

UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

MASTER DEGREE IN ICT FOR INTERNET AND MULTIMEDIA

**Toward a New Gaussian Splatting Compression
Scheme via Region Adaptive Hierarchical
Transform**

SUPERVISOR:

PROF.SSA FEDERICA BATTISTI

CO-SUPERVISOR:

GIUSEPPE VALENZISE

L2S, CentraleSupélec, Paris

CANDIDATE:

ANNALISA GALLINA

2103356

ACADEMIC YEAR 2024-2025

Abstract

3D Gaussian Splatting has recently emerged as a powerful representation for Novel View Synthesis but it comes with significant storage and bandwidth requirements, which limit its usability in practical applications. In this work, we tackle the problem of compressing Gaussian Splatting by interpreting it as a Point Cloud with rich per-point attributes, and we exploit the Region-Adaptive Hierarchical Transform (RAHT) – a lightweight and well-established tool in Point Cloud compression standards – to decorrelate and compact the attribute information.

While RAHT has been successfully applied in Point Cloud compression, previous attempts to directly integrate it into Gaussian Splatting pipelines have shown limited efficiency due to the redundant and unstructured nature of Gaussian attributes. To overcome these limitations, we introduce two key contributions. First, we propose a quantization-aware fine-tuning strategy that adapts the Gaussian parameters to the transform domain, promoting sparsity in the transformed coefficients. Second, we design a systematic bit allocation strategy across attribute channels, inspired by quantization matrices in image compression. By allocating bits according to the statistical relevance and energy distribution of each attribute, our method improves rate–distortion trade-offs without requiring changes to the baseline representation.

Our framework maintains full compatibility with standard Point Cloud coding tools, while reducing storage requirements. Experimental results show that, despite its simplicity, the proposed method achieves state-of-the-art performance among post-training compression approaches, and in several cases, it matches or outperforms more complex methods. This demonstrates that efficient compression can be achieved without altering the underlying representation, allowing practical deployment of Gaussian Splatting in real-world applications.

*To Paris,
for making me realize I am worth far more than I ever believed*

Contents

Abstract	III
List of Figures	XI
List of Tables	XIII
1 Introduction	1
1.1 Motivation	1
1.2 Proposed Contributions	3
1.3 Thesis Structure	4
2 Background	7
2.1 Novel View Synthesis	7
2.2 3D Scene representation	8
2.2.1 Point Cloud	8
2.2.2 NeRF	9
2.2.3 3D Gaussian Splatting	10
2.2.3.1 Training Process	11
2.2.3.2 Image Rendering	13
3 Related Works	17
3.1 Compression in Multimedia and 3D Data	17
3.1.1 Point Cloud Compression	18
3.1.1.1 VPCC	19
3.1.1.2 GPCC	20
3.2 3DGS compression	20
3.2.1 Post-training Methods	21
3.2.1.1 Gaussian Pruning	21

3.2.1.2	SH Coefficient Distillation	22
3.2.1.3	Geometry Compression	24
3.2.1.4	Attribute Compression	25
3.2.1.5	End-to-end Compression	27
3.2.2	Training Optimization Methods	28
3.2.3	Retraining Methods	29
3.2.3.1	Scaffold-GS	29
3.2.3.2	Scaffold-based methods	30
4	Proposed Method	35
4.1	Signal Analysis	35
4.1.1	Inter-point Correlation	35
4.1.2	Intra-point Correlation	36
4.1.3	Impact of AC SH coefficients	37
4.2	Proposed Pipeline	39
4.3	Pruning	40
4.4	Geometry Compression	41
4.4.1	Voxelization	41
4.4.2	Octree coding	41
4.5	Attribute Pre-processing	42
4.5.1	Recoloring	42
4.5.2	Rotation to Euler Mapping	42
4.6	RAHT	43
4.6.1	Weighted RAHT	46
4.7	Bit Allocation	48
4.8	Block-wise Scalar Quantization	49
4.9	Fine-tuning	50
4.9.1	Quantization-aware Fine-tuning	50
4.9.2	Sparsity-promoting Fine-tuning	51
4.10	Entropy Coding	53
4.10.1	Deflate	53
4.10.2	Set Partitioning in Hierarchical Trees	54
4.10.2.1	SPIHT for 3DGS Compression	55

5	Experimental Results	57
5.1	Experimental Setting	57
5.1.1	Datasets	57
5.1.2	Evaluation Metrics	59
5.1.2.1	Distortion Metrics	60
5.1.2.2	Rate Metric	62
5.1.3	Implementation Details	62
5.2	Compression Results	63
5.3	Ablation studies	65
5.3.1	Gaussian Pruning	65
5.3.2	Voxelization	66
5.3.3	Octree Coding	67
5.3.4	Block-wise Scalar Quantization	67
5.3.5	Bit Allocation	68
5.3.6	Sparsity-promoting Fine-tuning	69
5.3.7	Effects of varying λ_S	70
6	Conclusions	73
6.1	Open Issues	74
6.2	Future Works	75
	Acronyms	77
	Bibliography	79

List of Figures

2.1	Example of Novel View Synthesis	7
2.2	Gaussian Splatting training pipeline	11
2.3	Top: duplication in under-reconstructed areas. Bottom: splitting in over-reconstructed areas	12
2.4	(a) From 3D Gaussians to 2D splats. (b) - (d) Rasterization process.	14
3.1	Composition of each Gaussian	23
3.2	Scaffold-GS pipeline [1]	30
4.1	Correlation Matrix for <i>Room</i> (Mip-NeRF360) and <i>Train</i> (Tanks and Temples).	37
4.2	Entropy of AC SH Coefficients of scene <i>Room</i> (Mip-NeRF360).	38
4.3	Overview of the proposed system.	39
4.4	RAHT pipeline [2]	45
4.5	RD results for <i>Mic</i> (Nerf-Synthetic), <i>Counter</i> and <i>Room</i> (Mip-NeRF360) for different weights initializations	47
4.6	RD Curves for Rotation and 2 nd order SH coefficients.	49
4.7	RD curves for <i>Mic</i> (Nerf-Synthetic) for different numbers of fine-tuning iterations	53
4.8	SPIHT results for <i>Train</i> (left) and <i>Room</i> (right).	56
5.1	Mip-NeRF360 Dataset	58
5.2	Tanks and Temples Dataset	59
5.3	NeRF-Synthetic Dataset	59
5.4	RD curves for different voxelization depths	66
5.5	RD trade-off for different block sizes	68
5.6	RD curves with and without sparsity-promoting term for different numbers of fine-tuning iterations	70
5.7	RD Curves for different λ_S values	70

List of Tables

4.1	Local Variance ($k = 100$)	36
4.2	Global Variance	36
4.3	PSNR on training set for the baseline 3DGS [3], evaluated under two configurations: (i) with the maximum SH order set to 0, and (ii) with the maximum SH order set to 3.	38
4.4	Proposed channel-wise bit allocation.	49
5.1	Configuration values per scene: number of blocks (n_block), octree depth, and pruning ratio.	63
5.2	Compression performance results on 3 datasets. Model size is in MB. Best, second best, and third best values for each column are denoted in red, yellow, and green, respectively.	64
5.3	Scene-by-scene comparison with MesonGS [4] on the Mip-NeRF360 Dataset	65
5.4	Performance with and without Gaussian Pruning	66
5.5	Performance with and without Octree Coding	67
5.6	Our bit allocation scheme vs. uniform allocation	68
5.7	Comparison of fine-tuning methods: (i) no fine-tuning, (ii) fine-tuning without sparsity loss, and proposed.	69

Chapter 1

Introduction

1.1 Motivation

In recent years, the rise of mobile and wearable devices, coupled with their increasing affordability and availability, has made Virtual Reality (VR) and Augmented Reality (AR) technologies more accessible than ever before. This growing accessibility has driven significant changes in how users experience and interact with digital content, enabling new applications across diverse fields such as entertainment, education, and healthcare.

To ensure a high Quality of Experience, it is essential to provide users with a truly immersive environment, which demands that the visual quality of the content be as high as possible. Achieving this goal, however, poses significant challenges, as it requires carefully balancing the need for high-fidelity rendering with the limited computational and storage resources typical of low-power devices.

The need for this trade-off has motivated researchers to develop advanced, high-fidelity 3D scene representations and Novel View Synthesis algorithms, which generate unseen 3D views of a scene or object from limited input information, i.e. a finite set of input views.

Traditionally, 3D model rendering for eXtended Reality (XR) applications has relied on explicit representations such as Point Clouds or meshes. While these methods can be effective, they often demand significant computational power for real-time rendering and tend to deliver suboptimal visual quality.

Remarkable progress has been made with the introduction of Neural Radiance Fields [5], which use neural networks to learn a continuous radiance field capable of rendering highly realistic novel views. Despite their impressive results in terms of visual quality, these approaches suffer from expensive training requirements and slow rendering times due to their

implicit nature.

Further advancements have come with 3D Gaussian Splatting (3DGS) [3], which represents scenes explicitly using 3D volumetric Gaussians. This approach enables real-time rendering, fast training, and high visual quality, representing the state-of-the-art in the field at the moment. However, despite these advantages, the representation remains memory-intensive, creating challenges for practical deployment on low-power, resource-constrained devices.

For this reason, alongside developing more efficient Novel View Synthesis algorithms, research is also increasingly focused on making 3D representations themselves more efficient and better suited for practical applications. Several methods have already been proposed in the literature for compressing Gaussian Splatting representations and will be discussed in detail in Chapter 3, but most of them still have significant limitations:

- Many recent methods aim to improve compression by retraining the entire 3DGS representation [1], [6], [7] to enforce greater structural coherence, which typically leads to better Rate-Distortion (RD) performance. However, these retraining pipelines come with significant drawbacks: they are computationally intensive, time-consuming, and often require high-end hardware, limiting their applicability in real-world scenarios. Furthermore, they introduce major changes to the original training and rendering workflow, making them incompatible with existing implementations and unsuitable in situations where retraining from scratch is not possible.
- Several approaches rely on neural networks to improve compression, for example by predicting probability distributions for entropy coding (as in [6]), learning Gaussian attributes (as in [8], [9]), or performing end-to-end compression (as in [10]). Although these learning-based codecs can achieve excellent results, they introduce considerable complexity in both the encoder and the decoder, and they pose major challenges for standardization: every retraining, fine-tuning, or change in network architecture effectively creates a new codec version, complicating standard definitions and backward compatibility. Furthermore, even the same model can produce slightly different weights on different hardware or with different random seeds, meaning bitstreams might not decode identically everywhere.
- Many existing methods primarily focus on removing redundant or less perceptually important information without carefully considering the underlying structure of the signal itself. This is different from traditional approaches in other areas of multimedia compression, where signals are often transformed into alternative representations that are

easier to encode. Applying this kind of transformation strategy to Gaussian Splatting representations could potentially lead to better compression performance and reduced storage requirements.

Overall, overcoming these limitations is essential to develop practical, standardized, and efficient compression techniques that can make high-quality 3D content truly usable across a wide range of devices and real-world applications.

In contrast, classical Point Cloud compression techniques, such as those based on transform coding and quantization, have demonstrated good RD results and strong performance in standardization efforts thanks to their interpretability, modularity, and decoder simplicity. In simple terms, Gaussian Splat can be viewed as a Point Cloud where each point carries not only its 3D position and color but also additional attributes such as scale, rotation, opacity, and SH coefficients, as will be further detailed in Section 2.2.3. Despite this strong similarity, to the best of our knowledge, very few methods have explored the application of established Point Cloud compression techniques to the Gaussian Splatting domain [4], [11]. This gap presents a valuable opportunity, both for improving compression performance and for enabling compatibility with standardized, hardware-friendly decoding pipelines.

1.2 Proposed Contributions

This master’s thesis investigates how classical Point Cloud compression techniques can be effectively adapted and applied to compress Gaussian Splatting representations. We introduce a novel, hybrid compression framework for 3DGS that builds upon well-established principles from traditional Point Cloud coding, with a particular focus on the Region Adaptive Hierarchical Transform (RAHT) [2], taking inspiration from the MPEG GPCC standard [12]. Our goal is to combine the efficiency and simplicity of classical compression techniques with the high visual quality offered by modern 3D representations, while ensuring the method remains compatible with the original 3DGS pipeline.

The main contributions of this work are as follows:

- **Adapting Point Cloud compression tools to 3DGS:** We repurpose Octree coding and RAHT, two core techniques in standardized Point Cloud compression, for the Gaussian Splatting domain. This allows us to efficiently encode geometry and attributes while preserving the explicit structure of the 3DGS format and ensuring compatibility with standard processing pipelines.

- **Sparsity-aware fine-tuning strategy:** We introduce a fine-tuning phase applied before transformation. This step includes a sparsity-promoting loss that encourages more compressible transform-domain coefficients, thereby enhancing rate-distortion performance without retraining the full model.
- **Scene-independent bit allocation:** We propose a simple yet robust bit allocation strategy that assigns quantization levels to each attribute based on an extensive analysis of their energy and perceptual impact. This design avoids the need for scene-specific adaptation, reduces encoder complexity, and keeps side information to a minimum.
- **Comprehensive experimental validation:** We conduct extensive experiments across multiple scenes and datasets to evaluate the effectiveness of the proposed compression pipeline. Comparisons with recent state-of-the-art methods highlight that our approach achieves competitive performance, while maintaining full compatibility with the original 3DGS structure.

Overall, our results show that it is possible to achieve strong compression performance in 3DGS with minimal architectural changes, by leveraging classical, interpretable coding tools. The proposed approach offers a practical and efficient alternative to more complex retraining-based pipelines, retaining the rendering quality and speed that make Gaussian Splatting an attractive solution for real-time applications.

1.3 Thesis Structure

The thesis is structured as follows:

- Chapter 2 introduces the Novel View Synthesis problem and provides the theoretical background on 3D scene representation. It focuses on analyzing the strengths and challenges of existing approaches to explain how 3DGS works and why it is currently considered state-of-the-art in the field.
- Chapter 3 offers a comprehensive survey of existing work on Gaussian Splatting compression. It begins with a broad overview of common multimedia compression techniques to establish the foundations on which most 3DGS compression methods are built.
- Chapter 4 describes the proposed method in detail, explaining all implemented algorithms and discussing both their strengths and any limitations that prevented their integration into the final pipeline.

- Chapter 5 presents detailed information about the datasets used for validation, along with the methodology employed to evaluate the RD performance and reports the results of the conducted experiments, evaluating the performance of the proposed solution.
- Chapter 6 summarizes the key findings of the study, draws conclusions based on the results obtained, and discusses potential directions for future work.

Chapter 2

Background

Over the years, a variety of methods, ranging from traditional geometry-based models to recent learning-based techniques, have been developed to address Novel View Synthesis (NVS) problem. In this chapter, we review the foundations of Novel View Synthesis and analyze the most common 3D scene representations, from simple Point Clouds to Neural Radiance Fields for View Synthesis (NeRF) and 3DGS. This background will clarify the strengths and limitations of each approach and motivate the rise of advanced representations that enable state-of-the-art synthesis results.

2.1 Novel View Synthesis

Novel View Synthesis addresses the challenge of generating photorealistic images of a scene from new, unseen viewpoints, given a limited number of input views. In other words, it aims to reconstruct a continuous view space from sparse observations, allowing for realistic rendering of the scene from any arbitrary camera pose (Figure 2.1).

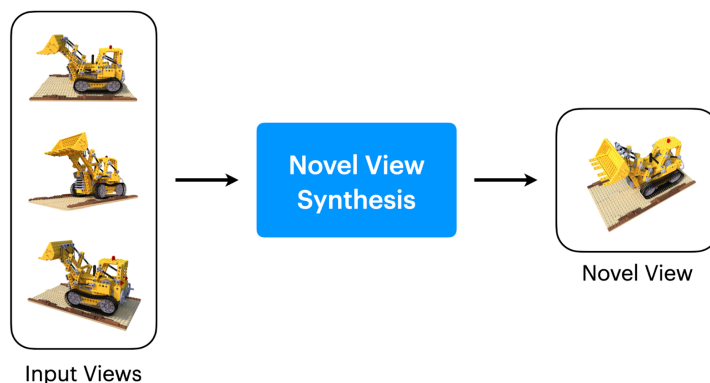


Figure 2.1: Example of Novel View Synthesis

NVS is tightly related to the problem of 3D scene representation, which refers to how

to model the structure, appearance, and geometry of the scene in a way that allows for accurate and efficient rendering from novel perspectives. The quality and efficiency of a view synthesis method is heavily influenced by the underlying scene representation it employs. A suitable representation must capture enough geometric detail and view-dependent appearance to enable realistic interpolation between the known views, while also remaining compact and computationally manageable.

Historically, traditional methods approached NVS by explicitly modeling geometry through meshes or depth maps and using image-based techniques, that means interpolating either between corresponding pixels from the input images, as proposed in [13], or between rays in space. However, these struggled with occlusions, non-Lambertian surfaces, and complex scene layouts.

With the advent of learning-based approaches, new forms of scene representations, both implicit (e.g., neural fields) and explicit (e.g., Point Clouds or Gaussians), have emerged. They typically better encode the scene’s properties and support higher-quality synthesis.

In this context, choosing the right 3D scene representation becomes central to solving NVS effectively. Modern techniques such as NeRF and 3DGS exemplify this shift, each leveraging a different kind of scene representation to achieve state-of-the-art results. These approaches, and the representations they rely on, will be discussed in detail in the following section.

2.2 3D Scene representation

In this section, the most commonly used 3D scene representation methods will be analyzed. For each method, we will highlight its strengths and weaknesses to understand why the current state-of-the-art representation, 3D Gaussian Splatting, has emerged.

2.2.1 Point Cloud

Point Clouds are one of the most fundamental representations for 3D data. A Point Cloud consists of a set of discrete points located in three-dimensional space, typically described by their Cartesian coordinates (x, y, z) . Depending on the source and application, each point can also carry additional attributes such as color (RGB), surface normals, intensity, or semantic labels. These representations are commonly produced by 3D acquisition technologies such as LiDAR scanners, structured light systems, time-of-flight cameras, and multi-view stereo reconstruction.

One of the main strengths of Point Clouds lies in their simplicity. They offer a straightforward and flexible way to represent the geometry of real-world objects and environments without requiring the explicit connectivity information inherent in meshes or the volumetric structure of grids. In other words, points are completely independent of one another, with no connections established to define surfaces approximating the 3D object. This simplicity allows Point Clouds to be easily acquired, stored, and visualized, making them suitable for a wide range of applications, from small-scale object modeling to large-scale mapping tasks such as those in robotics or autonomous driving. Moreover, their flexibility enables them to represent both dense and sparse data effectively, adapting well to the varying resolutions and coverage produced by different scanning devices.

However, despite their advantages, Point Clouds also present several weaknesses. A key limitation is their lack of inherent structure: unlike mesh-based models, Point Clouds do not encode relationships between points, which complicates tasks like surface reconstruction or high-quality rendering. Additionally, Point Clouds often exhibit non-uniform density, as some regions of the scene may be heavily oversampled while others are undersampled or even missing, depending on occlusions, sensor limitations, or scanning angles. This uneven distribution can create challenges and may lead to incomplete or biased representations of the scene.

Another significant drawback is the inability of Point Clouds to handle view-dependent components of color and appearance. Due to their simplicity, Point Clouds typically store only static per-point color values, without modeling how surface appearance changes with the viewing angle or lighting conditions. This limitation results in rendered images with reduced photorealism, particularly in scenes where reflective or specular effects are prominent, making Point Clouds less suitable for high-quality visual applications such as Novel View Synthesis. To address this issue, approaches such as Plenoptic Point Clouds have been introduced, which extend the basic point-based representation by incorporating richer light field information, enabling the capture of view-dependent effects and improving rendering quality. However, the more complex representation increases the cost of communication and storage compared to standard Point Clouds [14].

2.2.2 NeRF

NeRF [5] is a recently introduced method designed to address the problem of Novel View Synthesis. It leverages volume rendering and a continuous neural scene representation, implemented through a Multilayer Perceptron (MLP), to learn the geometry and lighting of a 3D

scene.

NeRF models a scene as a 5D function that maps each spatial position (x, y, z) and viewing direction (θ, ϕ) to an emitted color $c = (r, g, b)$ and a volume density σ . This function is approximated by an MLP that takes a 5D input vector and outputs the corresponding color and density:

$$F_{\Theta}(x, y, z, \theta, \phi) \rightarrow (c, \sigma) \quad (2.1)$$

To ensure consistency across multiple views, the network predicts density using only the spatial location, while the color is predicted based on both position and viewing direction.

This model enables volume rendering of the scene. Rays are cast from arbitrary camera positions through the scene, and the MLP outputs sampled RGB values along the ray based on the volume density. To render a specific view, a ray is cast through each pixel, and the corresponding RGB color is inferred by querying the network with the appropriate 5D coordinates.

The network parameters Θ are optimized by comparing the rendered images with the ground truth images from the training set. The loss is computed using the ℓ_2 distance between the predicted and actual pixel values, and backpropagation is used to update the network weights.

Overall, NeRF represents a significant advancement in the field; however, its implicit nature introduces several challenges. First, it lacks interpretability, as it works as a black box, making it difficult to understand why certain views are rendered correctly while others are not, and thus complicating targeted improvements. More critically, NeRF is highly computationally demanding: rendering a single image requires billions of network evaluations, as the model must be queried along rays for every pixel. For this reason, training the whole network is very time consuming and this high computational cost makes practical deployment challenging.

To address these limitations, various methods have been proposed, often involving spatial data structures [15] or alternative encoding techniques [16]. Nevertheless, the implicit structure of NeRF imposes fundamental constraints that limit the extent of these improvements.

2.2.3 3D Gaussian Splatting

3D Gaussian Splatting [3] has emerged as an explicit alternative to NeRFs, offering rendering quality comparable to state-of-the-art NeRF models while providing greater control over the scene.

In this approach, the scene is represented as a set of 3D Gaussians, each defined by a mean

$\mu \in \mathbb{R}^3$ and a covariance matrix $\Sigma \in \mathbb{R}^{3 \times 3}$. Each Gaussian is described in 3D space using the following function:

$$G(x) = e^{-\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu)} \quad (2.2)$$

where x is the spatial coordinate. The Gaussians are anisotropic, meaning they take on ellipsoidal shapes, with variances differing across dimensions. Although isotropic covariance matrices (which describe perfectly symmetric Gaussians with no preferred direction) are simpler, anisotropy enables Gaussians to better model complex, direction-dependent structures. The shape and orientation of each ellipsoid are determined by the eigenvalues and eigenvectors of the covariance matrix.

In addition to shape, each Gaussian is assigned an opacity value $\alpha \in \mathbb{R}$, required for rendering, and a color, represented using Spherical Harmonic (SH) coefficients to model view-dependent appearance $H \in \mathbb{R}^{D \times 3}$, following standard practice [15]. The dimension D is given by $(l + 1)^2$, where l denotes the maximum SH order, typically set to 3.

2.2.3.1 Training Process

The training pipeline for Gaussian Splatting is illustrated in Figure 2.2.

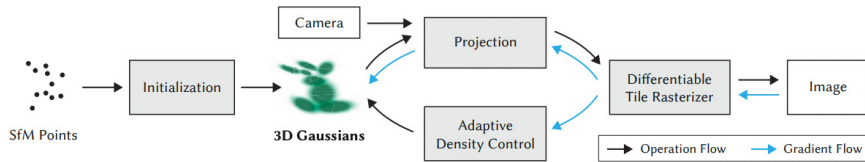


Figure 2.2: Gaussian Splatting training pipeline

Gaussian positions and colors are initialized from a sparse Point Cloud obtained from the input images, while the remaining parameters are initialized with default values. Training then proceeds along two parallel paths: one focuses on optimizing the parameters to accurately model the scene structure, while the other addresses the unknown optimal number of Gaussians required for an accurate representation. The proposed framework unifies these two goals into a single training process.

All Gaussian attributes are optimized iteratively using backpropagation. At each iteration, parameters are updated to better match the training images. The loss function combines an ℓ_1 reconstruction term and a perceptual similarity component based on the Structural Similarity Index (SSIM) [17]:

$$\mathcal{L} = (1 - \lambda)\mathcal{L}_1 + \lambda\mathcal{L}_{D-SSIM} \quad (2.3)$$

where λ is a weighting factor balancing the two components.

Because the covariance matrix must remain positive semi-definite, direct optimization is avoided since it can lead to invalid representations. Instead, it is factorized as:

$$\Sigma = RSS^T R^T \quad (2.4)$$

Here, R is a rotation matrix, represented using a quaternion $q \in \mathbb{R}^4$, and $S \in \mathbb{R}^3$ is a scaling vector that controls the ellipsoid’s dimensions. This decomposition guarantees a valid covariance matrix throughout training, with quaternion normalization ensuring proper rotations. By optimizing these components, the Gaussians can adapt to the scene’s geometry, enabling a compact and expressive representation that supports efficient optimization using Stochastic Gradient Descent techniques and high-quality Novel View Synthesis from sparse input views.

However, refining the attribute values alone may not be sufficient, as inaccurately placed geometry can lead to suboptimal scene reconstruction. Thus, the training process also involves dynamically creating and removing Gaussians to improve coverage and efficiency. Specifically, new Gaussians are introduced in regions where the existing ones are insufficient to represent the scene accurately. These areas are typically identified by high positional gradients in view-space, indicating that the optimization is actively trying to adjust these poorly reconstructed regions. These gradients serve as indicators of where the current Gaussian representation needs to be refined or expanded. As shown in Figure 2.3, this happens in two cases:

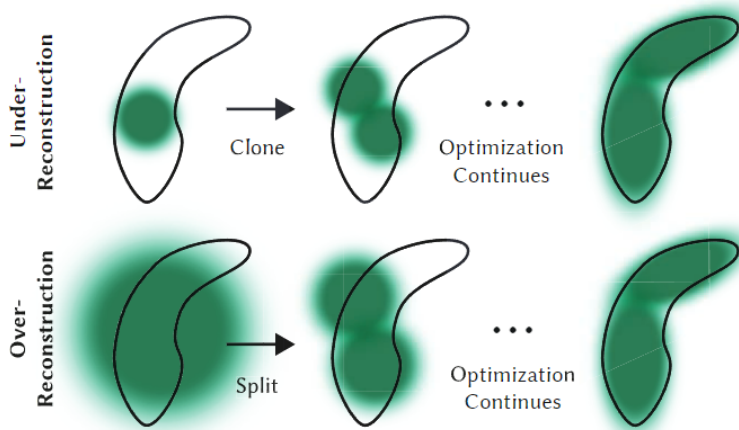


Figure 2.3: Top: duplication in under-reconstructed areas. Bottom: splitting in over-reconstructed areas

- **Under-reconstructed areas:** These regions lack sufficient Gaussian coverage. To address this, existing Gaussians are duplicated and displaced in the direction suggested by

the positional gradient.

- **Over-reconstructed areas:** These regions are characterized by too large Gaussians that fail to capture the complex structure of the scene. These Gaussians are split into smaller ones to better represent the scene’s fine-grained geometry.

Conversely, we need to remove Gaussians when they provide negligible contribution to the rendering, typically due to their transparency: every 100 iterations Gaussians whose opacity value α is under a predefined threshold are pruned.

Finally, to prevent uncontrolled growth in the number of Gaussians, an additional regularization mechanism is introduced. Since the optimization can get stuck with floaters close to the input cameras and lead to an excessive increase in the number of Gaussians, every $N = 3000$ iterations α is reduced close to zero for all Gaussians. During subsequent iterations, only those Gaussians that significantly contribute to the rendering regain opacity, allowing the model to prune redundant or non-contributing elements and maintain a compact representation.

2.2.3.2 Image Rendering

For rendering, 3D Gaussians need to be projected into 2D space. Given a viewing transformation W , the covariance matrix Σ' in camera coordinates is given by:

$$\Sigma' = JW\Sigma W^T J^T \quad (2.5)$$

where J is the Jacobian of the affine approximation of the projective transformation.

Once all 2D splats are generated, their contributions must be blended based on depth, color, and transparency to determine the attributes of each pixel and render the final image. This process of transforming the 3D scene representation into 2D images is called rasterization and is carried out using an α -blending process.

First, the screen is divided into 16×16 tiles. Then, 3D Gaussians are culled against both the view frustum and each tile. Specifically, those with less than a 99% confidence interval intersecting the view frustum are removed, and similarly, Gaussians in extreme positions are discarded because their 2D projection may cause instability.

Each Gaussian is assigned a key based on the number of tiles it overlaps and its depth in the view, and all Gaussians are efficiently sorted accordingly. After sorting, a list is created for each tile by identifying the first and last depth-sorted Gaussians that contribute to it. Note that

the entire rasterization process is performed at the tile level, with no additional reordering at the pixel level, which has been shown to introduce negligible distortion.

Each tile is processed by a separate thread block: the final color of each pixel is then computed by blending the weighted contributions of all relevant Gaussians, considering both their opacity and the influence of those in front. Specifically, the previously defined list is traversed from front to back, accumulating the contribution of each Gaussian in terms of color and opacity until the α value saturates (reaches one). The color at a pixel C_i , computed by blending the contributions of the N Gaussians influencing that pixel (indexed by n), is given by:

$$C_i = \sum_{n=1}^N T_n \alpha'_n c_n, \quad \text{with} \quad T_n = \prod_{j=1}^{n-1} (1 - \alpha'_j) \quad (2.6)$$

where α' refers to the final opacity, computed based on the learned opacity α and the 2D covariance matrix Σ' , and c_n is the view-dependent color of the n -th splat, computed from its SH coefficients H . A summary of the whole process is provided in Figure 2.4.

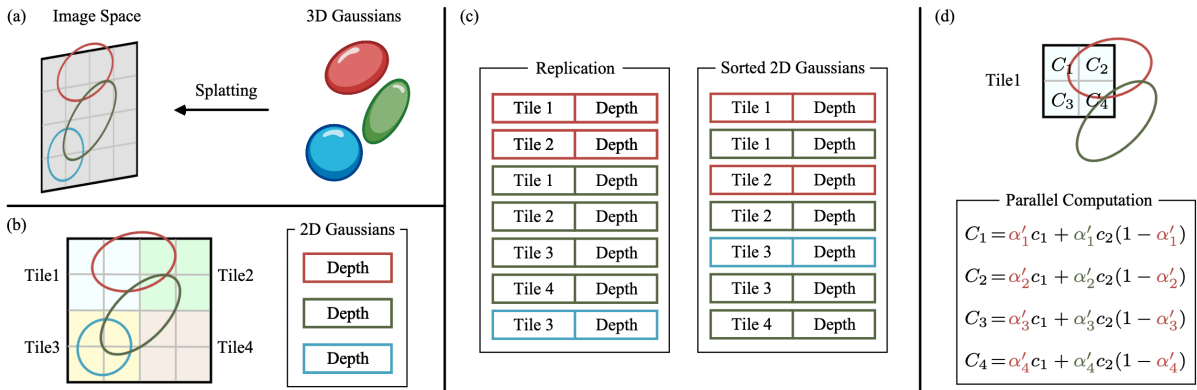


Figure 2.4: (a) From 3D Gaussians to 2D splats. (b) - (d) Rasterization process.

The proposed rasterization process is fully differentiable, meaning it allows the computation of gradients necessary for optimization. This is a key feature for tasks where parameters are adjusted to minimize a loss function during training, as in Novel View Synthesis. Moreover, it is highly efficient thanks to the tile-based processing approach: this method maximizes parallelism, allowing multiple pixels to be processed simultaneously within a tile, while threads can collaboratively manage the data sharing and processing of Gaussians.

Despite the significant advantages introduced by this method in terms of real-time rendering, fast training, and high visual quality, the 3DGS representation still requires substantial memory, posing challenges for practical applications.

The high memory usage mainly results from two factors: on one hand, the training process optimizes Gaussians solely to minimize distortion on training images, often leading to the creation of redundant Gaussians to fit all views accurately. On the other hand, the size is also influenced by the intrinsic complexity of the representation, as each Gaussian is described by 56 attributes in the standard configuration, where SH coefficients are considered up to the 3rd order.

Chapter 3

Related Works

In this chapter, we will review standard compression techniques for both general multimedia content and Point Clouds, which form the basis of 3DGS compression. Then, we will present a detailed analysis of existing solutions for 3DGS compression.

3.1 Compression in Multimedia and 3D Data

This section offers a general, high-level overview of some key concepts related to classical compression methods, without aiming to be exhaustive. The purpose is to introduce fundamental ideas that will be referenced and exploited in the subsequent chapters of this thesis.

Multimedia compression techniques refer to the set of methods and algorithms used to reduce the size of multimedia files, such as images, videos, and audio, while aiming to preserve their perceptual quality as much as possible. Reducing file size is essential for several applications, including efficient storage and faster transmission over networks. Compression enables multimedia content to be more easily shared, streamed, and archived, making it a key component of modern digital media workflows.

Compression algorithms are generally classified into two main categories, depending on whether the operations performed on the data are reversible or not.

Lossless compression aims to reduce file size without any loss of information. This is achieved by identifying patterns and redundancies in the data and encoding them more efficiently, thereby reducing the required storage space [18], [19]. In other words, when lossless compression is applied, the original data can be perfectly reconstructed from the compressed version. However, these methods typically achieve relatively modest compression ratios, often around a factor of two or three, which may not be sufficient for many practical multimedia

applications where greater compression is needed.

To achieve higher compression factors, lossy compression algorithms are employed. Lossy compression works by discarding redundant or perceptually less important information from the original data to reduce file size. While this process introduces some loss of quality, i.e. the original data can not be recovered exactly from the compressed version, the degradation is usually controlled to remain imperceptible or acceptable to users. The significant reduction in size achieved with lossy methods makes them indispensable for applications such as streaming video, image distribution, and 3D representations storage.

A key component of any lossy compression algorithm is quantization, which is the process of mapping a large set of input values to a smaller set, typically by rounding or grouping values. In multimedia compression, quantization reduces the precision of certain signal components that are less perceptually important, thereby lowering the number of bits needed to represent them. However, in raw multimedia data, many samples have similar importance, making it difficult to achieve significant size reduction through quantization alone without causing noticeable quality loss. For this reason, additional operations are applied to transform the signal into a form where its energy is concentrated in fewer components, making quantization more effective.

Transform coding is one such approach: it converts the data into a different domain (such as the frequency domain using the Discrete Cosine Transform [20], [21]) where much of the signal's energy is compacted into a small number of coefficients. These coefficients can then be quantized more aggressively, especially those that contribute less to perceptual quality.

Another common approach is predictive coding, which exploits the correlation between neighboring samples [22]. Instead of encoding absolute values, predictive coding encodes the difference between the actual value and a predicted value based on previous samples. These prediction errors are typically smaller due to the correlation between nearby samples, thereby making them easier to quantize and compress efficiently. By combining these techniques with quantization, lossy compression algorithms can achieve high compression ratios while maintaining acceptable visual or auditory quality.

3.1.1 Point Cloud Compression

Despite representing different formats for 3D scene representation, Point Clouds and 3DGS share important similarities. Both describe a 3D object or scene using a discrete, unstructured collection of elements – points in the case of Point Clouds, and Gaussians in the case of 3DGS.

As mentioned in the Introduction, 3DGS can be seen as an extension of a Point Cloud where each point carries not only its 3D position and color (RGB) but also additional attributes such as scale, rotation, opacity, and SH coefficients. Although this analogy is a simplification because Gaussians are not independent from each other, it remains helpful. It provides a conceptual bridge for understanding how compression techniques developed for Point Clouds can inspire and inform compression strategies for 3DGS.

For this reason, this section aims to provide a general overview of the main standardized Point Cloud compression methods in the literature. By understanding these foundational techniques, we can better appreciate the design choices and adaptations required for effective 3DGS compression, some of which build directly on ideas from Point Cloud compression.

The standard for Point Cloud compression has been released by Moving Picture Experts Group (MPEG) in 2021 and it includes two classes of solutions [23]:

1. Video based Point Cloud Compression (VPCC) [24]
2. Geometry based Point Cloud Compression (GPCC) [12].

3.1.1.1 VPCC

2D video compression is a mature and highly successful technology, supported by widely adopted standards and hardware implementations. To take advantage of these efficient technologies, such as H264 [25], the VPCC method projects sequences of 3D Point Clouds into 2D images, which can then be compressed using standard video codecs to remove temporal redundancy and reduce overall data size.

Specifically, VPCC first clusters Point Clouds into small patches grouped by point-wise surface normals. These patches are then projected and organized into three types of 2D image sequences: attribute images, occupancy images, and geometry (depth) images. Attribute images store color and any additional information. Occupancy images are binary maps indicating which regions are occupied in the corresponding geometry images. Geometry images encode 3D point positions as pixel values representing distances from the projection plane. Through this process, VPCC transforms an unstructured 3D representation into a structured sequence of 2D images that can be efficiently compressed with mature video coding standards.

VPCC is primarily designed for encoding dynamic Point Clouds, whereas this thesis focuses on static 3DGS compression. For this reason, VPCC will not be described in further detail here.

3.1.1.2 GPCC

GPCC is a software for static Point Cloud compression that belongs to the MPEG standard and is widely used as a benchmark due to its efficiency and reliable rate-distortion performance in a variety of 3D applications. It offers three different geometry encoding schemes:

- **Octree**: it is based on recursively subdividing the 3D space into octants.
- **Trisoup**: it relies on octree encoding, but improves the quality of the compressed object through local surface fitting.
- **PredGeom**: it encodes the current point using a prediction scheme based on previously processed points.

Regarding attribute encoding, two compression modes are available:

- **Predictive/Lifting Transform**: it divides points into Levels of Detail (LoD) and uses neighboring points from the previous LoD to predict the attributes of the current point.
- **RAHT**: it follows the Octree subdivision, but traverses the resulting tree in a bottom-up fashion, starting from individual voxels up to the root. Attribute encoding at each node is based on predictions from the already visited child nodes.

GPCC supports both lossy and lossless modes for both geometry and attributes, making it flexible for various applications with different accuracy and bitrate requirements.

3.2 3DGS compression

While 3DGS demonstrates remarkable capabilities, its scalability poses significant challenges, particularly when compared with NeRF-based methods. In the latter, only parameters of the learned MLP need to be stored, while for 3DGS all Gaussians with the corresponding attributes must be saved to be able to render new images. This scalability issue becomes particularly critical when dealing with large-scale scene management, where the computational and memory demands increase substantially, or when working with resource constrained devices characterized by limited memory capabilities. Consequently, significant effort has been made to reduce memory usage in both model training and storage, thus facilitating the practical deployment of such method.

In the literature, different compression techniques have been proposed, as shown in [26], [27], and they generally fall into three main design principles: post-training methods, training optimization methods and retraining methods.

3.2.1 Post-training Methods

Post-training methods operate by analyzing the numerical values of model parameters, such as attributes or Gaussians themselves, to identify and discard those deemed less important. The core idea is that parameters with minimal impact on the quality of the rendered images can be either removed or approximated to reduce redundancy. This enables compression through techniques like thresholding, where low-value parameters are pruned, or clustering, where similar parameter values are grouped and replaced with shared representatives. While these strategies effectively reduce the size by keeping only the most significant information, they often result in noticeable alterations to the original values. As a consequence, the performance may degrade, requiring a subsequent finetuning step to restore lost information and maintain overall fidelity.

In the following, a detailed description of all most commonly used techniques is provided.

3.2.1.1 Gaussian Pruning

Gaussian Pruning techniques aim to identify and eliminate redundant Gaussians that contribute little to the scene representation, such as those that are overly large, highly transparent, or carry overlapping information. This compaction strategy is motivated by the fact that the baseline 3DGS method often generates a highly redundant set of Gaussians, resulting in a final representation composed of millions of elements — many of which offer little or no meaningful contribution. This redundancy arises because, during training, the model is penalized solely based on the distortion in the training images. As a result, it tends to overfit by introducing a large number of Gaussians to accurately match all training views.

The key point when performing Gaussian pruning is to identify which Gaussians can be eliminated with little or no reduction in the quality of rendered images. Several approaches have been proposed in the literature, and they can be broadly categorized into two groups:

- **Importance-based scoring systems:** This category of methods focuses on assigning an importance score — or, equivalently, a redundancy score — to each Gaussian based on its contribution to the rendering process. In [28], a scale- and resolution-aware strategy is introduced for removing redundant primitives: for each Gaussian g , the number of overlapping primitives within a spherical region centered at g is counted. A redundancy score is then assigned considering both the number of overlaps and whether the Gaussian covers a region not redundantly populated, where its contribution may be critical.

However, the most common approach in the literature ([4], [11], [29]) involves computing an importance score for each Gaussian. It is defined as the product of two terms: a view-dependent score I_d , reflecting the Gaussian’s relevance in comparison to overlapping Gaussians and a view-independent score I_i , derived from the Gaussian’s volume. Finally, the Gaussians with the smallest score (based on the combined score $I_d \cdot I_i$) are pruned.

- **Dynamic masking methods:** This class of approaches employs dynamic binary masks during training to iteratively eliminate Gaussians based on their contribution. Specifically, a learnable binary mask is associated with each Gaussian and is used to progressively suppress those that contribute minimally to the rendering process. Gaussians are not immediately discarded but are instead gradually deactivated through the mask, which is updated at each training iteration. Final pruning is performed only at the end of training. To promote sparsity, an additional regularization term is typically added to the loss function. For example, in [30], the following formulation is adopted:

$$\mathcal{L}_{GSprune} = \frac{1}{N} \sum_i \phi_i^{\text{soft}} \quad (3.1)$$

where ϕ_i^{soft} is the learnable soft mask, from which the actual hard mask ϕ_i^{hard} is obtained through thresholding.

Similar strategies are also presented in [9], [31], and [32].

3.2.1.2 SH Coefficient Distillation

As mentioned in the description of 3D Gaussian Splatting (Section 2.2.3), one of the key limitations of the proposed method is the inherent complexity of the representation, with each Gaussian characterized by several attributes. Figure 3.1 shows the composition of each Gaussian, highlighting the dimension of each attribute and its percentage in size. It is evident that more than 75% of the parameters are represented by high-order SH coefficients (orders 1–3 in the standard configuration). Despite accounting for the majority of Gaussian attributes, they contribute only slightly to the final rendering, especially in regions with low photogrammetric complexity.

Specifically, while DC SH coefficients carry the information related to the average color values, the high-order AC coefficients are used to model view-dependent color features. These are necessary mainly to describe surfaces with complex light interactions, such as reflective or

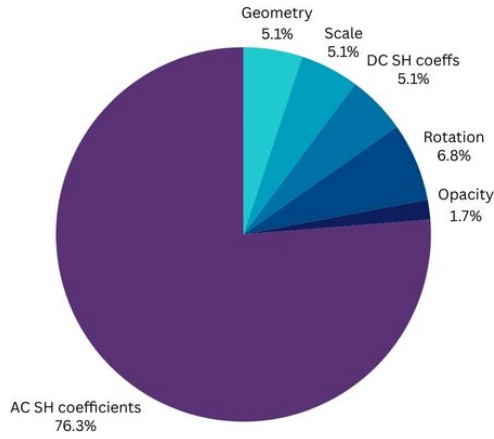


Figure 3.1: Composition of each Gaussian

refractive effects, whereas simpler surfaces can be effectively described with fewer coefficients. As an example, consider a fully diffusive point: it can be adequately represented using only the DC coefficients, as its color has no view-dependent component.

Based on this, different approaches in the literature aim to use only the minimum number of coefficients required to achieve a good representation. They do this by estimating the complexity of the color representation and accordingly determining the number of SH coefficients needed. In essence, the goal is to allocate high-order coefficients only where they are truly necessary, for instance, on surfaces with complex, view-dependent lighting effects, while relying solely on lower-order (or even just DC) coefficients in simpler, more uniform regions. This adaptive allocation significantly reduces the overall memory requirement without compromising visual quality, as it avoids wasting parameters on regions where the extra complexity brings negligible benefit.

In [28], they propose to evaluate whether low-order coefficients are sufficient for a good representation by analyzing the SH function across all input views. If the evaluated color varies significantly across different views, this indicates that the primitive exhibits view-dependent effects, and its representation must therefore include higher-order coefficients. This is done by computing, for every viewpoint, the per-channel average μ_c and standard deviation σ_c of the color c_i for each primitive, weighted by average transmittance:

$$\mu_c = \frac{\sum_i^N c_i \bar{T}_i}{\sum_i^N \bar{T}_i}, \quad \sigma_c = \frac{\sum_i^N (c_i - \mu_c)^2 \bar{T}_i}{\sum_i^N \bar{T}_i} \quad (3.2)$$

where N is the number of views containing the primitive, and \bar{T}_i is the average transmittance

of the primitive over the pixels splatted by view i , defined as:

$$\bar{T}_i = \frac{\sum_{k=1}^P T_{ik}}{P} \quad (3.3)$$

Here, P is the number of pixels the primitive is splatted onto in view i , and T_{ik} is the transmittance of the primitive for a specific pixel k . High-order coefficients for primitives with a variance σ_c below a predefined threshold are finally pruned.

A different approach is proposed in [30], where each Gaussian is associated with a dynamically learned mask per SH order. As described for Gaussian pruning, higher-order components are progressively nullified during training, while they are only definitively removed at the end. An additional loss term, which promotes sparsity by encouraging higher-order masks to be zero, is included for efficiency purposes.

The main drawback of all these approaches is the increased complexity of the representation, arising from the fact that not all points are characterized by the same number of coefficients. In [30], this issue is partially addressed by clustering Gaussians with similar SH masks, so that it is no longer necessary to transmit a separate mask for each primitive. This is feasible because Gaussians are order-invariant.

3.2.1.3 Geometry Compression

Geometry compression refers to compressing the Gaussian position vector (x, y, z) . While this reduces the data size, it not only introduces inaccuracies in the positions of Gaussians but also often causes different primitives to be merged when their compressed positions become identical. As a result, geometry compression does not just affect location accuracy but it also impacts attribute quality, since merging requires averaging the attributes of combined Gaussians. Because of this, most methods in the literature avoid compressing Gaussian coordinates [8], [33]: the quality loss is too high compared to the small rate savings, especially since only 3 out of 59 parameters are affected. Some approaches [30], [31], instead, simply reduce the precision in the representation, moving from `float32` to `float16`, as this is shown to have almost no impact in the final rendering quality.

Only a few attempts have been made to effectively compress the geometry. Inspired by Point Cloud compression techniques, [4] and [11] propose applying octree coding. They first convert the unstructured Gaussian Splatting representation into a structured voxel grid, then partition the 3D space hierarchically into octants. The occupancy of each region is encoded

using 8 bits, one for each child node.

3.2.1.4 Attribute Compression

Attribute compression refers to the compression of all other Gaussian parameters, except the three mean coordinates, and is a critical yet challenging task in the context of Gaussian Splatting. Unlike Point Clouds, which typically store only three attributes (the RGB color components) and thus balance geometry and attribute contributions similarly in terms of overall data size, 3DGS representations carry a much larger number of attributes per primitive. This increase means that attributes, rather than geometry, become the dominant factor affecting the total size of the representation. Therefore, developing an efficient method for compressing these attributes is essential to significantly reduce the overall storage and make the representation more practical for real-world applications.

Different approaches have been proposed in the literature to reduce the storage requirements of Gaussian attributes, using both lossless and lossy strategies. Below, we provide a detailed description of the most common and widely adopted methods:

- **Preprocessing:** Before applying actual compression, some methods first remap certain attributes into formats that are more compression-friendly — meaning formats that either reduce inter-channel correlation or lower the number of channels to compress. For example, [11] converts color channels from RGB to YUV, a format often preferred because the luminance component is much more perceptually significant than the chrominance components.

An alternative is proposed in [4], which transforms rotation quaternions ($q \in \mathbb{R}^4$) into Euler angles ($e \in \mathbb{R}^3$), effectively reducing the number of attributes per Gaussian by one without sacrificing quality since the transformation is fully reversible.

Lastly, [9] replaces the SH coefficients used for view-dependent color with a hash grid and a tiny MLP. Specifically, positions are fed into the hash grid, and the resulting feature, along with the view direction, is input into the MLP. This setup allows the model to exploit spatial redundancy by reusing similar colors from neighboring Gaussians, making the final representation much more efficient since only the MLP’s weights need to be stored.

- **Vector Quantization:** Vector quantization is a widely used strategy in 3DGS compression, aiming to reduce data complexity by grouping similar feature vectors and representing each group with a common codeword. The original high-dimensional space

is partitioned into clusters, typically using algorithms such as K-Means [34] or Linde-Buzo-Gray (LBG) [35], which iteratively assign features to their nearest cluster center and update the centers to minimize intra-cluster distance. Compression is then achieved by replacing each feature vector with the index of its corresponding codeword in the learned codebook.

Methods in the literature differ mainly in the selection of attributes to quantize: [36] and [33] apply vector quantization to all attributes, while [4] and [29] limit it to AC SH coefficients, which tolerate coarser approximations with minimal impact on visual quality. Some approaches further refine the clustering strategy; for instance, [36] uses sensitivity-aware K-Means to better preserve high-impact Gaussians.

- **Transform Coding:** To the best of our knowledge, very few works apply transform coding to map Gaussian features to a different domain where the signal becomes sparser, and none of them adopt this strategy in a systematic manner. [4] exploits the RAHT, originally introduced for Point Cloud compression, to encode key attributes such as opacity, rotation, and DC SH coefficients. Similarly, [11] uses the same transform to losslessly compress anchor attributes. More details about the mathematical formulation of this transformation are provided in Chapter 4.
- **Predictive Coding:** Some approaches introduce anchor points that serve as proxies for predicting the properties of associated Gaussian primitives. By deriving attributes from these anchors instead of storing them individually for each primitive, redundancy is reduced and memory efficiency is improved. In [11], the 3DGS is partitioned into blocks using a KD-tree, and Furthest Point Sampling is employed to select anchor primitives at multiple LoDs within each block. Non-anchor primitives predict their attributes using the k nearest anchors, with quantized residuals transmitted instead of full attributes. Conversely, [37] performs voxel down-sampling to define anchors, each of which is linked to K nearby primitives. In this case, inter-primitive prediction is used to infer the attributes of the coupled Gaussians from their respective anchor.
- **2D-based Compression:** Given the efficiency of 2D image and video compression techniques, some works have explored mapping the unstructured Gaussian Splatting representation onto structured 2D grids (or sequences of grids), enabling the use of standard codecs. In [38], spatial redundancy is leveraged to define such grids: since nearby Gaussians in a natural scene often share similar attributes, it is possible to organize them

into 2D grids (one per attribute) so that similar Gaussians are placed close to each other. This produces smooth grids that are well-suited for conventional image compression algorithms. In contrast, [39] targets video codecs by converting the unstructured 3DGS into a structured sequence of frames using a tri-plane representation, allowing standard video compression techniques to be applied effectively.

3.2.1.5 End-to-end Compression

As described in the previous subsections, frameworks based on traditional compression techniques are typically characterized by a sequence of blocks whose design and optimization are carried out independently. In contrast, an end-to-end compression method refers to a system – typically a neural network – in which the entire compression pipeline is jointly learned and optimized as a whole.

In the context of Gaussian Splatting, to the best of our knowledge, the only end-to-end compression method has been proposed in [10]. This work was motivated by the limitations of traditional approaches: to preserve the quality of rendered images, they typically require per-scene finetuning, which makes the compression process slow. As a result, the main goal of this method is to achieve a good rate-distortion trade-off without requiring scene-specific finetuning, significantly improving compression time efficiency. The entire pipeline can be divided into the following blocks:

- **Autoencoder-based structure:** Maps features to a latent space where quantization introduces less distortion compared to direct quantization, similarly to what is proposed in [40] for images. Geometry attributes are not mapped to the latent space, as this leads to significant quality degradation.
- **Multi-path entropy module:** Defines a mask to decide whether each attribute should pass through the autoencoder or be quantized directly, depending on what is most effective.
- **Inter-gaussian context model:** Partitions Gaussians into N^2 batches, where N is an hyperparameter of the system. For each batch, previously decoded ones are used to build a grid. Feature vectors associated with voxels in the grid are then used to predict the mean and standard deviation of a Gaussian distribution for each encoded point.
- **Intra-gaussian context model:** Splits each feature vector into chunks and uses previously decoded chunks to predict the distribution of the current chunk, improving intra-

Gaussian entropy modeling.

- **Gaussian Mixture Module:** Combines the inter-Gaussian, intra-Gaussian, and hyperprior-based distributions to define a final probability model for each latent, enabling more accurate entropy estimation.

3.2.2 Training Optimization Methods

All methods presented in Section 3.2.1 aim to remove redundant information that arises during the training process. In contrast, training optimization methods tackle the root of the problem directly by modifying the training loop itself to reduce the redundancy of the generated representation. All these methods propose hybrid approaches between the two main categories: unlike post-training methods, they modify the training loop and therefore cannot be applied to compress pre-trained representations; however, unlike full retraining methods, they preserve compatibility with the baseline representation. Typically, this is done by introducing an additional term into the loss function to optimize the representation.

For example, in [28] and [33], an ℓ_1 sparsity term on opacity is added to the loss, weighted by a hyperparameter λ . Since Gaussians are pruned based on their opacity value α , including this extra term makes the pruning procedure more effective, leading to a final representation with fewer primitives.

Alternatively, some approaches propose a coarse-to-fine training strategy. The key idea here is that Gaussians are initially placed in a sparse configuration, which prevents them from capturing fine details right away, resulting in a higher initial loss. However, starting from coarse image synthesis helps regularize the creation and deletion of Gaussians, ultimately reducing artifacts and producing a more compact representation. This can be implemented in several ways: for example, [8] initially renders at a low scene resolution and gradually increases the rendered image size until reaching full resolution. Similarly, [31] uses Gaussian blurring filters to reduce image detail, starting with a 9×9 kernel and $\sigma = 2.4$, progressively lowering them over time.

Finally, another strategy is to explicitly include a measure of rate in the loss function. As mentioned earlier, redundant Gaussians are mainly generated because optimization focuses solely on minimizing distortion in the training images. By adding an extra rate term to the

loss, the creation of unnecessary Gaussians is discouraged. In [37], for example, an embedding is associated with each primitive, and neural networks estimate the probability distributions over these embeddings. The rate is then estimated as:

$$R_{f_\omega} = \mathbb{E}_\omega \left[-\log p(\tilde{f}_\omega) - \log p(\eta_f) \right] \quad (3.4)$$

where \tilde{f}_ω is the feature embedding and $p(\eta_f)$ is the estimated probability of the hyperpriors.

3.2.3 Retraining Methods

Unlike post-training methods, which aim to improve compression efficiency by processing an already trained representation, and unlike training optimization methods, which modify the training loop without altering the final representation format, retraining methods introduce entirely new training frameworks. The goal is to generate a more compact and structured representation from the outset, rather than optimizing it afterward.

Specifically, these methods focus on leveraging the inherent structural information and spatial relationships within the scene to guide the organization of Gaussian primitives in a more efficient manner. By embedding these constraints and insights directly into the learning pipeline, the resulting representation tends to be more coherent, exhibits less redundancy, and is more suitable for compression.

These methods are particularly promising because they allow compression to be considered directly during the learning phase, potentially leading to more compact and efficient representations without the need for extensive post-processing. However, they also introduce greater complexity, as they require the design of entirely new training pipelines, often involving custom architectures, loss functions, and optimization strategies. This makes them more computationally intensive. As a consequence, this direction is not fully explored, and there remains significant room for innovation.

3.2.3.1 Scaffold-GS

Scaffold-GS [1] proposes a novel training framework to address key limitations of standard 3DGS. One of the main issues with the baseline approach is its tendency to overfit by minimizing reconstruction loss across all training images. This results in a representation that is overly dense, i.e. composed of a very large number of Gaussians, which makes it less efficient for practical deployment and limits its generalization capability to novel viewpoints that differ significantly from the training views.

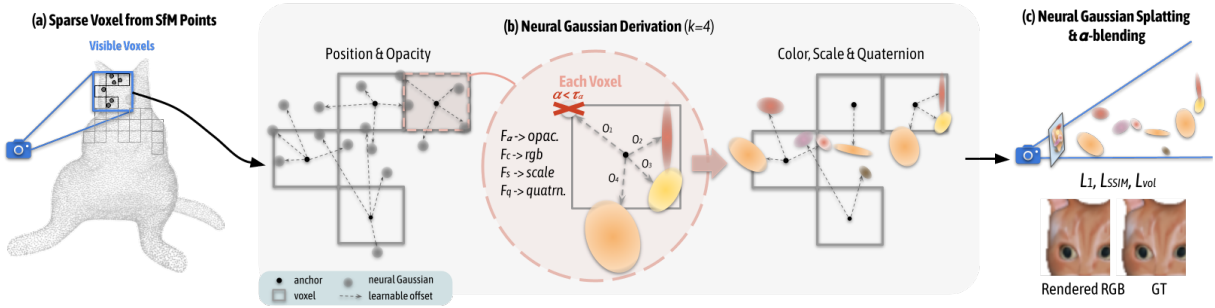


Figure 3.2: Scaffold-GS pipeline [1]

To mitigate these problems, Scaffold-GS introduces a system that starts from a sparse Point Cloud, as in the original 3DGS pipeline, and identifies a set of anchor points. Each anchor point v is associated with a local context feature $f_v \in \mathbb{R}^{32}$, a scale vector $l_v \in \mathbb{R}^3$, and a set of learnable spatial offsets $O_v \in \mathbb{R}^{k \times 3}$, where k is the number of neural Gaussians predicted per anchor. During rendering, for each anchor within the viewing frustum, k Gaussians are instantiated on the fly, with their positions determined by the offsets. Their remaining attributes, opacity, color, rotation (quaternion), and scale, are predicted using dedicated MLPs conditioned on the anchor features. This design enables dynamic generation of only the necessary Gaussians during rendering, reducing both memory usage and computational time.

The learnable parameters are optimized based on a loss function that includes, as for the original 3DGS loss, a distortion term related to both the Mean Square Error (MSE) and to the SSIM but, in addition, it considers also a regularization terms encouraging compactness and spatial separation of Gaussians. Finally, a dynamic anchor management strategy is introduced: regions with high cumulative gradient magnitude are enriched with new anchors because they are considered to be significant, while primitives with low cumulative opacity are pruned to reduce redundancy. A summary description of the proposed system is shown in Figure 3.2.

Overall, Scaffold-GS achieves comparable or superior rendering quality to standard 3DGS while producing a significantly more compact representation. This is primarily due to the reduced number of stored primitives and the fact that most Gaussian attributes are computed on demand rather than stored explicitly.

3.2.3.2 Scaffold-based methods

As highlighted in the previous section, Scaffold-GS [1] introduces an approach that exploits spatial relationships between nearby Gaussians by introducing anchor points. These anchors enable related primitives to be clustered, with their attributes inferred from a common reference, significantly reducing redundancy and yielding a more compact scene representation.

However, this approach focuses solely on the definition of a more convenient representation and does not address the design of an efficient coding scheme. In its original form, it treats anchors as independent entities, resulting in a sparse and unstructured set that is challenging to encode effectively. To overcome these limitations, several subsequent methods have been proposed to build upon the Scaffold-GS framework. By introducing dedicated coding and quantization strategies, these advances further reduce the memory footprint of the resulting scene, making it more suitable for practical deployment and large-scale applications.

HAC [6] takes Scaffold-GS as a baseline, aiming to further reduce its storage and bandwidth requirements by exploiting spatial correlations within the scene. Its core idea is to replace the direct storage of individual anchor features with a hash grid: the hash grid partitions 3D space into discrete cells, and each cell is associated with a hash function that allows for efficient indexing and retrieval of nearby information. By mapping Gaussian positions into this structure, HAC captures rich spatial context that can be used to encode attributes compactly. While a direct replacement of anchor features with hash grid features leads to significant performance loss, the authors observe that hash grids still encode valuable mutual information. To leverage this, HAC uses hash grid features as a context: this allows for adaptive quantization and more effective entropy coding, yielding significant storage savings.

More precisely, the method defines two nearly identical MLPs that take hash features as input. The first predicts a refinement term for adjusting the quantization step per attribute, given the different precision required by each of them, while the second predicts the mean and variance of a Gaussian distribution for anchor attributes, enabling more effective entropy coding.

Finally, HAC incorporates an adaptive offset masking strategy: an additional loss encourages the removal of Gaussians with negligible opacity or volume, allowing the method to mask as many redundant primitives as possible. With the same goal, an anchor pruning scheme removes any anchors that no longer have associated Gaussians, yielding a more compact and efficient final scene representation.

In a similar way, CAT-3DGS [7] proposes a compact representation that serves as context for entropy coding of Gaussian attributes. The approach adopts a multi-resolution triplane decomposition of the scene: rather than coding the whole 3D space, three planes (xy , xz , yz) are used, and each cell in these planes is associated with a feature vector. To compute a feature for any point in space, its projections onto the three planes are taken, and the corresponding

features are interpolated. However, as in [6], the triplane is used not as a direct source of features, which would degrade performance, but as a hyperprior context.

The triplane information is therefore used to estimate the probability distribution of anchor attributes, which are encoded via an arithmetic encoder. In this scheme, offset and scale are coded directly with this method. Meanwhile, the feature vector is split into slices: the first slice is encoded using the triplane hyperprior, and subsequent slices are encoded using an autoregressive model that conditions on both the triplane hyperprior and the previously encoded slices, thereby capturing both inter-Gaussian and intra-attribute correlations.

Since the triplane itself must be stored, CAT-3DGS applies a similar approach to encode it, treating each channel with an autoregressive model where the probability of an element is conditioned on those previously coded, yielding a Laplacian distribution.

Finally, this approach introduces also a frequency-aware masking scheme for pruning, extending dynamic masking approaches by preserving Gaussians that are used more frequently during rendering. In this way, important scene details are preserved while achieving a more compact and efficient final representation.

One final approach built upon the Scaffold-GS framework is ContextGS [32]. This method is based on the insight that, despite the Scaffold-GS structure is significantly reducing spatial correlations, certain areas still contain highly similar information that can be leveraged to further reduce the overall storage requirement. To this end, ContextGS introduces an autoregressive model at the anchor level. The pipeline can be summarized as follows:

- **Anchor partitioning strategy:** The anchors are divided into K disjoint levels, where level $K - 1$ is the coarsest and level 0 is the finest. Each level is designed to cover the scene almost uniformly and to be a downsampled version of the preceding one. This is achieved via a bottom-up scheme: lower-level anchors are quantized with a fixed voxel size, and the anchor for the next level is chosen from those within the same voxel (selecting the one with the lowest index). A target ratio between the number of anchors at level k and $k + 1$ is fixed to control the hierarchical sparsity.
- **Context modeling across anchor levels:** To encode the anchors efficiently, an entropy coding scheme is used where the conditional probability distribution of an anchor at level k is estimated based on its position and the feature and scale of the corresponding anchor at the coarser level $k + 1$. Similar to image compression, an MLP predicts this conditional distribution.

- **Hyperprior:** To further reduce redundancy, ContextGS introduces a learnable hyperprior vector z_i^k for each anchor. This vector captures global statistical information about the feature distribution, providing an additional prior that guides the entropy coding. The final prior for coding the feature of an anchor v_i^k is defined as a combination of its position, its parent anchor's feature and scale, and the learned hyperprior.
- **Masking:** To further reduce the total number of Gaussians, ContextGS adopts a dynamic masking approach. The overall loss is comprised of three terms: the standard distortion loss used in the original Scaffold-GS approach, a sparsity-inducing loss that encourages pruning of insignificant Gaussians, and an entropy loss that accounts for the coding cost of both the hyperprior features and the anchor-level feature representations.

Overall, this approach does not apply direct predictive coding, as features of finer anchors are not deterministically inferred from coarser anchors. Instead, it predicts their statistical distribution, making it possible to leverage this information for highly efficient entropy coding.

Chapter 4

Proposed Method

As highlighted in Chapter 1, the goal of this thesis is to develop a novel compression framework for Gaussian Splatting, drawing inspiration from traditional compression techniques — particularly those used in Point Cloud compression. In this chapter, we present the main components of the proposed system, explaining the motivations behind each design choice and discussing any challenges or limitations that led us to discard alternative approaches.

4.1 Signal Analysis

We begin by analyzing the baseline approach proposed in [3] to identify exploitable properties that could enable a more efficient representation. In general, compression techniques aim to transform a signal into a domain where it becomes sparser, thereby allowing for more compact storage with minimal quality degradation. A key strategy is to leverage the correlation among neighboring samples to reduce redundancy. For instance, in image compression, the similarity between adjacent pixels is a fundamental property exploited by nearly all algorithms. To apply a similar strategy in the context of Gaussian Splatting, we must examine the structure of the final representation. However, this task is far from trivial: unlike other representations, the Gaussian Splatting format is entirely data-driven. This means that any assumption about spatial or structural coherence must be empirically validated, as the learned representation may not directly reflect the actual geometry or redundancy of the scene.

4.1.1 Inter-point Correlation

We begin by analyzing each channel independently to assess whether nearby Gaussians exhibit similar values — i.e., whether there is local correlation that could be exploited for com-

pression. To do so, we apply the K-Nearest Neighbors (KNN) algorithm to identify neighboring points for each Gaussian. Then, for each attribute, we compute the variance within these local neighborhoods (local variance) as well as across the entire set of points (global variance). It is important to note that local variance alone does not provide a complete picture: different attributes can have vastly different value ranges. Therefore, comparing local variance to global variance allows us to normalize our observations and make meaningful comparisons across attributes.

Table 4.1: Local Variance ($k = 100$)

Attribute	Avg. Var.	Var. of Vars.
DC coeffs	0.596	0.11
AC coeffs	0.0028	1.02×10^{-6}
Opacity	3.76	0.59
Scale	1.97	0.47
Rotation	0.044	8.02×10^{-5}

Table 4.2: Global Variance

Attribute	Avg. Var.	Var. of Vars.
DC coeffs	1.783	0.53
AC coeffs	0.004	2.16×10^{-6}
Opacity	5.078	0.67
Scale	4.428	2.31
Rotation	0.07	0.0004

As an illustrative example, Tables 4.1 and 4.2 report the results for the *Garden* scene from the Mip-NeRF360 dataset. For clarity, channels belonging to the same attribute (e.g., position, opacity, spherical harmonics) are grouped together. As expected, the most strongly correlated attributes are the DC SH coefficients, which capture the average color of each Gaussian. This behavior is analogous to that observed in Point Clouds, where neighboring points often share similar colors because they belong to the same surface or object. More generally, we observe that all attributes exhibit significantly lower local variance compared to global variance. This indicates a consistent degree of local correlation and supports the idea that applying a transformation — such as RAHT — to exploit this redundancy is a reasonable strategy. Interestingly, the AC SH coefficients show both low local and global variances. This suggests that they may contribute only marginally to the final rendered image, which is an observation that will be considered in later design choices.

4.1.2 Intra-point Correlation

Unlike other multimedia formats — where each element is typically associated with only a small number of channels, such as the three RGB components — in the 3DGS representation, each point is associated with a much higher number of attributes (56 in total). Given this, it becomes worthwhile to investigate whether these attributes are correlated with one another. If strong correlations exist between different channels, they could potentially be exploited to

reduce redundancy and improve compression efficiency by applying joint processing, rather than treating each channel independently.

In the previous section, we observed that applying the RAHT transform is an effective way to exploit spatial correlation across neighboring points. Building on that, we now analyze intra-point correlation after the RAHT transform has been applied. Specifically, we compute the pairwise Pearson correlation coefficients between all channel pairs.

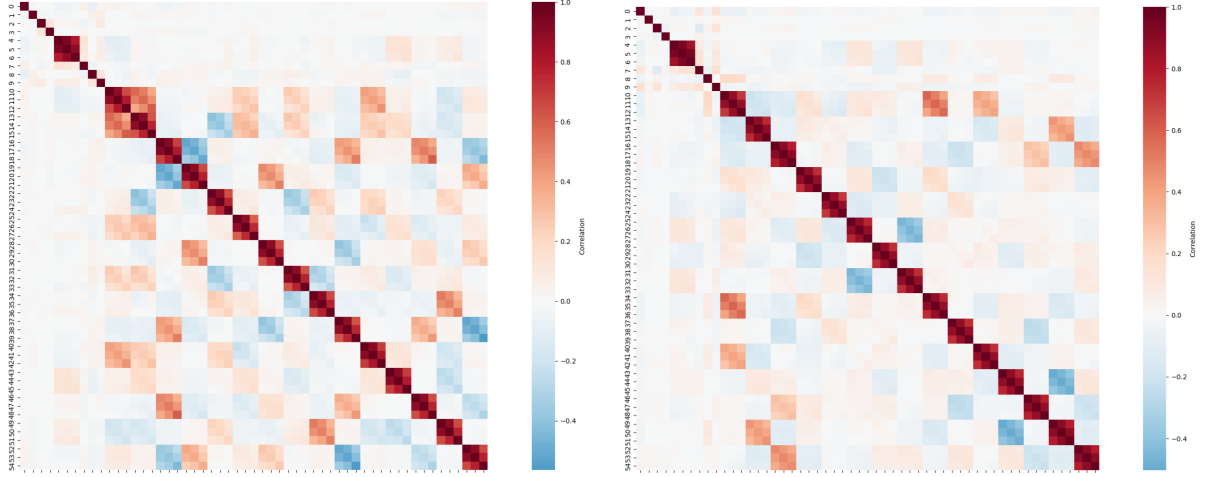


Figure 4.1: Correlation Matrix for *Room* (Mip-NeRF360) and *Train* (Tanks and Temples).

As an example, Figure 4.1 presents the resulting correlation matrices for two scenes: *Room* from the Mip-NeRF360 dataset and *Train* from the Tanks and Temples dataset. As expected, a strong correlation emerges between the corresponding SH coefficients for the red, green, and blue channels. This is visually evident from the 3×3 dark red blocks along the diagonal of the heatmaps, indicating high correlation between SH coefficients of different color channels.

However, aside from these color-related components, the majority of other attributes show only weak or scene-dependent correlations. These results suggest that while there is some redundancy among certain attributes, most other channel pairs do not exhibit significant interdependence. As a result, in this work we choose to ignore intra-point correlation and treat each attribute independently during compression. This simplifies the system design and avoids unnecessary complexity, while still preserving most of the compression potential derived from exploiting inter-point correlation.

4.1.3 Impact of AC SH coefficients

As shown in Figure 3.1, SH coefficients account for more than 75% of the total data, meaning that significant compression gains could be achieved either by removing irrelevant coefficients

(a process referred to as SH coefficient distillation) or by finding a more efficient representation for them. In order to do so without introducing a notable loss in rendering quality, it is necessary to analyze two aspects: (i) the actual importance of higher-order SH coefficients, and (ii) the correlation between these coefficients and the photogrammetric complexity of the scene.

Table 4.3: PSNR on training set for the baseline 3DGS [3], evaluated under two configurations: (i) with the maximum SH order set to 0, and (ii) with the maximum SH order set to 3.

Scene	PSNR (i)	PSNR (ii)
<i>Room</i>	34.63	35.36
<i>Train</i>	23.91	24.95

Regarding the first point, we assess the impact of removing high-order SH coefficients by comparing two configurations: one using the standard maximum SH order of 3 (i.e., including all coefficients), and another keeping only the DC components (i.e., maximum SH order equal to 0). Table 4.3 reports the corresponding PSNR values for the scenes *Room* (Mip-NeRF360) and *Train* (Tanks and Temples). The results show a minimal degradation in quality — an average PSNR drop of just 0.885 dB — indicating that high-order coefficients have limited influence on the final rendering and suggesting that even a coarse approximation of these components may be sufficient.

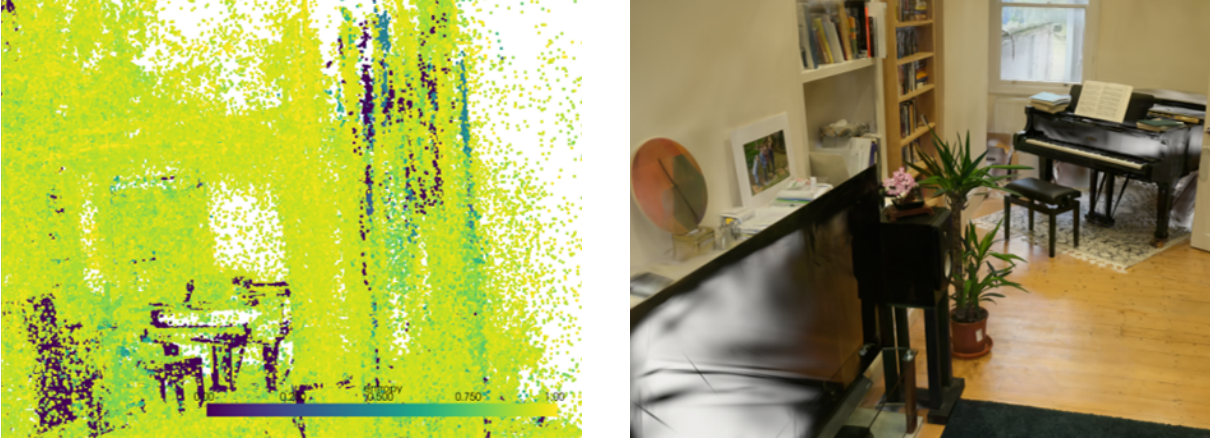


Figure 4.2: Entropy of AC SH Coefficients of scene *Room* (Mip-NeRF360).

For the second point, we explore the intuition that SH coefficients should primarily capture view-dependent color variations. Consequently, one might expect regions with higher photogrammetric complexity (e.g., reflective or specular surfaces) to exhibit stronger and more diverse SH coefficients, while diffuse (Lambertian) surfaces should contribute little to higher-order terms. If the 3DGS representation accurately encodes this, then a variable bit allocation

strategy could be applied based on the local complexity of the scene.

To investigate this hypothesis, we compute the entropy of the AC SH coefficients across the scene: high entropy would indicate high complexity, while low entropy would suggest that only few coefficients are needed. Figure 4.2 visualizes the entropy map for the *Room* scene, alongside a corresponding view of the 3D model. Contrary to expectations, the entropy does not exhibit clear correlation with visually complex areas, suggesting that the 3DGS representation does not consistently encode photometric complexity in a spatially coherent manner. This observation reinforces the idea that, due to its fully data-driven nature, the representation may be suboptimal from a compression perspective, complicating attempts to design content-aware encoding strategies.

4.2 Proposed Pipeline

Based on the insights gained from the signal analysis, we design the proposed compression framework, which is outlined in Figure 4.3.

The pipeline consists of six main steps: *Gaussian Pruning* removes redundant Gaussians produced during 3DGS generation [3]. Gaussian means are then voxelized and encoded using Octree coding. After this initial *Geometry Compression*, attributes of merged Gaussians are averaged, and their dimensionality is reduced. In the *Attribute Compression* step, each attribute is processed independently: a transform is applied and the resulting coefficients are quantized with a distinct quantization parameter chosen per channel based on its energy and significance. Prior to this step, a short *Fine-tuning* phase is introduced to help balance the RD trade-off and improve overall compression efficiency. Finally, both geometry and attribute data undergo an *Entropy Coding* stage to produce the final bitstream.

In the following sections, each step of the pipeline is described in detail, along with the rationale behind the corresponding design choices.

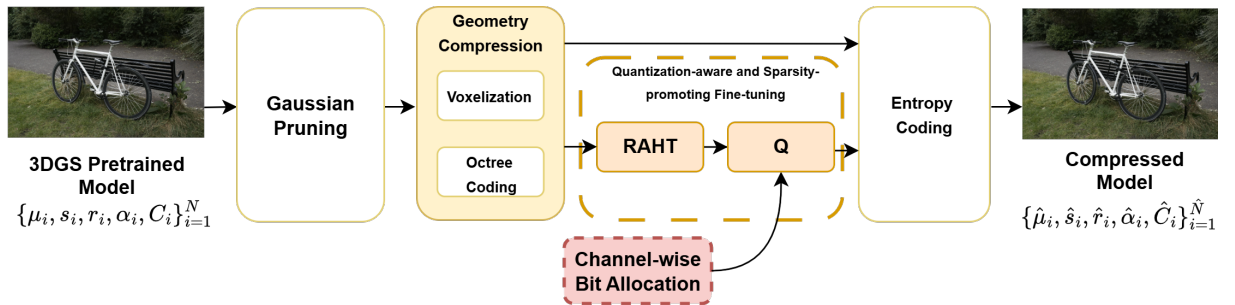


Figure 4.3: Overview of the proposed system.

4.3 Pruning

As discussed in Section 3.2.1.1, nearly all Gaussian Splatting compression methods incorporate a pruning step to address the inefficiencies of the baseline 3DGS. Since the standard pipeline often generates a large number of redundant Gaussians, substantial compression can be achieved by discarding those primitives that contribute minimally to the final rendered images.

Extensive research has already been dedicated to designing effective pruning strategies. Therefore, rather than proposing a new method, this work adopts the widely-used importance-based pruning approach found in the literature [4], [11], [29]. In this strategy, each Gaussian is assigned an importance score, computed as the product of two components:

$$I_g = I_d \cdot I_i \quad (4.1)$$

The first term, I_d , is a view-dependent score that quantifies a Gaussian’s significance relative to others overlapping the same pixels. It is calculated as:

$$I_d = \sum_{p \in P} \alpha_i \prod_{j=1}^{i-1} (1 - \alpha_j) \quad (4.2)$$

where P denotes the set of pixels covered by the Gaussian, and i is the rank of the Gaussian in the ordered list of those contributing to pixel p .

The second term, I_i , is a view-independent measure derived from the Gaussian’s scale:

$$I_i = (V_{\text{norm}})^\beta \quad (4.3)$$

Here, V_{norm} represents the normalized volume of the Gaussian (relative to the 90th percentile of all Gaussian volumes), clipped to the $[0, 1]$ range, and β is a tunable hyperparameter that controls the sensitivity to volume size.

Finally, the least important $\tau\%$ of Gaussians — based on their combined score I_g — are pruned from the scene. The pruning threshold τ is a scene-specific hyperparameter that can be adjusted to balance compression and visual fidelity.

This pruning step plays a crucial role in reducing the computational and memory costs of the representation, while preserving the perceptual quality of the rendered outputs.

4.4 Geometry Compression

4.4.1 Voxelization

After pruning, the 3D positions of the Gaussians are voxelized using an Octree structure, in which the 3D space is recursively subdivided into eight subvoxels up to a predefined depth d . This voxelization step is crucial for enabling the subsequent transform, as it converts the initial unstructured set of primitives into a structured format where spatial transforms can be effectively applied. Specifically, voxelization maps each 3D position onto a regular grid of size $[0, 2^d)^3$. The transformation applied to each point is:

$$\hat{X}_n = \text{round} \left(\frac{X_n^{\text{orig}} - T}{s} \right) \quad (4.4)$$

where X_n^{orig} denotes the original mean vector of the Gaussian. The parameter T is chosen based on the minimum values of the original coordinates, with the goal of centering the object, while s is a scaling factor determined by the quantization step and thus directly related to the content’s compression level. Indeed, for larger values of s , more points will fall within the interval between two consecutive integers and will therefore be approximated by the round function to the same integer.

However, this process introduces distortion for two main reasons: first, Gaussians are displaced from their original positions to align with voxel centers; second, multiple Gaussians falling within the same voxel may be merged into a single representative. These effects are significantly more pronounced in 3DGS than in traditional Point Clouds due to the sensitivity of the rendering process to the precise positioning of important primitives. Even small positional shifts can result in visible degradation in the rendered output. To mitigate this, the voxelization depth d is carefully selected on a per-scene basis to ensure that most Gaussians are preserved and the distortion introduced is minimized.

4.4.2 Octree coding

Taking inspiration from the GPCC standard, geometry is encoded after voxelization using Octree coding. The Octree encoding model is a lossless compression algorithm based on a recursive subdivision of the 3D space into octants, continuing until a target depth level is reached. Each subregion is further divided into eight child nodes only if it contains at least one point; otherwise, it is treated as a leaf node in the recursion tree.

Each node that contains points is encoded using a single byte, where each of the eight bits represents the occupancy status of one of its children. If a child octant contains at least one point, the corresponding bit is set to 1; otherwise, it is set to 0, indicating that the voxel is empty.

The main advantage of this representation lies in the typically sparse distribution of occupied voxels: they are usually far fewer than empty ones and tend to cluster in specific regions. As a result, the recursion can often be stopped at much shallower depths, significantly reducing the number of bytes that need to be transmitted.

4.5 Attribute Pre-processing

4.5.1 Recoloring

After voxelization, all Gaussians within the same voxel are merged. The attributes of the resulting representative point are computed based on the corresponding attributes of the merged Gaussians. There are different alternatives for computing the voxel's attributes but the most straightforward approach is to averaging out all attributes of the involved Gaussians.

4.5.2 Rotation to Euler Mapping

Following the approach proposed in [4], we replace the rotation quaternion representation ($q \in \mathbb{R}^4$) with its equivalent Euler angle formulation ($\mathbf{e} \in \mathbb{R}^3$). This substitution reduces storage requirements by one floating-point value per Gaussian without any loss in rendering quality, as the conversion between the two representations is fully reversible.

Given a quaternion $q = [w, x, y, z]$, we compute the corresponding Euler angles $\mathbf{e} = [\phi, \theta, \psi]$ using the following expressions:

$$\begin{bmatrix} \text{atan2}(2(wx + yz), 1 - 2(x^2 + y^2)) \\ -\frac{\pi}{2} + 2 \cdot \text{atan2}\left(\sqrt{1 + 2(wy - xz)}, \sqrt{1 - 2(wy - xz)}\right) \\ \text{atan2}(2(wz + xy), 1 - 2(y^2 + z^2)) \end{bmatrix} \quad (4.5)$$

During decoding, the corresponding rotation matrix can be reconstructed directly from the Euler angles using the following expression:

$$\begin{bmatrix} C_\theta C_\psi & -C_\phi S_\psi + S_\phi S_\theta C_\psi & S_\phi S_\psi + C_\phi S_\theta C_\psi \\ C_\theta S_\psi & C_\phi C_\psi + S_\phi S_\theta S_\psi & -S_\phi C_\psi + C_\phi S_\theta S_\psi \\ -S_\theta & S_\phi C_\theta & C_\phi C_\theta \end{bmatrix} \quad (4.6)$$

where $C_\phi = \cos(\phi)$, $S_\phi = \sin(\phi)$, and similarly for θ and ψ .

This transformation not only reduces the amount of data to encode per point, but also retains full reversibility and numerical stability, making it a suitable and efficient choice for our compression pipeline.

4.6 RAHT

After preprocessing, we apply a transformation to sparsify the signal. As mentioned in the Introduction, we adopt the RAHT transform. RAHT is a hierarchical Wavelet Transform coding algorithm designed for Point Cloud attribute compression [2], and it is included in the GPCC standard [12].

A Wavelet Transform (WT) decomposes a signal into a set of basis functions generated by scaling and shifting a single "mother" function $\phi(t)$, known as the wavelet. Specifically, the family of functions $\{\phi_{a,b}(t)\}$ is defined as:

$$\phi_{a,b}(t) = \frac{1}{\sqrt{a}} \phi\left(\frac{t-b}{a}\right) \quad (4.7)$$

If these functions satisfy certain mathematical properties, they can be used to represent signals in a way analogous to the Fourier basis.

It is not necessary to go deeper into the mathematical foundations behind Wavelet Transform since this is not the main purpose of this thesis and we are adopting a well-defined transform without any modification, but it is important to note that Wavelet Transforms are widely used in compression, particularly for images. In many compression systems [41], [42], [43], WTs replace the Discrete Cosine Transform (DCT) typical for example of the JPEG standard [21].

The general approach in wavelet-based compression is similar to subband coding. The signal is passed through filter banks to separate it into subbands. The resulting components are downsampled, quantized, and encoded. On the decoder side, these steps are reversed: the encoded data is decoded, upsampled, and recombined to reconstruct the original signal.

One of the simplest and most widely used WTs is the Haar Wavelet Transform [44], a very

simple transformation that relies on the principle of “averaging and differencing.” At each step, pairs of consecutive coefficients are averaged to capture low-frequency (coarse) information. Because this averaging loses some detail, corresponding detail coefficients are calculated and stored. These detail coefficients allow the decoder to accurately reconstruct the original signal. The average values are then passed to a coarser level where the process repeats until reaching the root of the hierarchical representation. The final transformed signal has the same number of elements as the original but is typically much sparser, since many detail coefficients are close to zero, making it highly efficient for entropy coding.

Although this explanation describes one-dimensional signals, Wavelet Transforms can be extended naturally to higher dimensions. A common approach uses separable transforms, applying one-dimensional filters along each dimension in turn. This technique is, for example, the basis of the JPEG2000 standard [43] for images.

The RAHT is inspired by classical Wavelet Transforms, but specifically tailored for compressing the attributes of 3D Point Clouds. It operates on a hierarchical structure based on an octree decomposition, beginning with the finest level of voxels and recursively merging them into larger subsets until reaching the root of the tree.

The transformation proceeds in layers, with each layer corresponding to a level of the octree. As in traditional 2D wavelet transforms, where processing occurs sequentially across spatial dimensions (e.g., first horizontal, then vertical), RAHT applies three passes per layer, one along each spatial axis. Only occupied voxels are considered during this process: if one voxel in a pair is empty, the occupied voxel is promoted to the next level without modification. Figure 4.4 illustrates the key steps of the RAHT process for a single octree layer. This procedure is repeated iteratively at each level.

At each level l , every voxel (x, y, z) is associated with two values:

- A coefficient $g_{l,x,y,z}$ representing the scaled average color of the voxel.
- A weight $w_{l,x,y,z}$ representing the number of original voxels aggregated to form the current voxel, essentially encoding local point density.

Taking as reference the x axis, the transformation between levels is defined as follows:

$$\begin{bmatrix} g_{l-1,x,y,z} \\ h_{l-1,x,y,z} \end{bmatrix} = \mathbf{T}_{w_1,w_2} \begin{bmatrix} g_{l,2x,y,z} \\ g_{l,2x+1,y,z} \end{bmatrix} \quad (4.8)$$

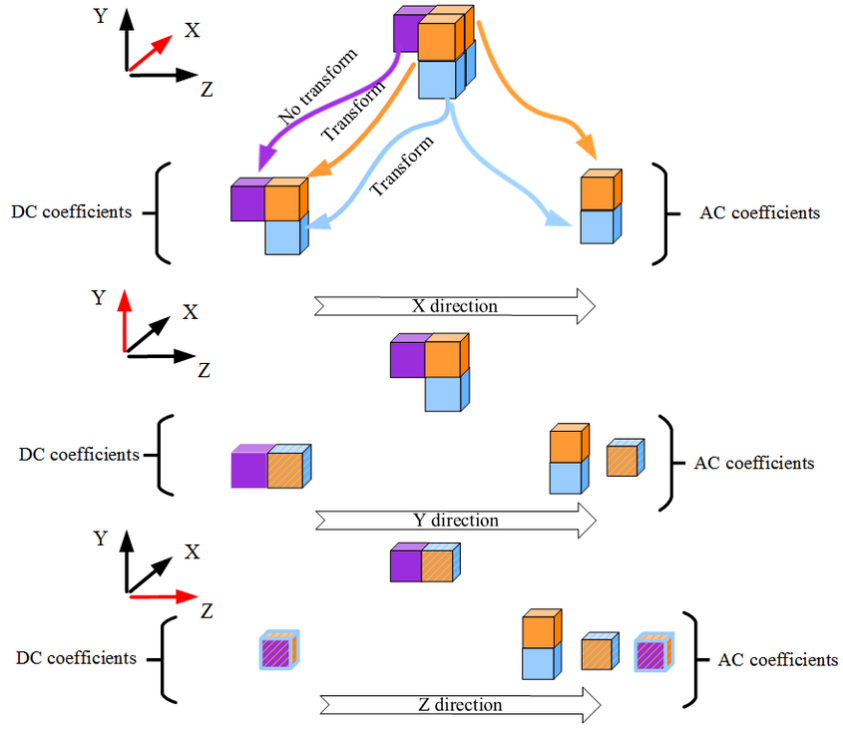


Figure 4.4: RAHT pipeline [2]

Here, $w_1 = w_{l,2x,y,z}$ and $w_2 = w_{l,2x+1,y,z}$ are the weights of the paired voxels, and the transformation matrix is defined as:

$$\mathbf{T}_{w_1,w_2} = \frac{1}{\sqrt{w_1 + w_2}} \begin{bmatrix} \sqrt{w_1} & \sqrt{w_2} \\ -\sqrt{w_2} & \sqrt{w_1} \end{bmatrix} \quad (4.9)$$

The resulting coefficient $g_{l-1,x,y,z}$ is a low-pass output used to compute subsequent levels and does not need to be stored. In contrast, $h_{l-1,x,y,z}$ is a high-pass coefficient capturing detail, which is encoded and transmitted.

Weights are updated recursively at each level:

$$w_{l-1,x,y,z} = w_{l,2x,y,z} + w_{l,2x+1,y,z} \quad (4.10)$$

The name Region Adaptive refers to the fact that the transformation matrix changes dynamically based on the local weights, which reflect the Point Cloud's spatial density. This adaptation improves reconstruction accuracy by moving low-pass coefficients closer to denser regions. Notably, when all weights are equal, the RAHT reduces to a scaled version of the classical Haar transform. For this reason, it is known as the Region Adaptive Hierarchical (or Haar) Transform.

Another useful property of RAHT is orthonormality, as it ensures the quantization error to be preserved across the transform domain, which is beneficial for maintaining signal integrity during compression.

All of these properties make RAHT highly effective for handling the irregular and sparse nature of 3D Point Cloud data. Given the strong structural and conceptual similarities between Point Clouds and 3DGS, where both represent 3D scenes as unstructured sets of discrete elements with associated attributes, RAHT is a promising candidate for compressing Gaussian Splatting data as well.

4.6.1 Weighted RAHT

To enhance the effectiveness of the transform, we explore modifications to the initialization of the weights used in the RAHT procedure. In the standard RAHT, weights are set to reflect the local density of points — typically by counting the number of primitives within a voxel. This design has the effect of positioning low-pass coefficients closer to dense regions, thus improving reconstruction quality where data is more concentrated.

Inspired by this principle, and knowing that in 3DGS not all primitives contribute equally to the rendering outcome, we propose alternative weight initialization strategies that favor regions with more perceptually important parameters or primitives. The goal is to shift low-frequency components toward areas with higher visual relevance, thereby improving the efficiency of the transformation.

To achieve this, we introduce non-uniform weight initializations while keeping the RAHT structure unchanged. Two importance metrics are considered:

- **Gradient-Based Importance:** Following the sensitivity analysis proposed in [36], this method defines the importance of each parameter based on its influence on the rendered output. Intuitively, if a small variation in a parameter significantly alters the rendered image, that parameter should be treated as more important. Formally, the importance of the j -th parameter of the i -th Gaussian, denoted p_{ij} , is computed as:

$$w_{ij} = \frac{1}{\sum_{k=1}^N P_k} \sum_{k=1}^N \left| \frac{\partial E_k}{\partial p_{ij}} \right| \quad (4.11)$$

where N is the number of training images, P_k the number of pixels in image k , and E_k the total image energy (sum of RGB values over all pixels). This formulation averages

the sensitivity across the training set.

- **Volume-Based Importance:** This simpler and computationally cheaper approach assigns a scalar importance value to each Gaussian based on its volume. The rationale is that larger Gaussians influence more pixels in the final render and are therefore more visually relevant. Given the scale vector of the i -th Gaussian, $\mathbf{s} = [s_1, s_2, s_3]$, the weight is defined as:

$$w_i = s_1 \cdot s_2 \cdot s_3. \quad (4.12)$$

In both cases, we evaluate the raw weights as well as a normalized version, where weights for each attribute channel are divided by their global sum to maintain consistency across the dataset.

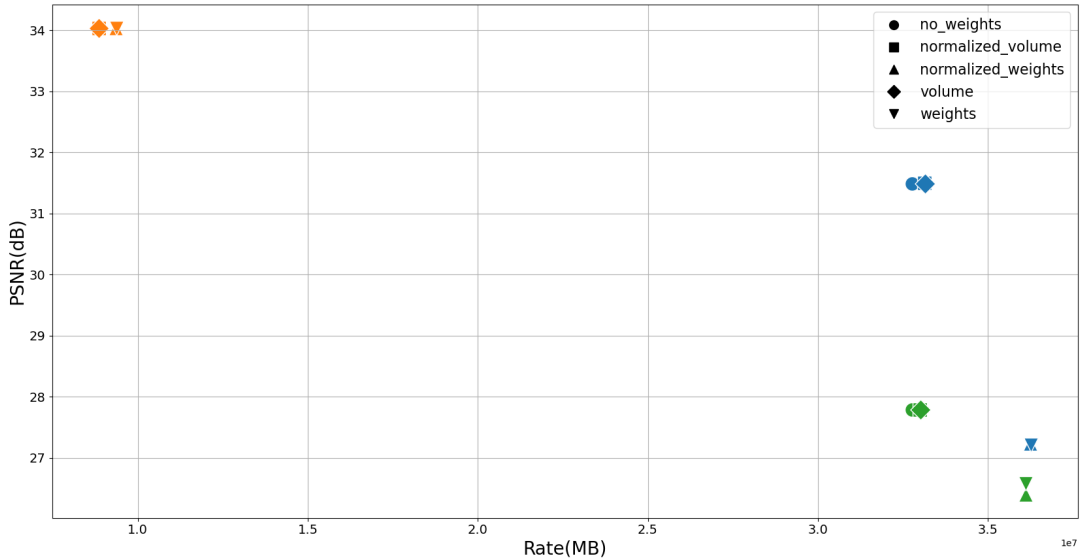


Figure 4.5: RD results for *Mic* (Nerf-Synthetic), *Counter* and *Room* (Mip-NeRF360) for different weights initializations

Figure 4.5 compares these weighting schemes to the default RAHT initialization. All experiments are performed under identical configurations, with weight initialization being the sole difference. For this preliminary study, we make the strong assumption that the decoder is aware of the exact weights — an unrealistic scenario in practice, except for the standard version, whose weights are scene-independent.

Despite this idealized condition, results indicate that the standard RAHT initialization consistently achieves better RD performance. A plausible explanation is that the alternative weightings produce higher-magnitude high-frequency coefficients, which are more difficult

to quantize and compress effectively. This outcome supports the usage of the standard initialization in practical settings.

4.7 Bit Allocation

To enable effective quantization and improve compression efficiency, a bit allocation scheme must account for the varying energy and rendering importance of each attribute. In this work, we adopt a scene-independent, channel-wise allocation strategy, which simplifies the encoder and avoids the overhead of transmitting large amounts of side information.

To formalize the problem, let b_i^* denote the optimal number of quantization bits for the i -th attribute. Our goal is to approximate it with a scene-independent value that performs well across different scenes. To this end, we conduct an extensive analysis of how the quantization of each attribute impacts final rendering quality. For every attribute, we plot the rendering distortion as a function of its per-channel bitrate:

$$D_{\text{render}} = f(R_i), \quad i \in [1, n_{\text{attr}}]. \quad (4.13)$$

Specifically, for each experiment we first apply the RAHT, and then code attribute i using a number of bits $b \in \{2, 4, 6, 8, 16, 32\}$. All the other attributes $j \neq i$ are instead coded at a fixed length of 8 bits. An example of the obtained RD curves is shown in Figure 4.6 for two attributes, rotation and second-order SH coefficients.

The observed RD curves reveal that different attributes exhibit distinct behaviors: they saturate at different bitrates — beyond which additional precision yields negligible improvements in rendering quality — and show varying slopes before saturation, reflecting their sensitivity to coarse quantization. For instance, in Figure 4.6, rotation (left) requires a higher bitrate to preserve quality, saturating around 8 bpp, while the second-order SH (right) exhibits a much flatter RD curve, with saturation near 4 bpp, indicating that it tolerates coarser quantization. This behavior can be attributed to several factors. On one hand, it depends on the intrinsic importance of the attribute for the rendering process: as partially discussed in Section 4.1.3, some attributes have limited impact on the final image quality and so they can be represented more coarsely. On the other hand, the quantization efficiency also depends on the numerical range of each attribute. Attributes defined over wider ranges are inherently more challenging to represent accurately with a fixed number of bits, as greater dynamic range leads to larger quantization steps for a given bit budget.

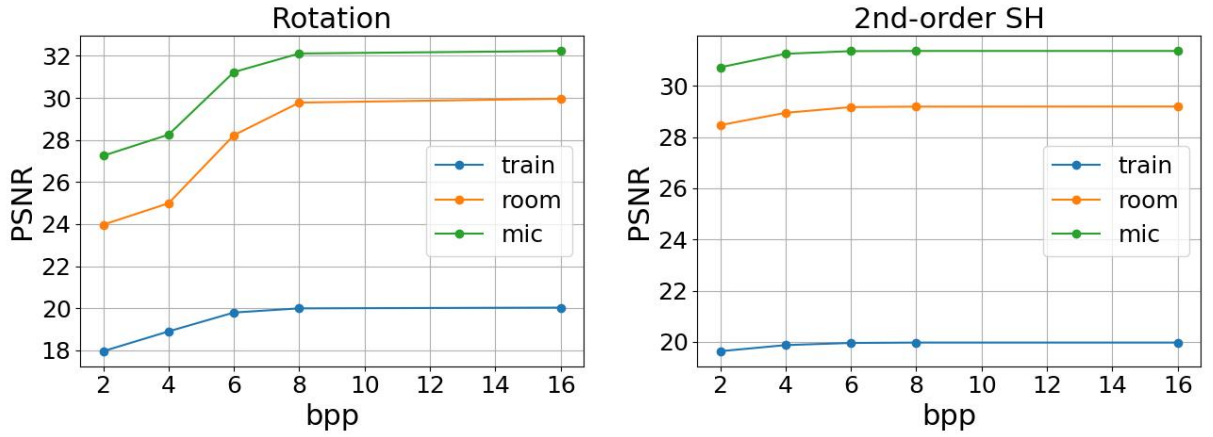


Figure 4.6: RD Curves for Rotation and 2nd order SH coefficients.

These observations motivate the need for attribute-specific quantization, which we address by guiding our bit allocation scheme based on saturation points. This behavior is consistent across scenes, supporting the generalization of bit allocation decisions rather than requiring scene-specific tuning. Similar experiments were performed for all attributes and across a broader set of scenes, leading to the design of the bit allocation scheme summarized in Table 4.4.

Table 4.4: Proposed channel-wise bit allocation.

Parameter (attribute channel)	b_i^*
α	8
$e_i, i \in [1, 3]$	8
$C_i, 0^{\text{th}}$ order	8
$s_i, i \in [1, 3]$	10
$C_i, 1^{\text{st}}$ order	4
$C_i, 2^{\text{nd}}$ order	4
$C_i, 3^{\text{rd}}$ order	2

4.8 Block-wise Scalar Quantization

After the transformation, we apply block-wise scalar quantization [45],[4], which offers a more refined approximation compared to uniform channel-wise quantization, thereby mitigating the associated quality loss. Specifically, each attribute channel is first partitioned into multiple blocks based on Morton ordering, which ensures spatially coherent grouping of Gaussians.

Morton ordering, also known as Z-order curve, is a space-filling curve that maps multidimensional data (3D coordinates in our case) into a one-dimensional sequence while preserving

spatial locality. This is achieved by interleaving the binary representations of the x , y , and z coordinates of each point. For example, given 3-bit coordinates $x = x_2x_1x_0$, $y = y_2y_1y_0$, and $z = z_2z_1z_0$, the Morton code is constructed as $z_2y_2x_2z_1y_1x_1z_0y_0x_0$. This ordering groups nearby spatial points into consecutive positions in the sequence, which is particularly useful for block-wise processing, as it increases the likelihood that points within a block are spatially close and share similar characteristics. As a result, quantization becomes more effective due to increased local correlation.

Each block is then quantized independently using uniform scalar quantizers. All blocks within the same channel share the same quantizer design. However, different channels may employ different bit depths, as determined by the bit allocation strategy discussed earlier. Since the dynamic range can vary for each block, channel and scene, this information must be transmitted to the decoder to enable correct signal reconstruction.

Finally, the number of blocks is manually selected for each scene to balance the trade-off between compression quality and the overhead of metadata required at the decoder. A higher number of blocks allows for a more accurate representation, but it also increases the amount of side information that must be transmitted. Notably, DC coefficients in all channels are left unquantized, since quantizing them causes quality degradation that is not justified by the compression gain.

4.9 Fine-tuning

After voxelization, and before applying transform coding and generating the final bitstream, we introduce a fine-tuning stage. During this process, densification is disabled and only appearance attributes are optimized, while Gaussian positions remain fixed. This step allows the system to adapt to the changes introduced by voxelization, helping to preserve rendering quality. Additionally, it serves multiple purposes, as described below.

4.9.1 Quantization-aware Fine-tuning

To enhance compression performance, fine-tuning is performed with quantization included in the optimization loop. This enables the model to account for quantization effects and adjust Gaussian parameters accordingly. Since quantization is a non-differentiable operation, it normally blocks gradient flow during backpropagation. To address this, we use the Straight-Through Estimator (STE) [46], a common technique to approximate gradients through non-

differentiable functions.

The STE works by applying the true quantization function in the forward pass (e.g., rounding to the nearest quantization level), ensuring that the model is trained on quantized values. However, during the backward pass, the gradient of the quantization function is approximated by the identity function. In practice, this means that the gradient of the quantized output with respect to its input is set to 1, allowing gradients to propagate unchanged through the quantization layer. Despite being a rough approximation, this method has been shown to be effective in training quantization-aware models.

It is worth noting that, since quantization is performed in the transform domain, a transformation and quantization step must be added at each forward pass, thereby increasing the complexity of the training process.

4.9.2 Sparsity-promoting Fine-tuning

The distortion loss \mathcal{L}_D in the original 3DGS pipeline is defined as:

$$\mathcal{L}_D = (1 - \lambda_{SSIM})\mathcal{L}_1 + \lambda_{SSIM}\mathcal{L}_{SSIM}. \quad (4.14)$$

This loss function combines two metrics to guide the rendering quality during training:

- \mathcal{L}_1 : the mean absolute error (ℓ_1 loss) between the predicted image and the ground-truth image, computed pixel-wise. It encourages pixel-level accuracy and penalizes large deviations, promoting faithful color reproduction.
- \mathcal{L}_{SSIM} : the SSIM loss, which captures perceptual differences by comparing structural information. It correlates better with human visual perception and helps preserve image structure and sharpness.

However, optimizing for visual quality alone often results in models where spatially close Gaussians exhibit highly diverse attributes, limiting the ability to exploit spatial redundancy in compression. To promote a more compressible representation, we introduce an additional rate proxy term in the form of a sparsity-promoting loss that penalizes large-magnitude transform coefficients, concentrating energy into fewer coefficients and thereby improving coding efficiency [47]. This regularization strengthens the local structure of the model, which, after transformation, significantly reduces the data entropy.

The sparsity loss \mathcal{L}_S is given by:

$$\mathcal{L}_S = \frac{1}{n_{ch}} \sum_{i=1}^{n_{ch}} \sum_{j=1}^{n_{ac}} |x_{ij}| \quad (4.15)$$

where n_{ch} is the number of attributes per Gaussian, n_{ac} is the number of AC coefficients after applying RAHT, and x_{ij} denotes the j -th transform coefficient of the i -th channel.

This formulation corresponds to the average ℓ_1 norm per channel, which encourages sparsity by pushing many coefficients toward zero [48], [49]. This strategy is inspired by the well-known LASSO regularization [50], widely used in statistical learning and signal processing to obtain sparse solutions. Similarly to LASSO, our objective promotes compact and efficient representations by selectively suppressing less relevant transform components.

In addition to the standard sparsity-promoting loss based on the ℓ_1 norm, we also experimented with a variant inspired by the *Fused* LASSO [51]. This approach not only encourages sparsity in the transform coefficients but also promotes smoothness by penalizing differences between adjacent coefficients. The corresponding loss function is defined as:

$$\mathcal{L}_{\text{Fused}} = \frac{1}{n_{ch}} \sum_{i=1}^{n_{ch}} \left(\sum_{j=1}^{n_{ac}} |x_{ij}| + \lambda \sum_{j=2}^{n_{ac}} |x_{ij} - x_{i(j-1)}| \right) \quad (4.16)$$

where λ is a tunable weight that controls the balance between sparsity and smoothness, x_{ij} is the j -th RAHT transform coefficient of the i -th attribute channel, and n_{ch} , n_{ac} are defined as before.

However, in our experiments, we observed that the *Fused* LASSO provided similar compression for the tested scenes but it also introduced a higher optimization overhead and was more sensitive to the choice of λ , requiring careful tuning, as shown in Figure 4.7 and for this reason it has been discarded.

The total loss function is defined as a linear combination of \mathcal{L}_D and \mathcal{L}_S , with their relative importance controlled by the hyperparameter λ_S :

$$\mathcal{L} = (1 - \lambda_S)\mathcal{L}_D + \lambda_S\mathcal{L}_S. \quad (4.17)$$

A key advantage of this formulation is that it allows for a straightforward and effective control of the RD trade-off via λ_S tuning, thereby making the system much more flexible compared

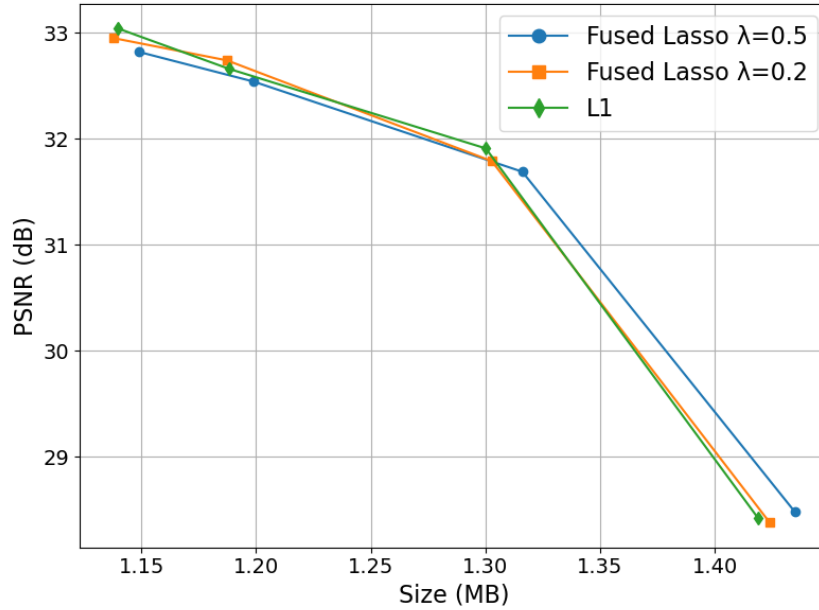


Figure 4.7: RD curves for *Mic* (Nerf-Synthetic) for different numbers of fine-tuning iterations

to most of existing approaches in the literature. Importantly, the computational overhead of this additional loss is negligible, as RAHT is already part of the training loop for quantization-aware fine-tuning.

4.10 Entropy Coding

To effectively exploit the sparsity of the transformed coefficients and to generate the final bitstream, we include an entropy coding block at the end of the pipeline. This final step is crucial because, after quantization and transformation (e.g., via RAHT), the resulting coefficients typically exhibit statistical redundancy. Entropy coding is designed to capture and exploit such redundancy by assigning shorter codewords to more frequent symbols and longer ones to rarer ones, thereby reducing the overall size of the bitstream.

4.10.1 Deflate

The final bitstream consists of all metadata required by the decoder, the Octree bitstream, and the quantized AC RAHT coefficients along with the unquantized DC coefficients. All components are further compressed using the Deflate algorithm [52]. This algorithm is a combination of the dictionary-based *LZ77* technique [19] and Huffman coding [18], an entropy encoding that exploits non-uniformity at the byte level to achieve a shorter representation at the bits level.

Although Deflate is not specifically tailored to the statistical properties of our transformed coefficients, it provides a simple and efficient method to further reduce the bitstream size with negligible computational overhead.

4.10.2 Set Partitioning in Hierarchical Trees

A more sophisticated alternative to Deflate has been explored, based on the Set Partitioning in Hierarchical Trees (SPIHT) algorithm originally developed for image compression.

SPIHT is a wavelet-based compression algorithm introduced in [53], known for its high compression efficiency, low computational complexity, and ability to produce an embedded bitstream. This embedded structure enables efficient decoding at multiple bitrates or quality levels, making the algorithm particularly suitable for progressive transmission and scalable applications.

The core idea behind SPIHT is the use of hierarchical tree structures to organize wavelet-transformed data. After applying a wavelet transform to the data, the resulting coefficients are organized in a tree-like structure, where the most significant information — i.e., the coefficients that contribute most to image quality — are at the top of the hierarchy and are encoded first.

Inspired by the earlier Embedded Zerotree Wavelet (EZW) algorithm, SPIHT improves both the compression performance and the RD trade-off by progressively encoding significant coefficients and refining them across multiple passes. The key steps of the algorithm are the following:

1. **Threshold initialization:** A threshold is initialized based on the largest wavelet coefficient in the image. This threshold determines which coefficients are considered significant at the current pass.
2. **Set partitioning:** The wavelet coefficients are organized into spatial orientation trees, which reflect the hierarchical structure of the image after wavelet transformation.
3. **Progressive significance testing:** At each step, the algorithm tests which coefficients or coefficient sets exceed the current threshold. Significant ones are further partitioned into smaller subsets, eventually down to single coefficients. Their significance information is encoded into the bitstream.
4. **Refinement pass:** After the significant coefficients have been identified, the algorithm refines their values bit by bit, increasing precision progressively in subsequent passes.

This ensures that the bitstream contains more accurate representations of important coefficients as more bits are transmitted.

5. **Threshold update and iteration:** The threshold is halved in each iteration, and the process is repeated until the desired bit budget or quality level is reached.

This hierarchical set partitioning allows SPIHT to focus on the most visually important parts of the image first, providing a scalable and embedded bitstream that supports progressive transmission and rate-distortion optimization.

4.10.2.1 SPIHT for 3DGS Compression

Since RAHT is a wavelet-like transform, it should be possible to adapt the SPIHT algorithm to encode the transformed coefficients in our context as well. Specifically, by constructing an appropriate hierarchical structure over the coefficients – analogous to the spatial orientation trees used in image compression – we could apply SPIHT-style significance testing and progressive refinement. This would enable efficient embedded coding tailored to the statistics and sparsity of the RAHT domain.

In [54], an adaptation of the SPIHT algorithm for RAHT-based Point Cloud compression is proposed. Although the method achieves slightly lower compression efficiency compared to arithmetic coding, it offers the significant advantage of producing an embedded bitstream, which enables greater flexibility and scalability. Motivated by this approach, we explore the application of a similar SPIHT-inspired strategy for 3DGS compression.

Our implementation adopts a pipeline similar to the one proposed in [54] for Point Cloud compression. To assess its effectiveness, we conducted preliminary experiments before integrating it into the full 3DGS compression framework. In this setup, 13 attribute channels are considered, including only 3 AC spherical harmonics coefficients. Due to the wide variability in dynamic ranges across channels, the quantization step size is defined separately for each one. Specifically, for the i -th channel, the quantization step is given by:

$$\Delta_i = \frac{\max(|C[1 :, i]|)}{2^b} \quad (4.18)$$

where $C[1 :, i]$ denotes the set of transformed coefficients for channel i , excluding the DC component, and b is the number of bitplanes.

We tested our approach on two representative scenes: *Train* (from Tanks and Temples) and *Room* (from Mip-NeRF360). The bitrate is computed as the ratio between the bitstream

length and the number of encoded coefficients, excluding the DC coefficients and metadata. PSNR is measured using the MSE between original and reconstructed attributes, rather than on rendered images, as the algorithm has not yet been integrated into the full rendering pipeline.

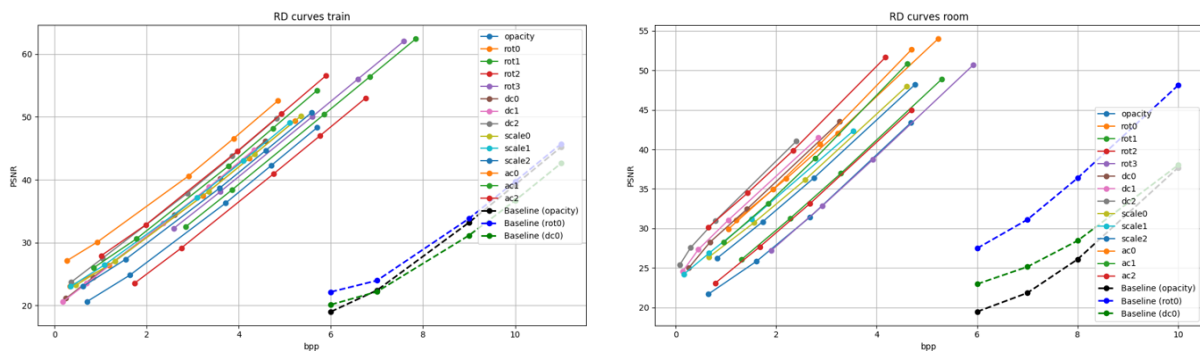


Figure 4.8: SPIHT results for *Train* (left) and *Room* (right).

Figure 4.8 presents the results for the two reference scenes. As shown, the application of the SPIHT algorithm yields some improvements over the baseline (standard quantization without entropy coding). However, these gains come with significant integration challenges. In particular, the algorithm proves too inefficient to be incorporated into the full compression framework. Although the current implementation could be optimized further, the inherent requirement to process each subtree independently makes parallelization difficult and limits scalability. We therefore conclude that the improvement in RD performance is insufficient to justify the added complexity and development effort required to adapt this algorithm.

Chapter 5

Experimental Results

5.1 Experimental Setting

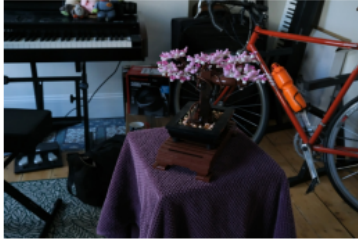
5.1.1 Datasets

To thoroughly evaluate the performance of our proposed system, we conduct experiments on three widely adopted benchmarks for Novel View Synthesis: Mip-NeRF360 [55], Tanks and Temples [56], and NeRF-Synthetic [57]. These datasets offer a broad variety of scenes and objects, encompassing different levels of geometric complexity, lighting conditions, and structural details. This diversity allows us to assess the robustness and generalizability of our approach across different real-world and synthetic scenarios and to ensure fair comparison with existing compression methods. In the following a brief description of the three datasets is provided.

Mip-NeRF360

Mip-NeRF360 is a dataset intended to support the development and evaluation of Novel View Synthesis techniques in complex, large-scale environments. It includes a set of nine high-resolution real scenes, five outdoor and four indoor, and each of them is characterized by a complex subject in the center, surrounded by detailed, photorealistic backgrounds. Each scene is captured from multiple viewpoints with accurate camera poses, making it well-suited for testing reconstruction quality in unbounded and realistic 3D settings. The following scenes are included: “bicycle”, “bonsai”, “counter”, “garden”, “kitchen”, “room”, “stump” and Figure 5.1 shows a sample image from each of them.

Mip-NeRF 360



bonsai



bicycle



counter



garden



kitchen



room



stump

Figure 5.1: Mip-NeRF360 Dataset

Tanks and Temples

Tanks and Temples is a benchmark dataset introduced in 2017 for evaluating image-based 3D scene reconstruction methods in real-world conditions. It includes a variety of high-resolution indoor and outdoor scenes, captured under uncontrolled lighting and environmental conditions. Ground-truth geometry is provided using industrial-grade laser scanning equipment, ensuring high accuracy for quantitative evaluation. Following standard evaluation protocols [26], [27], we focus on two large-scale unbounded outdoor scenes – “train” and “truck” – which are commonly used for testing 3DGS compression methods. Sample images from these scenes are shown in Figure 5.2.

NeRF-Synthetic

NeRF-Synthetic is a controlled benchmark dataset introduced alongside the original NeRF paper [5], specifically designed to evaluate Novel View Synthesis methods under ideal, noise-

Tanks and Temples



Figure 5.2: Tanks and Temples Dataset

free conditions. It features fully synthetic scenes where both geometry and appearance are perfectly modeled, ensuring reproducibility and eliminating the variability associated with real-world data. Camera poses are uniformly distributed across the viewing hemisphere, and each image is paired with precise pose information and ground-truth color and geometry. For this study, we include all eight object-centric scenes: "chair", "drums", "ficus", "hotdog", "lego", "material", "mic", and "ship". Example images from these scenes are shown in Figure 5.3.

Synthetic NeRF



Figure 5.3: NeRF-Synthetic Dataset

5.1.2 Evaluation Metrics

As emphasized in the Introduction, the primary objective of 3DGS compression techniques is to achieve a memory-efficient representation of 3D scenes while preserving high rendering quality. Therefore, the evaluation of any proposed method must account for both the distortion

in the rendered output and the corresponding compression rate. In the following, we outline the metrics used to assess both distortion and rate performance.

5.1.2.1 Distortion Metrics

While subjective quality assessment remains the most appropriate approach for capturing variations in perceived content quality and ultimately measuring the user’s Quality of Experience, it requires substantial effort. This includes conducting user studies and performing statistical analysis to ensure the reliability and validity of the results. Therefore, in practice, we rely on objective metrics that correlate well with human perception to assess the distortion introduced by compression. In this work, we focus exclusively on full-reference metrics—i.e., those requiring ground truth images for comparison. Specifically, we consider Peak Signal to Noise Ratio (PSNR), SSIM, and Learned Perceptual Image Patch Similarity (LPIPS).

PSNR

Peak Signal-to-Noise Ratio quantifies the ratio between the maximum possible signal power of an image and the power of the noise that corrupts its representation. It is computed by comparing a distorted image against a reference “clean” image. Due to the wide dynamic range of image signals (i.e., the ratio between the largest and smallest possible values), PSNR is typically expressed in a logarithmic decibel (dB) scale.

The metric is formally defined as:

$$PSNR(I) = 10 \cdot \log_{10} \left(\frac{MAX(I)^2}{MSE} \right) \quad (5.1)$$

where $MAX(I)$ is the maximum possible pixel value of the image (255 for 8-bit images), and MSE is the Mean Squared Error, calculated as:

$$MSE(I, \hat{I}) = \frac{1}{mn} \sum_{i=0}^{n-1} \sum_{j=0}^{m-1} \|\hat{I}(i, j) - I(i, j)\|^2 \quad (5.2)$$

Here, \hat{I} denotes the original (ground-truth) image, I the reconstructed image being evaluated, and m and n the image dimensions. A higher PSNR value indicates better image quality, meaning the reconstructed image is more similar to the original reference.

In essence, PSNR provides a straightforward pixel-wise comparison between images. While its computational simplicity makes it a widely used distortion metric, it is important to note that PSNR does not correlate strongly with human visual perception. It focuses solely on nu-

merical differences and does not account for characteristics of the human visual system, which limits its ability to reflect perceived image quality accurately.

SSIM

The Structural Similarity Index Measure [17] is a widely used full-reference metric for evaluating image degradation. It assesses the visual impact of differences between a reference image and a distorted one by focusing on structural information rather than pixel-wise differences. Its mathematical formulation is:

$$SSIM(x, y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)} \quad (5.3)$$

In this equation, x denotes the reference image and y the image being evaluated. The SSIM compares them using statistical features: the means μ_x and μ_y , the variances σ_x^2 and σ_y^2 , and their covariance σ_{xy} . The constants C_1 and C_2 are included to stabilize the calculation in the presence of small denominators.

Unlike traditional pixel-based metrics like PSNR, SSIM is designed to align more closely with human visual perception by considering three key components: luminance, contrast, and structural similarity. This allows SSIM to better reflect perceptual quality, although it comes with higher computational complexity.

A higher SSIM value indicates greater structural similarity between the reference and generated images, and thus better perceptual quality.

LPIPS

The Learned Perceptual Image Patch Similarity metric [58] is a perceptual similarity measure that compares images based on learned visual features rather than low-level pixel differences. Unlike traditional metrics such as PSNR and SSIM, which operate in the pixel domain, LPIPS evaluates image similarity at the feature level, leveraging deep neural networks (e.g., VGG) pretrained on large-scale image classification tasks.

To compute the LPIPS score, both the reference and the test image are passed through the same pretrained network. The resulting feature activations at multiple layers are compared, and the differences are aggregated spatially. The mathematical formulation is:

$$LPIPS(x, y) = \sum_{l=1}^L \frac{1}{H_l W_l} \sum_{h=1}^{H_l} \sum_{w=1}^{W_l} |w_l (x_{lhw} - y_{lhw})|_2^2 \quad (5.4)$$

Here, x_{lhw} and y_{lhw} are the deep feature representations of the reference and test images at spatial location (h, w) in layer l , and w_l denotes learned weights for each channel. H_l and W_l are the height and width of the feature map at that layer.

LPIPS is considered one of the most perceptually aligned image quality metrics available. It captures high-level structural and semantic differences that are often missed by pixel-wise metrics, making it especially effective for evaluating tasks like image synthesis or neural rendering. However, this comes at the cost of increased computational complexity and a strong dependence on the pretrained backbone network.

A lower LPIPS score indicates higher perceptual similarity between the two images.

5.1.2.2 Rate Metric

The rate in our evaluation is expressed in terms of file size, measured in Megabytes (MB). This choice departs from the convention commonly adopted in other multimedia compression domains – such as Point Cloud compression – where bits per point (bpp) is typically used. Bits per point offers a more meaningful and scalable metric because it normalizes compression efficiency relative to the number of primitives. This normalization allows for fair comparisons between scenes of vastly different sizes, as it reflects the average storage cost per point. In contrast, raw file size alone can obscure this relationship, making it harder to assess how efficiently the data is compressed when the scene complexity varies.

However, in order to ensure consistency and comparability with existing 3DGS compression methods – whose results are conventionally reported in megabytes – we adopt file size in MB as our primary rate metric throughout this work.

5.1.3 Implementation Details

All trainings, optimizations, and encodings are performed using a GPU *NVIDIA GeForce RTX 3090* with 24 GB of RAM and CUDA Version 12.4. We first train the model based solely on 3DGS [3] using the standard configuration provided. The output after $30k$ steps is then fed into our system.

System parameters are set to standard values commonly used in the literature; in particular, the pruning percentage, octree depth, and number of quantization blocks are adopted from [4]. Table 5.1 contains a summary description of all parameters used during our experiments for the considered scenes.

For Octree coding, we use the reference GPCC software [12], available online at [https :](https://)

Table 5.1: Configuration values per scene: number of blocks (n_block), octree depth, and pruning ratio.

Scene	n_block	Depth	Prune
<i>bicycle</i>	57	20	0.40
<i>bonsai</i>	57	19	0.34
<i>counter</i>	57	19	0.22
<i>garden</i>	57	20	0.28
<i>kitchen</i>	57	19	0.24
<i>room</i>	57	19	0.32
<i>stump</i>	57	20	0.30
<i>train</i>	57	20	0.22
<i>truck</i>	57	20	0.40
<i>chair</i>	52	14	0.16
<i>drums</i>	57	14	0.28
<i>figus</i>	57	14	0.38
<i>hotdog</i>	57	14	0.42
<i>lego</i>	57	14	0.20
<i>materials</i>	57	14	0.20
<i>mic</i>	48	14	0.40
<i>ship</i>	57	14	0.28

[//github.com/MPEGGroup/mpeg-pcc-tmc13](https://github.com/MPEGGroup/mpeg-pcc-tmc13), instead of implementing it from scratch.

Results are reported after $2k$ and $5k$ fine-tuning iterations, with λ_{SSIM} fixed at 0.2 and λ_S set to $5 \cdot 10^{-7}$.

5.2 Compression Results

We evaluate the performance of our proposed method by comparing it against several existing approaches. Specifically, we consider the original 3DGS baseline [3], as well as a set of state-of-the-art post-training compression methods identified in the recent survey [27], including [29], [59], [33], and [30]. Additionally, we include MesonGS [4], which is the most relevant prior work that employs RAHT-based transform coding for attribute compression. Although HierarchicalGS [11] also leverages RAHT, we do not include it in the comparison due to the lack of publicly available results for all scenes and datasets.

For completeness, and to ensure a broad and fair comparison, we also report results for representative retraining-based methods, namely [6] and [32]. To ensure consistency and fairness in the evaluation, we follow the same experimental protocol as the original 3DGS paper [3], including identical train/test splits. All results for the compared methods are quoted directly

from the respective publications, with the exception of 3DGS on the NeRF-Synthetic dataset. In that case, we recomputed the results using the original 3DGS code, as only PSNR was reported in the original paper for that dataset.

Table 5.2: Compression performance results on 3 datasets. Model size is in MB. Best, second best, and third best values for each column are denoted in red, yellow, and green, respectively.

	Mip-NeRF360				Tanks & Temples				NeRF-Synthetic			
	PSNR	SSIM	LPIPS	Size	PSNR	SSIM	LPIPS	Size	PSNR	SSIM	LPIPS	Size
3DGS [3]	27.21	0.815	0.214	734	23.14	0.841	0.183	411	31.07	0.959	0.051	61.39
MesonGS+FT [4]	28.61	0.856	0.206	27.62	23.32	0.837	0.193	16.99	32.92	0.968	0.033	3.66
LightGaussian [29]	27.28	0.805	0.243	42.0	23.11	0.817	0.231	22.0	32.73	0.965	0.037	7.8
CompGS-16K [33]	27.03	0.804	0.243	18.0	23.39	0.836	0.200	12.0	-	-	-	-
Trimming the Fat [59]	27.13	0.798	0.248	20.1	23.69	0.831	0.210	8.6	-	-	-	-
RDO-Gaussian [30]	27.05	0.802	0.239	23.5	23.34	0.835	0.195	12.0	32.97	0.966	0.035	1.84
HAC-lowrate [6]	27.53	0.807	0.238	15.3	24.04	0.846	0.187	8.1	33.24	0.967	0.037	1.18
HAC-highrate [6]	27.77	0.811	0.230	21.9	24.40	0.853	0.177	11.2	33.71	0.968	0.034	1.86
ContextGS-lowrate [32]	27.62	0.808	0.237	12.7	24.20	0.852	0.184	7.1	-	-	-	-
ContextGS-highrate [32]	27.75	0.811	0.231	18.4	24.29	0.855	0.176	11.8	-	-	-	-
Ours 5K	28.44	0.876	0.213	15.61	23.35	0.841	0.197	9.91	32.60	0.965	0.039	1.92
Ours 2K	28.54	0.867	0.206	18.04	23.41	0.841	0.198	10.71	32.53	0.955	0.039	2.20

The quantitative results in Table 5.2 clearly demonstrate the effectiveness of our proposed method. Our system outperforms all considered post-training approaches [33], [29], [4], [59], [30], achieving compression gains of $47\times$, $41\times$, and $32\times$ over the baseline on Mip-NeRF360, Tanks and Temples, and NeRF Synthetic datasets, respectively, while maintaining or even improving the visual quality in some cases. Moreover, our method remains competitive with the retraining approaches [6], [32] on real-world scenes, surpassing their high-rate variants on Mip-NeRF360 for both the rate and the distortion. On the Tanks and Temples dataset, our PSNR is slightly lower — similar to other post-training methods — due to baseline artifacts that are challenging to mitigate. Conversely, retraining methods show slightly better performance on synthetic objects. However, it is important to note that our 3DGS baseline models achieve lower performance compared to those reported in the original paper [3], with PSNR scores approximately 2.25 dB lower on average. This discrepancy negatively affects the overall results of our method, as it is built upon a weaker baseline.

For completeness, Table 5.3 provides a detailed, scene-by-scene comparison between our method and MesonGS [4] on the Mip-NeRF360 dataset. As shown, our approach consistently achieves better rate performance while maintaining comparable visual quality across all scenes, with the exception of *Kitchen*, where our method yields a PSNR that is 1.46 dB lower.

Table 5.3: Scene-by-scene comparison with MesonGS [4] on the Mip-NeRF360 Dataset

	MesonGS + FT [4]		Ours 5k	
	PSNR	Size	PSNR	Size
<i>Bicycle</i>	24.70	46.70	<u>24.92</u>	<u>17.99</u>
<i>Bonsai</i>	<u>31.65</u>	12.69	31.38	<u>7.52</u>
<i>Counter</i>	<u>28.70</u>	13.86	28.14	<u>8.40</u>
<i>Garden</i>	26.59	46.55	<u>26.77</u>	<u>24.12</u>
<i>Kitchen</i>	<u>31.12</u>	18.86	30.16	<u>11.78</u>
<i>Room</i>	<u>31.62</u>	14.71	31.06	<u>7.63</u>
<i>Stump</i>	25.91	39.97	<u>26.62</u>	<u>31.75</u>

5.3 Ablation studies

We conduct a series of ablation studies to assess the impact of each individual component within the proposed compression framework. The analysis is divided into two parts. First, we examine the influence of non-central modules – namely Gaussian pruning, voxelization, octree coding, and the number of quantization blocks. In these experiments, to isolate the effect of each component, both our fine-tuning process and the bit allocation strategy are disabled, and the quantization is uniformly set to 8 bits across all attributes.

Subsequently, we evaluate the two core contributions of this work: the proposed fine-tuning strategy and the bit allocation scheme. In this second stage, all auxiliary components are enabled to simulate realistic system behavior. For all ablation experiments, we use three representative scenes: *Room* from the Mip-NeRF360 dataset, *Truck* from the Tanks and Temples dataset, and *Mic* from the NeRF Synthetic dataset. When fine-tuning is applied, the number of optimization iterations is fixed at $2k$.

5.3.1 Gaussian Pruning

Table 5.4 reports the results obtained with and without applying the pruning strategy. By removing redundant Gaussians, a significant reduction in bitrate is achieved—specifically, the rate decreases by a factor of $1.32\times$, $1.43\times$, and $1.51\times$ for the *Room*, *Truck*, and *Mic* scenes, respectively. These results highlight two key points: first, the effectiveness of the pruning strategy adopted in this work; second, a notable shortcoming of the standard 3DGS pipeline, which tends to generate redundant Gaussians and overfit to the training views. Interestingly, the pruning not only reduces the bitrate but also slightly improves rendering quality in all three test scenes, further confirming that many of the removed Gaussians do not contribute meaningfully to the final output.

Table 5.4: Performance with and without Gaussian Pruning

	Room		Truck		Mic	
	PSNR	Size	PSNR	Size	PSNR	Size
With	<u>30.3</u>	<u>30.16</u>	<u>22.05</u>	<u>44.87</u>	<u>32.77</u>	<u>8.30</u>
Without	29.77	40.01	21.04	64.58	32.13	12.57

5.3.2 Voxelization

In this ablation study, we assess the impact of voxelization depth on compression performance, supporting our design choice of selecting the depth d to retain nearly all points. Specifically, we begin with the depths used in our framework for the three reference scenes – 14 for *Mic*, 19 for *Room*, and 20 for *Truck* – and progressively decrease them in steps of 2, down to a reduction of 8 levels. Figure 5.4 presents the resulting RD curves.

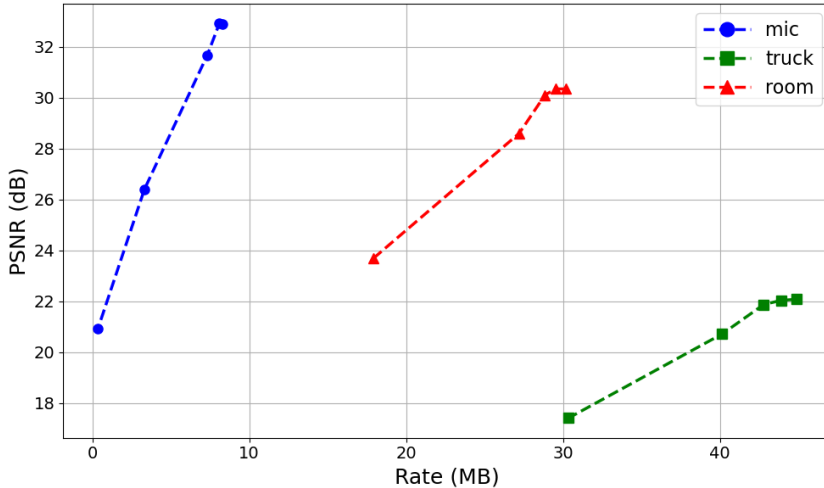


Figure 5.4: RD curves for different voxelization depths

As shown, reducing the voxelization depth significantly degrades rendering quality, especially for certain scenes. This sharp decline derives from a fundamental difference between Point Clouds and Gaussian Splatting. While Point Cloud compression treats all points as equally important and independent, in 3DGS each primitive contributes differently to the final rendering depending on its opacity and volume. In particular, larger Gaussians influence a wider area of the rendered image and therefore require higher positional accuracy. Coarse voxelization displaces these critical primitives or merges them, resulting in noticeable visual artifacts.

Recent work has proposed an adaptive voxelization approach [60], which dynamically adjusts the resolution based on local geometric complexity. Integrating such a strategy into our framework could be a promising direction for future work, as it may help better preserve visual

quality while maintaining compression efficiency.

5.3.3 Octree Coding

Table 5.5 reports the results obtained with and without Octree coding. As discussed in Chapter 4, Octree is a lossless compression technique, meaning that the quality of the rendered images remains unchanged regardless of its use. However, the rate is consistently lower when Octree coding is applied, demonstrating its effectiveness. Given that voxelization is a necessary step for enabling the transform, and Octree coding introduces no distortion while significantly reducing the bit rate, there is no practical reason to exclude it from the compression pipeline.

Table 5.5: Performance with and without Octree Coding

	Room		Truck		Mic	
	PSNR	Size	PSNR	Size	PSNR	Size
With	30.3	<u>30.16</u>	22.05	<u>44.87</u>	32.77	<u>8.30</u>
Without	30.3	31.84	22.05	47.70	32.77	8.60

5.3.4 Block-wise Scalar Quantization

To understand the impact of the number of blocks considered during quantization on compression performance, we conduct an ablation study where we vary the number of blocks used for quantization. Figure 5.5 shows the results for *Room*: we start from the configuration adopted in our framework and progressively decrease the number of blocks down to 1, testing several intermediate values.

As expected, reducing the number of blocks leads to a smaller file size for two main reasons: fewer metadata need to be transmitted, and larger quantization intervals allow for higher compression. However, this comes at the cost of significantly reduced quality, since using a single range per attribute results in a much coarser approximation.

Finding the optimal trade-off is non-trivial and has not been explicitly optimized in this work. Manual tuning is not a scalable solution, and we believe that future work should address this limitation. A possible direction could be to learn the block partitioning automatically, or to develop heuristics capable of estimating an effective configuration based on scene characteristics.

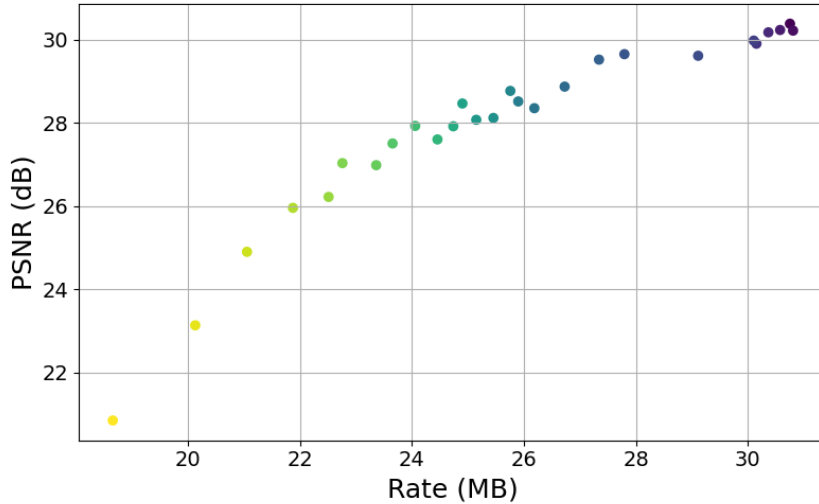


Figure 5.5: RD trade-off for different block sizes

5.3.5 Bit Allocation

Table 5.6 compares the proposed bit allocation scheme against two uniform quantization baselines using 8 and 6 bits per attribute, respectively. Compared to the 8-bit uniform allocation, our method achieves an average compression gain of $3.3\times$ with a negligible quality loss of only 0.32 dB, an imperceptible difference to the human eye. This indicates that certain attributes, particularly AC SH coefficients, contribute minimally to the final rendering and can be approximated coarsely without noticeable degradation.

In contrast, when compared to the 6-bit uniform allocation, our scheme outperforms both quality and rate, demonstrating that some attributes — especially scale parameters — require finer precision due to their high impact on rendering quality. These results confirm that different attributes vary in importance for rendering, and approximating them uniformly is sub-optimal. The proposed scheme effectively addresses this by adapting precision according to attribute significance.

Table 5.6: Our bit allocation scheme vs. uniform allocation

	Room		Truck		Mic	
	PSNR	Size	PSNR	Size	PSNR	Size
8 bits	<u>31.7</u>	26.15	<u>25.62</u>	44.87	<u>35.54</u>	7.44
6 bits	24.88	20.71	18.08	28.22	20.30	6.21
Ours	31.15	<u>9.09</u>	25.61	<u>10.85</u>	35.13	<u>2.57</u>

5.3.6 Sparsity-promoting Fine-tuning

Table 5.7 presents a comparison between our proposed fine-tuning scheme and two alternative strategies: (i) no fine-tuning, and (ii) fine-tuning without the sparsity-promoting term.

While skipping fine-tuning (i) still leads to a significant reduction in size compared to the original 3DGS – primarily due to pruning, voxelization, and RAHT-based quantization – it results in noticeable quality degradation. This drop in performance is due to the fact that the original 3DGS parameters are not optimized for the structured representation introduced by voxelization, nor are they robust to quantization noise, which leads to visible artifacts.

Table 5.7: Comparison of fine-tuning methods: (i) no fine-tuning, (ii) fine-tuning without sparsity loss, and proposed.

	Room		Truck		Mic	
	PSNR	Size	PSNR	Size	PSNR	Size
(i)	30.01	11.04	22.14	16.35	33.03	3.08
(ii)	<u>31.26</u>	11.03	25.55	16.36	35.12	3.07
Ours	31.15	<u>9.09</u>	<u>25.61</u>	<u>10.85</u>	<u>35.13</u>	<u>2.57</u>

Fine-tuning without the sparsity-promoting term (ii) improves rendering quality. This improvement stems from the fact that quantization-aware training allows the model to adjust its parameters to better match the transformed and quantized representation. Essentially, the representation adapts to the new 3D positions (after voxelization) and learns a configuration that is more resilient to quantization, demonstrating that the same scene can be represented by different sets of Gaussians with similar precision. However, since this strategy focuses solely on maintaining quality rather than improving compressibility, it results in negligible rate reduction and thus offers limited benefits for compression.

To better illustrate this behavior, Figure 5.6 shows RD curves for different numbers of fine-tuning iterations for the *Mic* and *Room* scenes. In both cases, two curves are plotted: one obtained with the sparsity-promoting term and one without it. While the quality (PSNR) eventually converges to similar levels, the key difference lies in the rate. When the sparsity-promoting loss is included, the rate consistently decreases with more iterations, indicating that the model is converging toward increasingly compact representations. In contrast, the model trained without sparsity term maintains a nearly constant rate, missing opportunities for further compression.

Overall, these results highlight the importance of our fine-tuning strategy. It does not merely improve robustness to compression but actively moves the model toward alternative

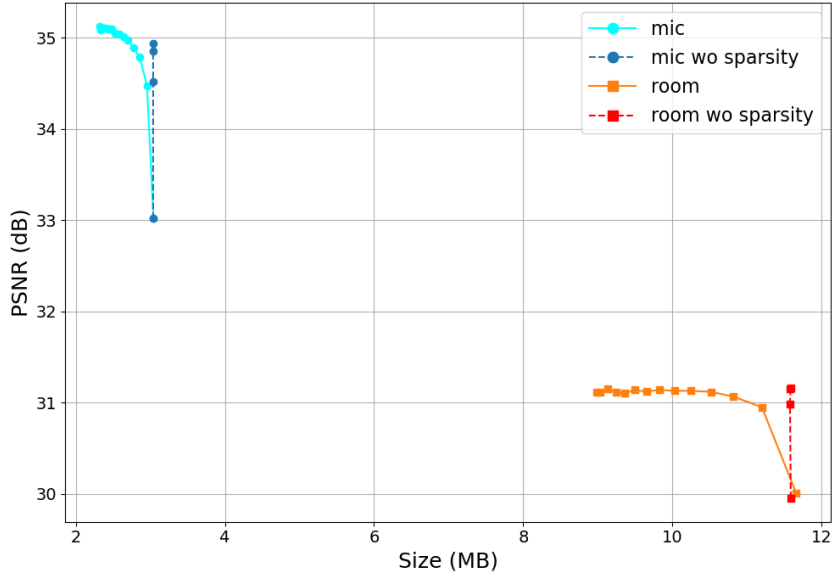


Figure 5.6: RD curves with and without sparsity-promoting term for different numbers of fine-tuning iterations

parameter configurations that preserve visual fidelity while enhancing compressibility.

5.3.7 Effects of varying λ_S

Finally, we conduct experiments to analyze how varying the relative importance of the two loss terms impacts the RD performance. By modifying the parameter λ_S in Equation 4.17, we can effectively control the trade-off between rate and distortion, and generate the corresponding RD curves for each scene, as shown in Figure 5.7. In particular, we report results for $\lambda_S \in \{5 \cdot 10^{-5}, 5 \cdot 10^{-6}, 5 \cdot 10^{-7}, 5 \cdot 10^{-8}\}$ across the three reference scenes.

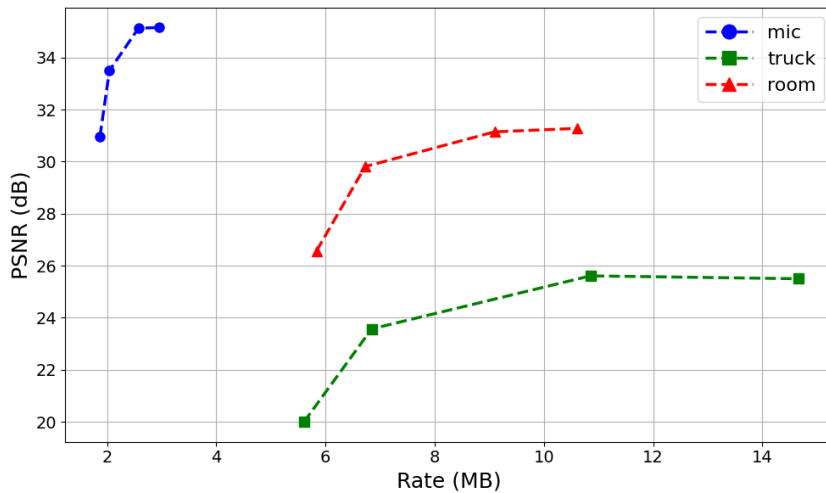


Figure 5.7: RD Curves for different λ_S values

This parameter λ_S directly controls the importance given to the sparsity-promoting term in the fine-tuning loss function. A higher value of λ_S encourages the model to produce sparser

transform coefficients, making the signal more compressible and resulting in lower bitrate. However, this typically comes at the expense of a slight degradation in visual quality, as the model prioritizes compression over precise reconstruction. On the other hand, a smaller value of λ_S reduces the influence of the sparsity term, focusing the optimization more on preserving rendering quality, but leading to denser coefficients and, consequently, higher bitrates.

This tunable mechanism gives our method a significant advantage: it enables flexible navigation of the RD curve depending on the specific needs of the application. In contrast to most existing compression methods, which typically target a single operating point on the RD curve and offer limited adaptability, our framework supports this kind of dynamic adjustment. This is a typical strength of learning-based methods [61], and in our case it is achieved through a simple modification of a single scalar hyperparameter.

Chapter 6

Conclusions

In this thesis, we addressed the problem of compressing 3D Gaussian Splatting representations, with the goal of enabling compact storage and efficient transmission without compromising rendering quality. We approached the problem by treating 3DGS as a form of Point Cloud enriched with a high-dimensional set of attributes. This interpretation allowed us to leverage and adapt the RAHT, traditionally used in Point Cloud compression.

Our method builds upon this foundation with two key contributions. First, we introduced a novel bit allocation strategy tailored to the characteristics of each attribute channel. Inspired by quantization matrices used in traditional image codecs like JPEG, we assigned different quantization steps across channels to account for their differing visual importance and value ranges. This adaptive quantization approach enables efficient compression while preserving perceptual quality.

Second, we integrated quantization into the optimization loop via quantization-aware fine-tuning. By employing the STE, we were able to maintain differentiability, allowing the network to adjust parameters in a way that compensates for quantization noise. We also introduced a sparsity-promoting loss term, based on the ℓ_1 norm of the transform coefficients. This loss encouraged the model to concentrate energy in fewer coefficients, facilitating entropy coding and improving rate-distortion performance.

Our experimental evaluation on both the Mip-NeRF360, Tanks and Temples and NeRF Synthetic datasets demonstrated that the proposed method offers state-of-the-art compression performance among post-training 3DGS approaches. Notably, the method maintains compatibility with standard Point Cloud coding tools — such as the MPEG GPCC codec — making it highly deployable. While some recent neural methods may outperform our approach in absolute metrics, our solution achieves a strong balance between performance, simplicity, and

compatibility with existing standards.

In summary, this work demonstrates that traditional signal processing tools, when combined with modern learning-based techniques, can advance the state of 3DGS compression. Our hybrid framework provides a principled and extensible solution, offering a valuable step toward scalable, high-quality novel view synthesis.

6.1 Open Issues

Our method preserves the rendering efficiency of the baseline, so NVS is performed at comparable speeds. However, compression time is significantly increased due to the inclusion of quantization-aware fine-tuning, which applies quantization at every training iteration. This overhead makes the current system less suitable for time-sensitive or real-time compression scenarios.

Several factors contribute to this limitation. First, the operation is inherently complex, as it involves quantizing a large number of Gaussians, each with multiple attributes. Furthermore, our approach employs block-wise quantization, which requires a distinct quantizer per block, adding computational complexity. A more thorough analysis of the trade-off between the number of blocks and compression efficiency could help reduce this overhead.

Nevertheless, the most significant bottleneck lies in the current implementation, which is not fully optimized. In particular, each attribute channel is processed sequentially without parallelization, leading to inefficiencies. Introducing parallel processing could substantially accelerate compression and improve system usability.

Similarly, the application of the transform and its inverse at each fine-tuning iteration introduces additional computational cost. While the current implementation leverages sparse matrix multiplications for efficiency, the construction of the transformation matrix itself remains time-consuming — especially at high octree depths — and could benefit from further optimization.

Finally, in order to obtain a fair comparison across different methods, all experiments should ideally be conducted using the same baseline, training procedure, and hardware setup. Relying solely on results reported in previous publications may introduce discrepancies due to varying experimental conditions. Future evaluations should aim for a more consistent and controlled benchmarking environment.

6.2 Future Works

Several directions remain open to improve the efficiency, flexibility, and overall performance of the proposed framework:

Efficient Quantization. As said, one key limitation of the current implementation lies in the quantization stage, which significantly slows down the fine-tuning process. Future work will focus on developing a more efficient quantization mechanism that enables faster training while preserving accuracy and RD performance.

Parameter Optimization. Some hyperparameters, such as the number of quantization blocks, are currently taken from the literature without additional tuning. We aim to design methods that can automatically optimize such parameters on a per-scene basis. Ideally, future systems should be able to "learn" these configurations adaptively, improving generalization across diverse scene types without requiring manual tuning.

Adaptive Voxelization. The voxelization stage could benefit from adopting more flexible approaches, such as the adaptive voxelization method proposed in [60]. Their results demonstrate improved RD performance compared to standard uniform voxel grids, making integration with our framework a promising avenue for future exploration.

Scene-Adaptive Bit Allocation. A promising direction is to introduce more sophisticated, scene-aware bit allocation strategies. For example, one could learn optimal quantization steps per attribute or per block, potentially using a differentiable quantization framework as proposed in [62]. This would allow the model to dynamically allocate bits where they matter most for rendering fidelity.

Improved Loss Formulations. Another opportunity lies in refining the loss function to promote compressibility more directly. One possible extension includes incorporating an explicit rate term based on the entropy of the transformed coefficients. While this introduces additional computational overhead (e.g., for probability estimation), it can provide a more accurate signal for rate control and lead to better RD trade-offs.

Advanced Entropy Coding. The current use of Deflate is convenient but not optimal. Future implementations could benefit from more advanced entropy coding schemes tailored to the statistics of the transformed signal. For example, integrating arithmetic coders similar to

those used in GPCC [12] may significantly improve compression efficiency by better modeling symbol distributions at the bitstream level.

Acronym

NeRF Neural Radiance Fields for View Synthesis

MLP Multilayer Perceptron

SSIM Structural Similarity Index

PSNR Peak Signal to Noise Ratio

LPIPS Learned Perceptual Image Patch Similarity

MB Megabytes

SH Spherical Harmonic

3DGS 3D Gaussian Splatting

NVS Novel View Synthesis

MSE Mean Square Error

MPEG Moving Picture Experts Group

GPCC Geometry based Point Cloud Compression

VPCC Video based Point Cloud Compression

LoD Levels of Detail

SPIHT Set Partitioning in Hierarchical Trees

LBG Linde-Buzo-Gray

KNN K-Nearest Neighbors

RAHT Region Adaptive Hierarchical Transform

VR Virtual Reality

AR Augmented Reality

XR eXtended Reality

RD Rate-Distortion

WT Wavelet Transform

EZW Embedded Zerotree Wavelet

STE Straight-Through Estimator

Bibliography

- [1] Tao Lu, Mulin Yu, Linning Xu, Yuanbo Xiangli, Limin Wang, Dahua Lin, and Bo Dai. Scaffold-gs: Structured 3d gaussians for view-adaptive rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 20654–20664, 2024.
- [2] Ricardo L. de Queiroz and Philip A. Chou. Compression of 3d point clouds using a region-adaptive hierarchical transform. *IEEE Transactions on Image Processing*, 25(8):3947–3956, 2016.
- [3] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics*, 42(4), July 2023.
- [4] Shuzhao Xie, Weixiang Zhang, Chen Tang, Yunpeng Bai, Rongwei Lu, Shijia Ge, and Zhi Wang. Mesongs: Post-training compression of 3d gaussians via efficient attribute transformation. In *European Conference on Computer Vision*. Springer, 2024.
- [5] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis, 2020.
- [6] Yihang Chen, Qianyi Wu, Weiyao Lin, Mehrtash Harandi, and Jianfei Cai. Hac: Hash-grid assisted context for 3d gaussian splatting compression. In *European Conference on Computer Vision*, 2024.
- [7] Yu-Ting Zhan, Cheng-Yuan Ho, Hebi Yang, Yi-Hsin Chen, Jui Chiu Chiang, Yu-Lun Liu, and Wen-Hsiao Peng. Cat-3dgs: A context-adaptive triplane approach to rate-distortion-optimized 3dgs compression, 2025.
- [8] Sharath Girish, Kamal Gupta, and Abhinav Shrivastava. Eagles: Efficient accelerated 3d gaussians with lightweight encodings, 2024.

- [9] 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.
- [10] Yihang Chen, Qianyi Wu, Mengyao Li, Weiyao Lin, Mehrtash Harandi, and Jianfei Cai. Fast feedforward 3d gaussian splatting compression, 2025.
- [11] He Huang, Wenjie Huang, Qi Yang, Yiling Xu, and Zhu li. A hierarchical compression technique for 3d gaussian splatting compression, 2025.
- [12] MPEG 3DG. G-pcc codec description v4. Technical Report N18673, ISO/IEC JTC1/SC29/WG11, September 2019. Geometry-based Point Cloud Compression (G-PCC).
- [13] Shenchang Eric Chen and Lance Williams. *View Interpolation for Image Synthesis*. Association for Computing Machinery, New York, NY, USA, 1 edition, 2023.
- [14] Li Li, Zhu Li, Shan Liu, and Houqiang Li. Plenoptic point cloud compression using multiview extension of high efficiency video coding. *IEEE Transactions on Multimedia*, 25:2007–2021, 2023.
- [15] Alex Yu, Sara Fridovich-Keil, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. Plenoxels: Radiance fields without neural networks, 2021.
- [16] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Trans. Graph.*, 41(4):102:1–102:15, July 2022.
- [17] Zhou Wang, A.C. Bovik, H.R. Sheikh, and E.P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, 2004.
- [18] David A Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, 2007.
- [19] J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23(3):337–343, 1977.
- [20] Andrew B Watson et al. Image compression using the discrete cosine transform. *Mathematica journal*, 4(1):81, 1994.

- [21] William B. Pennebaker and Joan L. Mitchell. *JPEG Still Image Data Compression Standard*. Van Nostrand Reinhold, New York, 1992.
- [22] R. Srinivasan and K. Rao. Predictive coding based on efficient motion estimation. *IEEE Transactions on Communications*, 33(8):888–896, 1985.
- [23] D. Graziosi, O. Nakagami, S. Kuma, A. Zaghetto, T. Suzuki, and A. Tabatabai. An overview of ongoing point cloud compression standardization activities: video-based (v-pcc) and geometry-based (g-pcc). *APSIPA Transactions on Signal and Information Processing*, 9:e13, 2020.
- [24] MPEG 3DG. V-pcc test model v8. Test Model N18884, ISO/IEC JTC1/SC29/WG11, Geneva, Switzerland, October 2019. Video-based Point Cloud Compression (V-PCC).
- [25] Iain E. G. Richardson. H.264/mpeg-4 part10 white paper: Transform & quantization. White Paper —, Joint Video Team (JVT), March 2003. © 19 March 2003.
- [26] Milena T. Bagdasarian, Paul Knoll, Yi-Hsin Li, Florian Barthel, Anna Hilsmann, Peter Eisert, and Wieland Morgenstern. 3DGS.zip: A survey on 3D Gaussian Splatting Compression Methods, 2025.
- [27] Muhammad Salman Ali, Chaoning Zhang, Marco Cagnazzo, Giuseppe Valenzise, Enzo Tartaglione, and Sung-Ho Bae. Compression in 3D Gaussian Splatting: A Survey of Methods, Trends, and Future Directions, 2025.
- [28] 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), May 2024.
- [29] Zhiwen Fan, Kevin Wang, Kairun Wen, Zehao Zhu, DeJia Xu, and Zhangyang Wang. Lightgaussian: Unbounded 3d gaussian compression with 15x reduction and 200+ fps, 2023.
- [30] Henan Wang, Hanxin Zhu, Tianyu He, Runsen Feng, Jiajun Deng, Jiang Bian, and Zhibo Chen. End-to-end rate-distortion optimized 3d gaussian representation. In *European Conference on Computer Vision*, 2024.
- [31] Anil Armagan, Albert Saà-Garriga, Bruno Manganelli, Mateusz Nowak, and Mehmet Kerim Yucel. Trick-gs: A balanced bag of tricks for efficient gaussian splatting, 2025.

- [32] 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.
- [33] KL Navaneet, Kossar Pourahmadi Meibodi, Soroush Abbasi Koochpayegani, and Hamed Pirsiavash. Compgs: Smaller and faster gaussian splatting with vector quantization. *ECCV*, 2024.
- [34] S. Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28(2):129–137, 1982.
- [35] Y. Linde, A. Buzo, and R. Gray. An algorithm for vector quantizer design. *IEEE Transactions on Communications*, 28(1):84–95, 1980.
- [36] Simon Niedermayr, Josef Stumpfegger, and Rüdiger Westermann. Compressed 3d gaussian splatting for accelerated novel view synthesis, 2023.
- [37] Xiangrui Liu, Xinju Wu, Pingping Zhang, Shiqi Wang, Zhu Li, and Sam Kwong. Compgs: Efficient 3d scene representation via compressed gaussian splatting, 2024.
- [38] Wieland Morgenstern, Florian Barthel, Anna Hilsmann, and Peter Eisert. Compact 3d scene representation via self-organizing gaussian grids. In *Computer Vision – ECCV 2024*, pages 18–34, Cham, 2025. Springer Nature Switzerland.
- [39] Soonbin Lee, Fangwen Shu, Yago Sanchez, Thomas Schierl, and Cornelius Hellge. Compression of 3d gaussian splatting with optimized feature planes and standard video codecs, 2025.
- [40] Johannes Ballé, David Minnen, Saurabh Singh, Sung Jin Hwang, and Nick Johnston. Variational image compression with a scale hyperprior, 2018.
- [41] Chang Li Yan, Quan Cai Deng, and Xin Zhang. Image compression based on wavelet transform. *Applied Mechanics and Materials*, 568:749–752, 2014.
- [42] S Piramu Kailasam. Efficient haar wavelet transform with embedded zerotrees of wavelet compression for color images. *International Journal of Computer and Information Engineering*, 14(11):406–412, 2020.
- [43] Athanassios Skodras, Charilaos Christopoulos, and Touradj Ebrahimi. The jpeg 2000 still image compression standard. *IEEE Signal processing magazine*, 18(5):36–58, 2002.

- [44] Kamrul Hasan Talukder and Koichi Harada. Haar wavelet based approach for image compression and quality assessment of compressed image, 2010.
- [45] Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. GPTQ: Accurate Post-Training Quantization for Generative Pre-trained Transformers, 2023.
- [46] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation, 2013.
- [47] Khalid Sayood. *Introduction to Data Compression*. Morgan Kaufmann, 5th edition, 2017.
- [48] Emmanuel J Candes, Michael B Wakin, and Stephen P Boyd. Enhancing sparsity by reweighted l_1 minimization. *Journal of Fourier analysis and applications*, 14(5):877–905, 2008.
- [49] Francis Bach, Rodolphe Jenatton, Julien Mairal, Guillaume Obozinski, et al. Optimization with sparsity-inducing penalties. *Foundations and Trends® in Machine Learning*, 4(1):1–106, 2012.
- [50] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 58(1):267–288, 1996.
- [51] Robert Tibshirani, Michael Saunders, Saharon Rosset, Ji Zhu, and Keith Knight. Sparsity and smoothness via the fused lasso. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 67(1):91–108, 2005.
- [52] Savan Oswal, Anjali Singh, and Kirthi Kumari. Deflate compression algorithm. *International Journal of Engineering Research and General Science*, 4(1):430–436, 2016.
- [53] A. Said and W.A. Pearlman. A new, fast, and efficient image codec based on set partitioning in hierarchical trees. *IEEE Transactions on Circuits and Systems for Video Technology*, 6(3):243–250, 1996.
- [54] André L Souto, Victor F Figueiredo, Philip A Chou, and Ricardo L de Queiroz. Set partitioning in hierarchical trees for point cloud attribute compression. *IEEE Signal Processing Letters*, 28:1903–1907, 2021.
- [55] Jonathan T. Barron, Ben Mildenhall, Dor Verbin, Pratul P. Srinivasan, and Peter Hedman. Mip-NeRF 360: Unbounded Anti-Aliased Neural Radiance Fields, 2022.

- [56] Arno Knapitsch, Jaesik Park, Qian-Yi Zhou, and Vladlen Koltun. Tanks and temples: benchmarking large-scale scene reconstruction. *ACM Trans. Graph.*, 36(4), July 2017.
- [57] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis. In *ECCV*, 2020.
- [58] Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric, 2018.
- [59] Muhammad Salman Ali, Maryam Qamar, Sung-Ho Bae, and Enzo Tartaglione. Trimming the Fat: Efficient Compression of 3D Gaussian Splats through Pruning, 2024.
- [60] Chenjunjie Wang, Shashank N. Sridhara, Eduardo Pavez, Antonio Ortega, and Cheng Chang. Adaptive voxelization for transform coding of 3d gaussian splatting data, 2025.
- [61] Johannes Ballé, Valero Laparra, and Eero P. Simoncelli. End-to-end Optimized Image Compression, 2017.
- [62] Thierry Dumas, Aline Roumy, and Christine Guillemot. Autoencoder based image compression: Can the learning be quantization independent? In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1188–1192, 2018.

