality. Among neural-based methods, HAC [15] and Context-GS [30] currently achieve the strongest rate-distortion performance, both built on top of Scaffold-GS [19]. When applying GoDe to Scaffold-GS and Octree-GS, we obtain results that are competitive and often slightly superior on Mip-NeRF360, while HAC and Context-GS remain marginally better on Tanks&Temples and DeepBlending. A similar trend holds for explicit models: when applied to 3DGS, GoDe achieves state-of-the-art rate-distortion curves on Mip-NeRF360. Importantly, these results are obtained without training separate models for each operating point: the full

Table 1: Comparison of our methods against recent methods. All methods have been retrained and tested on the same hardware. For clarity, we present 3 compression levels out of 8 for our configurations, selected to match the compression rate of the other methods. These levels, labeled as *Ours*, are derived from the same model without the need for retraining. Red and yellow respectively indicate the first and the second best result.

	<i>Dataset</i>	<i>Mip-NeRF360</i>						<i>Tanks&Temples</i>						<i>Deep Blending</i>					
		PSNR	SSIM	LPIPS	Size	W	FPS	PSNR	SSIM	LPIPS	Size	W	FPS	PSNR	SSIM	LPIPS	Size	W	FPS
	<i>Compression</i>	↑	↑	↓	[MB]↓	↓	↑	↑	↓	↓	[MB]↓	↓	↑	↑	↓	↓	[MB]↓	↓	↑
Ours 3DGS	high	26.71	0.773	0.339	9.1	242	304	23.22	0.811	0.278	7.2	208	388	29.54	0.897	0.351	7.5	212	489
	medium	27.32	0.804	0.289	20.6	603	213	23.85	0.835	0.239	13.6	438	320	29.64	0.901	0.336	14.1	452	360
	low	27.45	0.812	0.273	31.3	962	174	23.99	0.840	0.226	18.7	640	388	29.65	0.901	0.331	19.3	660	311
Ours Scaffold-GS	high	27.25	0.789	0.318	10.2	129	108	23.86	0.839	0.242	7.4	93	138	30.23	0.906	0.338	6.2	80	165
	medium	27.79	0.810	0.284	19.3	248	83	24.01	0.848	0.224	11.0	141	116	30.33	0.909	0.331	8.4	109	155
	low	27.87	0.813	0.276	36.9	483	69	24.04	0.851	0.218	16.5	215	98	30.33	0.909	0.328	11.3	149	148
Ours Octree-GS	high	26.44	0.761	0.341	10.8	133	113	23.98	0.834	0.281	10.3	127	107	29.91	0.899	0.350	5.1	72	205
	medium	27.55	0.808	0.279	20.5	262	86	24.45	0.859	0.235	18.2	235	78	30.02	0.901	0.344	6.4	91	195
	low	27.81	0.817	0.261	38.5	527	73	24.61	0.868	0.219	31.0	438	60	30.03	0.901	0.342	7.8	116	192
Context- 3DGS	high	27.57	0.808	0.289	12.4	364	66	24.17	0.856	0.215	10.0	222	77	30.10	0.907	0.341	3.4	155	121
	medium	27.74	0.812	0.279	18.3	429	58	24.30	0.856	0.214	10.0	247	78	30.30	0.911	0.332	5.5	167	133
	low	27.74	0.812	0.279	21.6	477	55	24.25	0.855	0.218	10.3	251	77	30.27	0.911	0.329	6.7	181	135
HAC [15]	high	27.17	0.798	0.306	11.5	371	74	24.03	0.843	0.237	7.4	261	82	29.92	0.903	0.250	3.9	180	166
	medium	27.53	0.807	0.290	15.2	425	72	24.05	0.846	0.222	7.9	300	85	29.98	0.904	0.340	4.1	187	173
	low	27.78	0.812	0.277	22.0	501	68	24.37	0.853	0.215	11.1	307	88	30.34	0.909	0.329	6.3	196	166
RDO [18]	high	26.45	0.782	0.322	9.7	862	280	23.19	0.827	0.253	5.5	395	370	29.61	0.903	0.331	7.0	531	335
	medium	26.89	0.796	0.298	15.3	1236	227	23.16	0.833	0.239	8.0	598	307	29.68	0.905	0.323	11.0	891	269
	low	27.05	0.801	0.288	23.3	1860	178	23.32	0.839	0.232	11.9	912	257	29.72	0.906	0.318	18.0	1478	200
Reduced-3DGS [17]	high	27.05	0.807	0.272	22.7	1245	247	23.46	0.840	0.228	10.5	558	384	29.63	0.906	0.318	13.6	804	337
	medium	27.19	0.810	0.267	29.3	1436	234	23.55	0.843	0.223	14.0	656	358	29.70	0.907	0.315	18.3	990	304
	low	27.28	0.813	0.264	45.8	1434	240	23.57	0.844	0.221	20.7	648	360	29.69	0.907	0.314	35.3	988	311
Comp-GS [16]	high	26.40	0.778	0.323	9.1	454	233	23.06	0.816	0.278	6.1	270	334	29.22	0.894	0.369	6.0	311	440
	medium	26.71	0.790	0.305	10.9	470	193	23.32	0.828	0.259	7.3	242	311	29.56	0.899	0.351	7.1	271	315
	low	27.28	0.802	0.283	16.4	485	165	23.62	0.837	0.248	9.8	241	286	29.89	0.904	0.336	10.0	246	254
SOG [24]	high	26.56	0.791	0.291	16.7	2150	119	23.15	0.828	0.277	9.3	1207	216	29.12	0.892	0.351	9.3	800	219
	low	27.08	0.799	0.277	40.3	2176	132	23.56	0.837	0.221	22.8	1242	227	29.26	0.894	0.336	17.7	890	236
EAGLES [14]	-	27.13	0.809	0.278	57.1	1290	156	23.20	0.837	0.241	28.9	651	227	29.75	0.910	0.318	52.4	1192	144
LightGaussian [25]	-	27.24	0.810	0.273	51.0	2197	161	23.55	0.839	0.235	28.5	1211	225	29.41	0.904	0.329	43.2	956	348
Compact-3DGS [20]	-	27.02	0.800	0.287	29.1	1429	108	23.41	0.836	0.238	20.9	842	147	29.76	0.905	0.324	23.8	1053	144

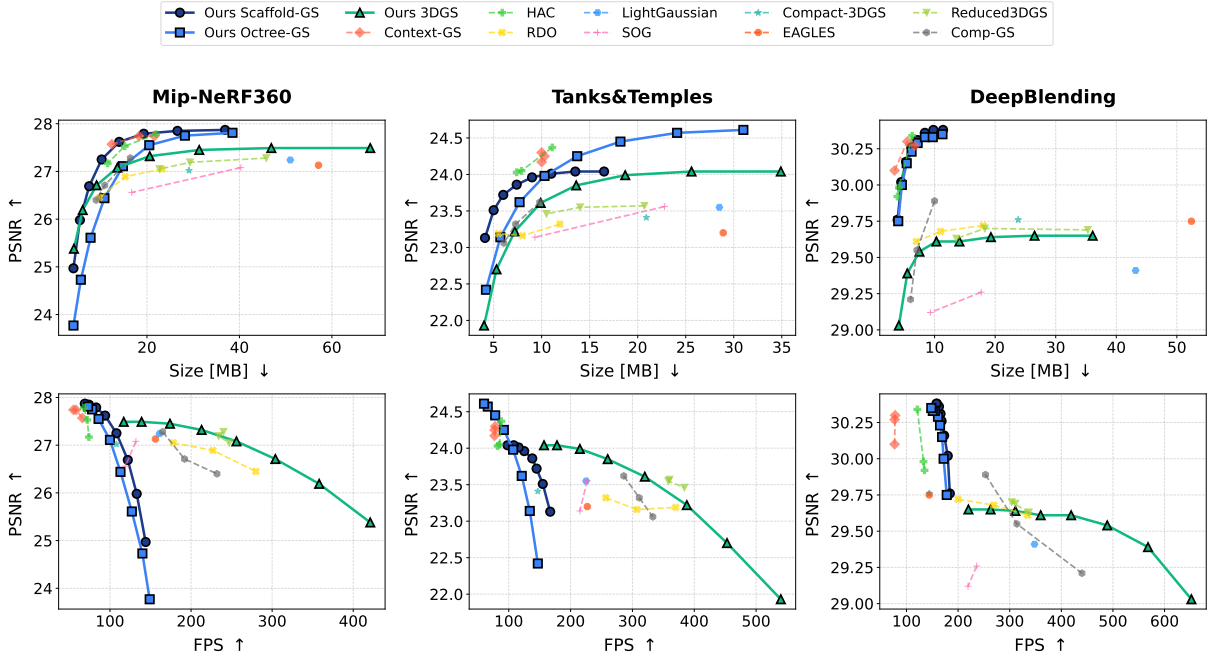


Fig. 3: Rate–distortion (top) and PSNR/FPS (bottom) curves on the three main datasets. GoDe remains competitive across both explicit and neural 3DGS backbones, while providing a single progressive model that supports multiple operating points without per-point retraining.

rate–distortion curve is produced by a single progressive representation, and different compression levels are obtained by selecting the corresponding hierarchy depth.

4.3 Visualizing LoDs and Transitions

Fig. 4 visualizes representative levels across several scenes. Lower LoDs are extremely compact and fast, often exceeding 800 FPS and reaching model sizes as low as 4.1 MB for *drjohnson* (DeepBlending), compared to vanilla 3DGS at 146 FPS and ~ 780 MB. As expected, lower levels lose fine details, particularly in high-frequency regions such as reflections, transparency, and distant background structures. For example, the glass bottle in *kitchen* and the dish reflections in *counter* progressively emerge as higher enhancement layers are added. This behavior is consistent with our hierarchy construction: lower levels preserve coarse structure and dominant appearance, while enhancement layers refine localized geometry and view-dependent details. To further reduce pop-in artifacts, we enable continuous transitions

between adjacent levels by linearly interpolating the opacity of the next enhancement layer. Fig. 5 shows a smooth transition on *Garden* using eight interpolation factors. This interpolation is applied only at render time for visualization and does not affect training, hierarchy construction, or compression.

4.4 Performance

In Tab. 2 we report training, encoding, and decoding times and compare against state-of-the-art baselines. For non-scalable methods, the reported training time corresponds to a single operating point; obtaining L operating points would require training (or fine-tuning) L separate models, thus multiplying the total computational cost. In contrast, GoDe produces all L operating points within a single pipeline: the reported time includes the backbone training as well as the additional 30k quantization-aware fine-tuning iterations used to optimize the progressive hierarchy. From a deployment perspective, the main advantage of GoDe is decoding efficiency. At high compression rates, we



Fig. 4: Visualization of LoDs. For clarity, we show the first three levels and the final one. For each LoD, we report average FPS, PSNR, primitive count W (in thousands), and model size. From top to bottom: *kitchen*, *counter*, *drjohnson*, and *truck*.

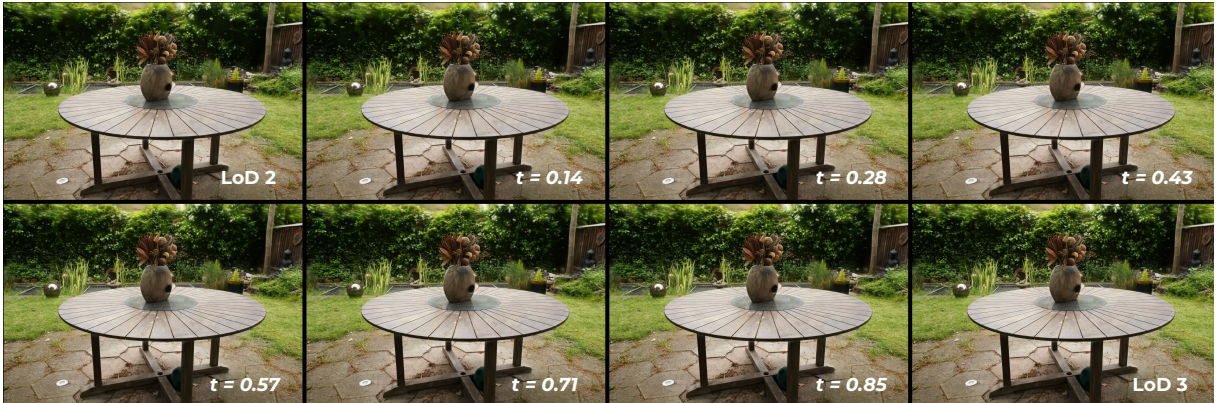


Fig. 5: Continuous transitions between adjacent levels using eight linear interpolation factors on *Garden*.

observe decoding speedups of up to $1762\times$ compared to prior methods, which makes progressive transmission and on-the-fly decompression practical in streaming settings. Decoding times include (i) decompression of all levels up to the selected

one and (ii) the merging step required to assemble the active set of primitives. For consistency with non-scalable baselines, we report three representative levels (out of eight) that match the table

Table 2: Performance comparison in terms of training, encoding, and decoding times. Encoding and decoding times are reported in seconds, while training time is reported in hours. For non-scalable methods, training time refers to a single compression level; obtaining multiple operating points would require training multiple models. GoDe produces all levels within a single training run.

Level	Context-GS			HAC			RDO-GS			Comp-GS		
	Enc↓	Dec↓	Train↓	Enc↓	Dec↓	Train↓	Enc↓	Dec↓	Train↓	Enc↓	Dec↓	Train↓
High	52.7	53.1	1.33	8.6	11.8	1.19	1.1	2.2	1.31	10.8	9.1	1.63
Medium	87.8	82.5		9.7	12.5		2.9	7.4		10.8	9.0	
Low	97.2	88.1		14.9	17.9		4.8	19.0		10.9	8.8	

Level	Ours (3DGS)			Ours (Scaffold-GS)			Ours (Octree-GS)		
	Enc↓	Dec↓	Train↓	Enc↓	Dec↓	Train↓	Enc↓	Dec↓	Train↓
High	10.6	1.2	1.20	2.1	0.2	0.57	5.0	0.2	1.12
Medium	18.1	1.9		2.9	0.3		6.6	0.3	
Low	32.7	2.4		4.2	0.5		8.8	0.3	

format and the range of model sizes of competing methods.

4.5 Ablations

We study the impact of key design choices: the number of levels L , the ranking strategy used to construct the hierarchy, and the effect of enforcing hierarchy constraints compared to training independent operating points.

4.5.1 Number of levels

We use $L = 8$ as a practical balance between granularity and overhead. Fig. 6a shows that increasing the number of levels slightly reduces quality at the lowest LoDs, while the highest levels remain largely unaffected. Overall, the method scales well with L , supporting finer control of operating points without introducing significant degradation.

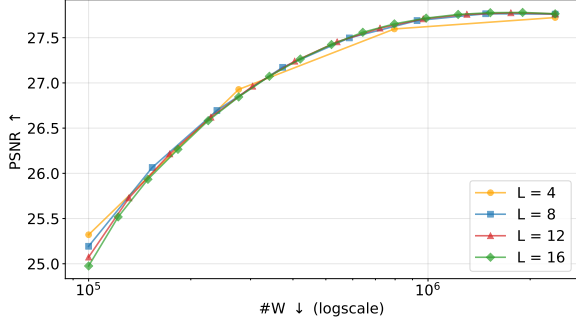
4.5.2 Pruning methods

Fig. 6b compares common ranking strategies used in 3DGS pruning and compaction. Our gradient-aggregated importance yields better reconstruction quality at matched primitive count compared to opacity-based pruning [1] and positional-gradient criteria [10]. We also compare against LightGaussian [25] under the same iterative masking protocol and observe superior results. Finally, we evaluate whether hierarchy construction requires iterative masking. We compare against a one-shot variant (*Ours (One Shot)*)

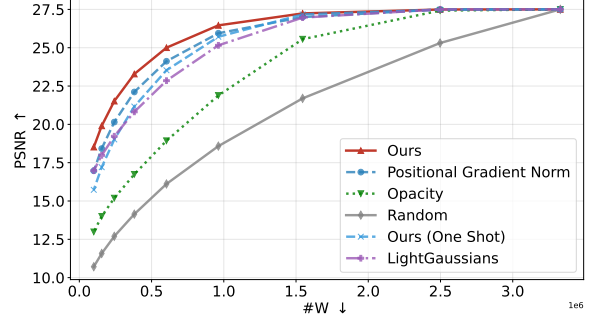
where gradients are accumulated once and levels are formed in a single pass. This results in a measurable quality drop, especially at lower LoDs, confirming that iterative masking improves stability and allocates coarse information more reliably. Since hierarchy construction is fast in practice, we adopt the iterative procedure throughout.

Comparison with PUP-3DGS

We compare our gradient-based pruning criterion with PUP-3DGS [21], a recent and well-known pruning approach for 3D Gaussian Splatting that relies on uncertainty- and parameter-driven heuristics rather than gradient information. To ensure a fair comparison, we strictly match the experimental conditions: both methods use the same backbone (vanilla 3DGS), the same number of Gaussians, and the same optimization budget, consisting of two pruning stages interleaved with two fine-tuning phases of 5k iterations each. Under these controlled settings, our approach consistently achieves higher reconstruction quality across all evaluated datasets, as reported in Tab. 3. Since the model architecture, the number of retained Gaussians, and the total number of optimization iterations are identical, the observed improvements can be attributed solely to the pruning criterion. While PUP-3DGS estimates Gaussian importance from uncertainty-related and parameter-level signals, our method ranks Gaussians using the aggregated gradient norm of all their parameters, directly measuring their contribution to the reconstruction loss.



(a) Effect of varying the number of levels L .



(b) Comparison of ranking strategies for hierarchy construction.

Fig. 6: Ablation study on Mip-NeRF360.

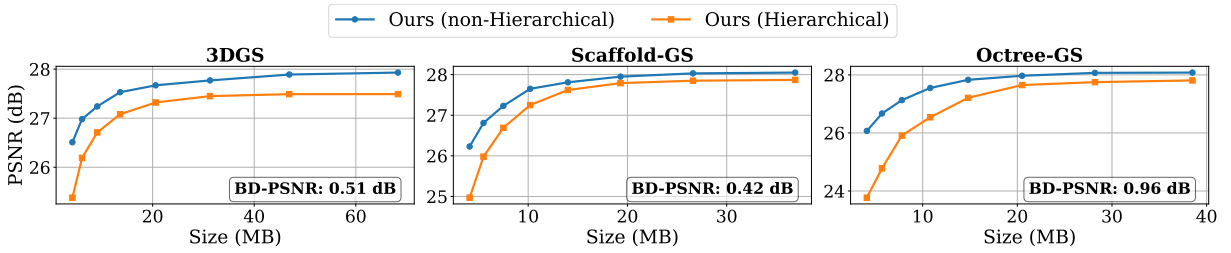


Fig. 7: Hierarchical vs. non-hierarchical variants of GoDe on Mip-NeRF360.

These results indicate that gradient-based importance provides a more accurate and stable pruning signal, leading to systematically better reconstruction quality even when used in isolation, without any hierarchical or multi-level structure.

4.5.3 Impact of hierarchical representation

We quantify the performance impact introduced by enforcing hierarchy constraints by comparing against a non-hierarchical variant that trains independent models at each operating point. Specifically, we repeat the following cycle for eight points: accumulate gradients for one epoch, mask the k lowest-score Gaussians, then fine-tune for 3,750 iterations (keeping the total budget at 30,000 iterations). This produces eight independent models, each optimized for a single point on the curve. Across the three backbones, BD-PSNR differences range from 0.42 to 0.96 dB (Fig. 7), indicating a small-to-moderate gap. This is expected: a single scalable model is necessarily constrained compared to independent per-point optimization.

Nevertheless, the practical benefits of GoDe (single training pipeline, instant access to multiple operating points, and fast progressive decoding) outweigh this gap in many deployment scenarios.

4.6 Limitations

GoDe enforces a hierarchical structure across compression levels, which constrains the solution space compared to independently optimized single-rate models. As shown in our ablation study, this can introduce a moderate performance gap with respect to non-scalable upper bounds, particularly at intermediate operating points. This trade-off is inherent to scalable representations and reflects the cost of supporting multiple operating points within a single model. In practice, the gap remains limited and is offset by substantial gains in deployability, storage efficiency, and instant rate selection. Our framework also assumes the availability of a pretrained 3DGS model as a starting point. We consider this assumption realistic in most application pipelines, where a high-quality base model is typically trained once and reused.

Table 3: Comparison with PUP-3DGS under matched pruning conditions. Both methods use the same vanilla 3DGS backbone and the same pruning/optimization budget. W denotes the number of retained Gaussians (in thousands).

	PUP-3DGS [21]				Ours (3DGS)			
	W	PSNR \uparrow	SSIM \uparrow	LPIPS \downarrow	W	PSNR \uparrow	SSIM \uparrow	LPIPS \downarrow
Mip-NeRF360	336	26.67	0.792	0.272	336	26.92	0.796	0.264
Tanks&Temples	178	22.72	0.807	0.244	178	22.96	0.808	0.243
DeepBlending	297	28.86	0.881	0.302	297	29.65	0.904	0.262

Importantly, GoDe does not require any additional retraining or per-rate fine-tuning beyond a single joint optimization stage, resulting in a total computational cost that is competitive with, or lower than, that of non-scalable alternatives producing multiple operating points.

5 Conclusion

In this work, we introduced GoDe, a practical framework for scalable compression and progressive level-of-detail in 3D Gaussian Splatting. By organizing Gaussians into a progressive hierarchy and integrating general, model-agnostic optimization mechanisms—including gradient-informed masking, quantization-aware fine-tuning, and entropy coding—GoDe enables a single trained model to operate across a wide range of compression rates and rendering budgets. Extensive experiments on explicit, neural, and hierarchical 3DGS backbones demonstrate that scalability and high reconstruction quality are not mutually exclusive. Despite its general design, GoDe achieves rate-distortion performance that is competitive with, and often comparable to, specialized non-scalable methods at matched model sizes, while uniquely providing instant access to multiple operating points without retraining. Beyond compression, the progressive representation naturally supports adaptive rendering, allowing models to dynamically adjust to changing performance constraints with smooth transitions between levels. Overall, GoDe shows that scalable compression and adaptive rendering in 3D Gaussian Splatting can be achieved without introducing specialized architectures or per-rate training procedures, by elevating generic optimization signals into a coherent and reusable hierarchical representation.

6 Data availability

Data supporting this study are openly available [26–28]. The source code will be open-sourced on a public GitHub link upon acceptance of the article.

References

- [1] Kerbl, B., Kopanas, G., Leimkühler, T., Dretakis, G.: 3d gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics* **42**(4) (2023)
- [2] Clark, J.H.: Hierarchical geometric models for visible surface algorithms. *Communications of the ACM* **19**(10), 547–554 (1976)
- [3] Hoppe, H.: View-dependent refinement of progressive meshes. In: *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, pp. 189–198 (1997)
- [4] Yan, Z., Low, W.F., Chen, Y., Lee, G.H.: Multi-scale 3d gaussian splatting for anti-aliased rendering. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 20923–20931 (2024)
- [5] Liu, Y., Luo, C., Fan, L., Wang, N., Peng, J., Zhang, Z.: Citygaussian: Real-time high-quality large-scale scene rendering with gaussians. In: Leonardis, A., Ricci, E., Roth, S., Russakovsky, O., Sattler, T., Varol, G. (eds.) *Computer Vision – ECCV 2024*, pp. 265–282. Springer, Cham (2025)
- [6] Ren, K., Jiang, L., Lu, T., Yu, M., Xu, L., Ni, Z., Dai, B.: Octree-gs: Towards consistent real-time rendering with lod-structured 3d

- gaussians. arXiv preprint arXiv:2403.17898 (2024)
- [7] Kerbl, B., Meuleman, A., Kopanas, G., Wimmer, M., Lanvin, A., Drettakis, G.: A hierarchical 3d gaussian representation for real-time rendering of very large datasets. *ACM Transactions on Graphics* **43**(4) (2024)
- [8] Shi, Y., Morin, G., Gasparini, S., Ooi, W.T.: Lapisgs: Layered progressive 3d gaussian splatting for adaptive streaming. arXiv preprint arXiv:2408.14823 (2024)
- [9] Deng, C.L., Tartaglione, E.: Compressing explicit voxel grid representations: fast nerfs become also small. In: *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pp. 1236–1245 (2023)
- [10] Ali, M.S., Qamar, M., Bae, S.-H., Tartaglione, E.: Trimming the fat: Efficient compression of 3d gaussian splats through pruning. arXiv preprint arXiv:2406.18214 (2024)
- [11] Ali, M.S., Bae, S.-H., Tartaglione, E.: Elmgs: Enhancing memory and computation scalability through compression for 3d gaussian splatting. arXiv preprint arXiv:2410.23213 (2024)
- [12] Niedermayr, S., Stumpfegger, J., Westermann, R.: Compressed 3d gaussian splatting for accelerated novel view synthesis. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10349–10358 (2024)
- [13] Xie, S., Zhang, W., Tang, C., Bai, Y., Lu, R., Ge, S., Wang, Z.: Mesongs: Post-training compression of 3d gaussians via efficient attribute transformation. In: *European Conference on Computer Vision* (2024). Springer
- [14] Girish, S., Gupta, K., Shrivastava, A.: EAGLES: Efficient Accelerated 3D Gaussians with Lightweight EncodingS (2024). <https://arxiv.org/abs/2312.04564>
- [15] Chen, Y., Wu, Q., Lin, W., Harandi, M., Cai, J.: Hac: Hash-grid assisted context for 3d gaussian splatting compression. In: *European Conference on Computer Vision* (2024)
- [16] Liu, X., Wu, X., Zhang, P., Wang, S., Li, Z., Kwong, S.: CompGs: Efficient 3d scene representation via compressed gaussian splatting. In: *Proceedings of the 32nd ACM International Conference on Multimedia*, pp. 2936–2944 (2024)
- [17] Papantonakis, P., Kopanas, G., Kerbl, B., Lanvin, A., Drettakis, G.: Reducing the memory footprint of 3d gaussian splatting. *Proceedings of the ACM on Computer Graphics and Interactive Techniques* **7**(1), 1–17 (2024)
- [18] Wang, H., Zhu, H., He, T., Feng, R., Deng, J., Bian, J., Chen, Z.: End-to-end rate-distortion optimized 3d gaussian representation. arXiv preprint arXiv:2406.01597 (2024)
- [19] Lu, T., Yu, M., Xu, L., Xiangli, Y., Wang, L., Lin, D., Dai, B.: Scaffold-gs: Structured 3d gaussians for view-adaptive rendering. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 20654–20664 (2024)
- [20] Lee, J.C., Rho, D., Sun, X., Ko, J.H., Park, E.: Compact 3d gaussian representation for radiance field. arXiv preprint arXiv:2311.13681 (2023)
- [21] Hanson, A., Tu, A., Singla, V., Jayawardhana, M., Zwicker, M., Goldstein, T.: Pup 3d-gs: Principled uncertainty pruning for 3d gaussian splatting. In: *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 5949–5958 (2025)
- [22] Ziv, J., Lempel, A.: A universal algorithm for sequential data compression. *IEEE Transactions on information theory* **23**(3), 337–343 (1977)
- [23] Kheradmand, S., Rebain, D., Sharma, G., Sun, W., Tseng, J., Isack, H., Kar, A., Tagliasacchi, A., Yi, K.M.: 3d gaussian splatting as markov chain monte carlo. arXiv preprint arXiv:2404.09591 (2024)

- [24] Morgenstern, W., Barthel, F., Hilsmann, A., Eisert, P.: Compact 3D Scene Representation via Self-Organizing Gaussian Grids (2024). <https://arxiv.org/abs/2312.13299>
- [25] Fan, Z., Wang, K., Wen, K., Zhu, Z., Xu, D., Wang, Z.: Lightgaussian: Unbounded 3d gaussian compression with 15x reduction and 200+ fps. arXiv preprint arXiv:2311.17245 (2023)
- [26] Barron, J.T., Mildenhall, B., Verbin, D., Srinivasan, P.P., Hedman, P.: Mip-nerf 360: Unbounded anti-aliased neural radiance fields. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 5470–5479 (2022)
- [27] Knapitsch, A., Park, J., Zhou, Q.-Y., Koltun, V.: Tanks and temples: Benchmarking large-scale scene reconstruction. *ACM Transactions on Graphics (ToG)* **36**(4), 1–13 (2017)
- [28] Hedman, P., Philip, J., Price, T., Frahm, J.-M., Drettakis, G., Brostow, G.: Deep blending for free-viewpoint image-based rendering **37**(6), 257–125715 (2018)
- [29] Bulò, S.R., Porzi, L., Kotschieder, P.: Revising densification in gaussian splatting. arXiv preprint arXiv:2404.06109 (2024)
- [30] Wang, Y., Li, Z., Guo, L., Yang, W., Kot, A., Wen, B.: Contextgs: Compact 3d gaussian splatting with anchor level context model. *Advances in neural information processing systems* **37**, 51532–51551 (2024)

A Additional Details

This appendix provides additional implementation details, ablation studies, and extended experimental results that complement the main paper. We report the full pseudocode of the proposed hierarchical construction and fine-tuning procedure, details on quantization-aware training, implementation specifics for Scaffold-GS, alternative sampling strategies, and extended quantitative and qualitative evaluations.

Algorithm 1 Iterative Masking

Require: Pre-trained Gaussian Model \mathcal{G}_p , number of levels L , progression k_s , training set $\mathcal{D}_{\text{train}}$

Ensure: Hierarchical Gaussian model $(\mathcal{G}_0, \mathcal{E}_s)$

- 1: $\mathcal{E}_s \leftarrow$ empty list of size $L - 1$
- 2: $\mathcal{G}_{L-1} \leftarrow \mathcal{G}_p$
- 3: **for** $l \leftarrow L - 1$ to 1 **do**
- 4: computeGradients($\mathcal{G}_l, \mathcal{D}_{\text{train}}$)
- 5: $k_l \leftarrow k_s[l]$
- 6: $\mathcal{E}_l \leftarrow \text{lowest}_{k_l} \left\{ \left\| \frac{\partial \mathcal{L}}{\partial \theta_i} \right\|_2 \right\}_{i \in \mathcal{G}_l}$
- 7: append($\mathcal{E}_l, \mathcal{E}_s$)
- 8: $\mathcal{G}_{l-1} \leftarrow \text{maskGaussians}(\mathcal{E}_l, \mathcal{G}_l)$
- 9: **end for**
- 10: reverse(\mathcal{E}_s)
- 11: **return** $(\mathcal{G}_0, \mathcal{E}_s)$

Algorithm 2 Fine Tune

Require: Hierarchical Gaussian Model $(\mathcal{G}_0, \mathcal{E}_s)$, training set $\mathcal{D}_{\text{train}}$, number of iterations F , boolean *compress*

Ensure: Fine-tuned Hierarchical Model $(\mathcal{G}_0, \mathcal{E}_s)$

- 1: **for** $i \leftarrow 0$ to F **do**
- 2: Random sample viewpoint v and level $l \in \{0, |\mathcal{E}_s|\}$
- 3: $\mathcal{G}_l \leftarrow \mathcal{G}_0 \cup \bigcup_{i=1}^l \mathcal{E}_i$
- 4: render(\mathcal{G}_l, v)
- 5: compute loss and optimize \mathcal{G}_l
- 6: **if** *compress* **then**
- 7: compressModel($\mathcal{G}_0, \mathcal{E}_s$)
- 8: **end if**
- 9: **end for**
- 10: **return** $(\mathcal{G}_0, \mathcal{E}_s)$

B Pseudocode

Here, we present the pseudocode for constructing the hierarchical Gaussian model and for the subsequent fine-tuning procedure. Following Eq. 3, the hierarchical representation \mathcal{G} consists of a base layer \mathcal{G}_0 and an ordered list of enhancement layers $\mathcal{E}_s = \{\mathcal{E}_1, \dots, \mathcal{E}_{L-1}\}$. The procedure starts by defining a progression that specifies the number of primitives assigned to each level. Given a minimum number of Gaussians c_{min} for the base layer and a maximum number c_{max} for the full model, we compute a sequence k_s that determines the size of each enhancement layer. In our experiments, c_{max} is set to 0.75% of the original model size. Given the progression k_s , the hierarchy is constructed in a top-down manner using gradient-informed iterative masking, as detailed in Alg. 1. Starting from the full model, Gaussians are progressively assigned to enhancement layers based on their aggregated gradient importance. Finally, all levels are jointly refined through a single quantization-aware fine-tuning stage with random level sampling, as described in Alg. 2.

C Quantization-Aware Fine-Tuning Details

To enable efficient compression while preserving reconstruction quality, we adopt a quantization-aware fine-tuning (QAT) strategy based on PyTorch FakeQuantization. QAT simulates low-precision arithmetic during training and propagates gradients through the quantizer using a straight-through estimator (STE), allowing the model to adapt to quantization noise prior to deployment. For vanilla 3D Gaussian Splatting, spherical harmonics (SH) coefficients are quantized to 8-bit integers, as they dominate memory usage. Opacity, scaling, and rotation parameters are stored in 16-bit floating point, while Gaussian positions are left unquantized to avoid geometric distortions. This design yields a favorable trade-off between compression efficiency and reconstruction quality. We use per-tensor affine quantization. Given a floating-point value x , quantization is defined as

$$x_q = \text{clip} \left(\text{round} \left(\frac{x}{\text{scale}} + \text{zero_point} \right), q_{\text{min}}, q_{\text{max}} \right), \quad (12)$$

with

$$\text{scale} = \frac{x_{\max} - x_{\min}}{q_{\max} - q_{\min}}, \quad \text{zero_point} = q_{\min} - \frac{x_{\min}}{\text{scale}}. \quad (13)$$

Dequantization is performed as

$$x_{\text{dq}} = (\text{scale} \cdot x_q) - \text{zero_point}. \quad (14)$$

To further stabilize training under quantization, we add an ℓ_1 regularization term on the quantized parameters (SH coefficients for explicit 3DGS, anchor features for neural variants), encouraging compact representations. Quantization-aware fine-tuning is performed jointly with random level sampling, enabling all levels of the hierarchy to consistently adapt to quantization effects within a single training run.

D Scaffold-GS-based Implementations

Scaffold-GS [19] is a neural-based 3DGS formulation in which anchor points generate view-dependent Gaussians. Since our approach is model-agnostic, the hierarchical construction follows the same iterative masking strategy used for explicit 3DGS, with anchor points replacing Gaussians as the pruning unit. Each anchor point is associated with a 32-dimensional feature vector. The base layer \mathcal{G}_0 and enhancement layers \mathcal{E}_s are therefore defined in terms of anchor points. The progression is computed identically to the explicit case, using $c_{\min} = 50,000$ and $c_{\max} = 0.85 \cdot W$, where W denotes the number of anchor points in the original model. For compression, instead of pruning spherical harmonics, we apply quantization-aware fine-tuning with random level sampling. Features are quantized to 8-bit integers, while anchor positions, scaling, and offsets are stored in 16-bit floating point.

E Sampling Strategies During Fine-Tuning

During fine-tuning, levels are sampled uniformly according to $l \sim \text{Uniform}(0, L - 1)$. We additionally explored a weighted sampling strategy that assigns higher probability to higher levels. To implement weighted sampling, we transform the

progression k_s into a probability distribution normalized between 0 and 1. While this strategy improves reconstruction quality at higher levels, it significantly degrades lower levels, as shown in Tab. 4. Across all experiments, uniform sampling provides a more balanced trade-off and is therefore used by default.

F Impact of Fine-Tuning and Compression

We analyze the impact of fine-tuning and compression both quantitatively and qualitatively. As reported in Tab. 5, fine-tuning substantially improves reconstruction quality, especially for lower levels of detail, while slightly reducing the quality of the highest levels by approximately 0.02, dB. Compression achieves storage reductions ranging from 86.88% to 99.37%, depending on the level, with negligible impact on reconstruction quality. Fig. 8 visualizes the effect of fine-tuning, showing that non-fine-tuned models exhibit visible artifacts and holes, while fine-tuned models are significantly more stable.

G Additional Results

G.1 Full results

In Tab. 6 we report the full results for all the 8 LoD levels obtained with our framework.

G.2 Full LoD Visualizations

Figures 9, 10, and 11 present full level-of-detail visualizations for all datasets using vanilla 3DGS. Transitions between levels are smooth, with progressively increasing detail and no visible popping artifacts. Lower levels primarily lose high-frequency background details, while preserving overall scene structure.

Table 4: Comparison among sampling strategies on Mip-NeRF360 and 3DGS. US: Uniform Sampling; WS: Weighted Sampling.

Level	W	US			WS (G=5)			WS (G=8)			WS (G=10)		
		PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓
LoD 0	100	25.20	0.719	0.408	24.72	0.706	0.417	24.12	0.693	0.426	23.30	0.675	0.436
LoD 1	155	26.06	0.752	0.372	25.70	0.742	0.379	25.26	0.732	0.386	24.70	0.721	0.392
LoD 2	242	26.65	0.778	0.340	26.41	0.771	0.345	26.13	0.764	0.349	25.90	0.760	0.351
LoD 3	381	27.08	0.796	0.312	26.94	0.792	0.315	26.89	0.791	0.315	26.77	0.788	0.316
LoD 4	603	27.33	0.807	0.291	27.37	0.808	0.289	27.35	0.807	0.289	27.34	0.807	0.288
LoD 5	962	27.45	0.813	0.276	27.56	0.815	0.272	27.58	0.815	0.271	27.58	0.815	0.270
LoD 6	1546	27.47	0.814	0.269	27.58	0.816	0.264	27.61	0.817	0.263	27.63	0.817	0.261
LoD 7	2497	27.47	0.815	0.267	27.57	0.816	0.262	27.60	0.817	0.260	27.62	0.817	0.259
AVG	811	26.84	0.787	0.317	26.73	0.783	0.318	26.57	0.779	0.320	26.36	0.775	0.322



Fig. 8: Effects of fine-tuning on the train scene of *Tanks&Temples* dataset. In the first row, LoDs without fine-tuning; on the second, LoDs after fine-tuning.

Table 5: Effects of fine-tuning and compression on Mip-NeRF360 with 3DGS (vanilla version).

LoD	W	Mask Only			Fine-Tuning			Fine-Tuning + Compression		
		PSNR↑	SSIM↑	Size [MB]↓	PSNR↑	SSIM↑	Size [MB]↓	PSNR↑	SSIM↑	Size [MB]↓
0	100	18.52	0.553	24.1	25.20	0.719	24.1	25.28	0.719	3.8
1	155	19.90	0.606	36.7	26.06	0.752	36.7	26.10	0.751	5.6
2	242	21.52	0.662	56.4	26.65	0.778	56.4	26.63	0.776	8.6
3	381	23.28	0.715	89.1	26.84	0.787	89.1	27.05	0.793	13.3
4	603	25.00	0.760	142.8	27.08	0.796	142.8	27.30	0.804	20.6
5	962	26.45	0.793	216.6	27.33	0.807	69.8	27.40	0.809	32.1
6	1546	27.23	0.810	365.9	27.45	0.813	227.2	27.43	0.811	49.0
7	2497	27.49	0.815	562.2	27.47	0.814	365.6	27.42	0.811	73.7



Fig. 9: Full LoD visualization with 3DGS on *Tanks&Temples* dataset.

Table 6: Main results on the 3 datasets (Mip-NeRF360, Tanks&Temples, and Deep Blending). All results were tested on a NVIDIA A40.

Dataset	Mip-NeRF360						Tanks&Temples						Deep Blending						
	PSNR	SSIM	LPIPS	Size	W	FPS	PSNR	SSIM	LPIPS	Size	W	FPS	PSNR	SSIM	LPIPS	Size	W	FPS	
Level	↑	↑	↓	[MB]↓	↓	↑	↑	↑	↓	↓	↑	↑	↑	↑	↓	↓	↓	↑	
Ours 3DGS	LoD 0	25.38	0.716	0.407	4.2	100	421	21.93	0.768	0.334	4.0	100	540	29.03	0.886	0.376	4.1	100	652
	LoD 1	26.19	0.748	0.371	6.1	155	358	22.70	0.792	0.304	5.3	144	453	29.39	0.893	0.362	5.5	146	568
	LoD 2	26.71	0.773	0.339	9.1	242	304	23.22	0.811	0.278	7.2	208	388	29.54	0.897	0.351	7.5	212	489
	LoD 3	27.08	0.791	0.311	13.6	381	256	23.61	0.825	0.256	9.9	301	320	29.61	0.900	0.342	10.3	310	419
	LoD 4	27.32	0.804	0.289	20.6	603	213	23.85	0.835	0.239	13.6	438	260	29.64	0.901	0.336	14.1	452	360
	LoD 5	27.45	0.812	0.273	31.3	962	174	23.99	0.840	0.226	18.7	640	215	29.65	0.901	0.331	19.3	660	311
	LoD 6	27.49	0.815	0.263	46.9	1546	139	24.04	0.844	0.218	25.6	938	178	29.65	0.902	0.327	26.5	965	263
	LoD 7	27.49	0.816	0.259	68.3	2497	117	24.04	0.844	0.216	34.9	1379	157	29.61	0.901	0.326	36.1	1411	220
baseline	27.84	0.831	0.239	794.6	3528	87	24.15	0.862	0.187	435.2	1839	133	29.65	0.904	0.309	667.0	2821	164	
Ours Scaffold-GS	LoD 0	24.97	0.710	0.409	4.1	50	144	23.13	0.806	0.295	4.1	50	167	29.75	0.897	0.358	4.0	50	178
	LoD 1	25.98	0.742	0.376	5.5	68	133	23.51	0.821	0.273	5.0	61	155	30.00	0.901	0.349	4.6	58	172
	LoD 2	26.69	0.768	0.345	7.5	94	122	23.72	0.832	0.256	6.0	75	145	30.15	0.904	0.343	5.4	68	169
	LoD 3	27.25	0.789	0.318	10.2	129	108	23.86	0.839	0.242	7.4	93	138	30.23	0.906	0.338	6.2	80	165
	LoD 4	27.62	0.803	0.297	14.0	179	94	23.96	0.844	0.231	9.0	114	125	30.29	0.908	0.334	7.2	93	161
	LoD 5	27.79	0.810	0.284	19.3	248	83	24.01	0.848	0.224	11.0	141	116	30.33	0.909	0.331	8.4	109	155
	LoD 6	27.85	0.812	0.278	26.6	346	74	24.04	0.850	0.219	13.5	174	108	30.35	0.909	0.329	9.7	127	151
	LoD 7	27.87	0.813	0.276	36.9	483	69	24.04	0.851	0.218	16.5	215	98	30.33	0.909	0.328	11.3	149	148
baseline	27.77	0.813	0.270	171.1	555	79	24.03	0.853	0.210	76.4	247	108	30.28	0.909	0.318	53.5	171	179	
Ours Octree-GS	LoD 0	23.77	0.664	0.447	4.1	50	149	22.42	0.776	0.369	4.2	50	147	29.38	0.891	0.365	3.7	50	213
	LoD 1	24.73	0.697	0.413	5.7	69	140	23.14	0.799	0.339	5.7	68	134	29.65	0.895	0.359	4.1	56	212
	LoD 2	25.61	0.729	0.378	7.8	96	127	23.62	0.818	0.310	7.7	93	121	29.81	0.897	0.354	4.6	64	212
	LoD 3	26.44	0.761	0.341	10.8	133	113	23.98	0.834	0.281	10.3	127	107	29.91	0.899	0.350	5.1	72	205
	LoD 4	27.11	0.789	0.305	14.8	186	100	24.25	0.848	0.256	13.7	173	92	29.97	0.900	0.347	5.7	81	200
	LoD 5	27.55	0.808	0.279	20.5	262	86	24.45	0.859	0.235	18.2	235	78	30.02	0.901	0.344	6.4	91	195
	LoD 6	27.75	0.815	0.265	28.2	371	78	24.57	0.866	0.223	24.1	321	66	30.03	0.901	0.343	7.1	103	194
	LoD 7	27.81	0.817	0.261	38.5	527	73	24.61	0.868	0.219	31.0	438	60	30.03	0.901	0.342	7.8	116	192
baseline	27.96	0.821	0.252	157.2	606	69	24.69	0.873	0.204	131.0	503	49	30.09	0.902	0.338	34.6	503	174	



Fig. 10: Full LoD visualization with 3DGS on *DeepBlending* dataset.



Fig. 11: Full LoD visualization with 3DGS on *Mip-NeRF360* dataset.