SizeGS: Size-aware Compression of 3D Gaussians with Hierarchical Mixed Precision Quantization

Shuzhao Xie[1]*, Jiahang Liu[2]*, Weixiang Zhang[1], Shijia Ge[1], Sicheng Pan[1], Chen Tang[3], Yunpeng Bai[4], Zhi Wang[1]†
[1]SIGS&TBSI, Tsinghua University [2]Harbin Institute of Technology, Shenzhen
[3]MMLab, Chinese University of Hong Kong [4]The University of Texas at Austin
https://shuzhaoxie.github.io/sizegs

Abstract

Effective compression technology is crucial for 3DGS to adapt to varying storage and transmission conditions. However, existing methods fail to address size constraints while maintaining optimal quality. In this paper, we introduce SizeGS, a framework that compresses 3DGS within a specified size budget while optimizing visual quality. We start with a size estimator to establish a clear relationship between file size and hyperparameters. Leveraging this estimator, we incorporate mixed precision quantization (MPQ) into 3DGS attributes, structuring MPQ in two hierarchical levels—inter-attribute and intra-attribute—to optimize visual quality under the size constraint. At the inter-attribute level, we assign bit-widths to each attribute channel by formulating the combinatorial optimization as a 0-1 integer linear program, which can be efficiently solved. At the intra-attribute level, we divide each attribute channel into blocks of vectors, quantizing each vector based on the optimal bit-width derived at the inter-attribute level. Dynamic programming determines block lengths. Using the size estimator and MPQ, we develop a calibrated algorithm to identify optimal hyperparameters in just 10 minutes, achieving a 1.69× efficiency increase with quality comparable to state-of-the-art methods. We will release the source code.

1. Introduction

In recent years, 3D Gaussian Splatting (3DGS) [18] has revolutionized 3D scene representation and has been widely adopted in a variety of applications [4, 21, 23, 41, 51]. By re-parameterizing point clouds into 3D Gaussian functions, 3DGS has achieved unprecedented reconstruction quality. Moreover, with efficient CUDA implementation, real-time rendering is enabled. Despite its success, 3DGS still faces limitations in storage efficiency. To address this, various 3DGS compression methods have been proposed [2]. However, these methods primarily aim to improve compression quality, often overlooking requirements arising from applications such as volumetric video streaming [27, 41, 48] and remote teleoperation [21]. These applications frequently encounter fluctuating network bandwidth [8], leading to jitter and blurriness that significantly impact user experience [7, 24]. Consequently, compressing 3DGS to accommodate varying network conditions is essential for enhancing the quality of service in these applications.

We define this requirement as the task of efficiently identifying a set of hyperparameters that compresses 3DGS to a specified size while maximizing visual quality. Achieving this goal involves two primary challenges: 1) How to generate hyperparameters based on the size budget, which requires establishing a precise mapping between size and hyperparameters. 2) Given multiple sets of hyperparameters that meet the size budget, how to effectively select the one that can maximize visual quality. This step requires quickly estimating the highest achievable quality for each set of hyperparameters.

To this end, we propose *SizeGS*, which incorporates a size estimator to enable size-constrained hyperparameter generation and introduces a hierarchical mixed precision quantization scheme to identify the optimal configuration within the size budget. First, to generate hyperparameters based on the size budget, the key lies in constructing a compression framework that ensures a simple relationship between hyperparameters and size. Current compression frameworks fall into two categories: offline [11, 32, 33, 44] and online [6, 20, 26, 30, 33, 39, 42, 43, 45]. Offline methods perform compression without additional training, ensuring time efficiency while also supporting fine-tuning for improved visual quality. In contrast, online methods require training during file compression. This is because some components of the compressed model, such as the learnable mask [20, 39] or context model [6, 42], must be trained

*Equal contributions.
†Corresponding author.

from scratch. This dependency makes it difficult to predict the final size of the compressed model, complicating the relationship between hyperparameters and model size. Therefore, we choose an offline method, MesonGS [44]. Based on this model, we define size as a multivariable function of hyperparameters, including octree depth, reserve ratio, and the number of blocks. Thus, generating hyperparameters within a given size budget becomes an indeterminate equation. To simplify solving this equation, we fix the hyperparameters for octree depth and the number of blocks, focusing only on solving for the reserve ratio.

Second, to maximize compression quality, enhancing the marginal compression performance of the chosen framework is the primary consideration. We choose ScaffoldGS [28] as the base model and upgrade the uniform block quantization in MesonGS to hierarchical mixed-precision quantization (H-MPQ). This method divides each channel of the attribute matrix into multiple blocks, requiring bit-width settings for each channel and length settings for each block, thus introducing additional hyperparameters. To efficiently determine these MPQ settings, we use the mean square error between the original and restored data to measure information loss during compression. By minimizing this information loss and setting the size budget as a constraint, we formulate bit-width searching as a combinatorial optimization problem and effectively solve it with 0-1 linear programming. For block-length settings, we use dynamic programming based on information loss. Based on the size estimator and H-MPQ, we finally design a calibrated searching algorithm to find a hyperparameter set that can fulfill the size budget while maximizing quality.

In summary, our contributions are as follows: **1)** We define the task of compressing 3DGS to a target size and introduce an estimator to predict the compressed size, enabling efficient hyperparameter search within the size budget. **2)** We propose a size-aware hierarchical mixed-precision quantization scheme. At the inter-attribute level, bit-width selection is formulated as a 0-1 ILP problem, while at the intra-attribute level, a dynamic programming algorithm partitions attribute channels into variable-length blocks. **3)** We design a calibrated algorithm that identifies size-constrained hyperparameter settings within 10 minutes, achieving 1.69× faster performance than state-of-the-art methods, with comparable or superior results across multiple datasets.

2. Related Work

2.1. 3D Gaussian Splatting and Its Compression

3D Gaussian Splatting [18] is proposed for reconstructing 3D scenes from 2D images, which represents a scene using a set of 3D Gaussian distributions. These Gaussians capture the density, color, and opacity of the scene. It is faster than previous methods [29] and can produce high-quality, smooth results with fewer artifacts. Recently, a lot of work has been proposed for compressing 3D Gaussians. At first, they focused on compressing the original 3DGS model [11, 15, 20, 30–33, 39, 43, 44]. Then, many works pay attention to compressing a more efficient GS model [1, 12, 28, 34], in which ScaffoldGS [28] became the hot spot. It proposed to divide anchors into voxels and introduce an anchor feature for each voxel to grasp the common attributes of neural Gaussians in the voxel, i.e., the neural Gaussians are predicted by the anchor features. HAC [6] proposed a tailored compression method for it, extracting a context from the 3D coordinates to guide the quantization and entropy encoding. ContextGS [42] divides anchors into hierarchical levels and encodes them progressively.

However, these compression frameworks for ScaffoldGS require training, and thus, the size varies greatly over time, making size estimation under given hyperparameters challenging. This paper transfers the relatively easy-to-estimate MesonGS to ScaffoldGS and conducts an in-depth analysis of the compression components of MesonGS to build a nearly delay-free size estimator. Though FCGS [5], a contemporary work, can produce a compressed model through a single inference of a feed-forward model in 16 seconds on an NVIDIA 3090, the feed-forward model can only compress a pre-trained 3DGS to one specific size. It cannot compress the same 3DGS to a different size, lacking the flexibility to adjust the compressed file size.

2.2. Mixed Precision Quantization

Mixed Precision quantization (MPQ) is a widely-used technique to improve the trade-off between the accuracy and efficiency of neural networks [9, 10, 37, 40, 46]. The challenge with this approach is to find the right mixed-precision setting for the different layers of neural networks. A brute force approach is not feasible since the search space is exponentially large in the number of layers. HAQ [40] employed reinforcement learning to search this space. However, this RL-based solution requires tremendous computational resources. HAWQ [9, 10, 46] proposed first to assign each layer a sensitivity score with the Hessian spectrum and then formulate an ILP solution that can generate mixed-precision settings with various constraints (such as model size and latency). By identifying the opportunity to employ MPQ to the attributes of the GS model to enhance the compression quality, we propose a hierarchical scheme to quickly determine the optimal mixed-precision settings. Besides, 3DGS compression has 16 bit options available, far exceeding 2 bit options of HAWQ (INT4 and INT8). Directly using the integer programming formulation from HAWQ3 [46] makes it difficult to achieve good results. To address this, we establish a more general and fast 0-1 integer programming formulation to determine the optimal bit-width for each at-

tribute channel. CA-NeRF [25] also uses the MPQ scheme. However, their goal is to reduce the memory requirement of NeRF, and it is not suitable for reducing the storage of 3DGS. Although HAC [6] employs MPQ, its granularity is relatively coarse, whereas our method employs a finer-grained, intra-channel quantization. More illustrations are provided in the supplementary material.

3. Preliminary

3DGS [18] is an explicit 3D representation in the form of point clouds, utilizing Gaussians to model the points. Each Gaussian is characterized by a covariance matrix $\mathbf{\Sigma}$ and a center point μ, which is referred to as the mean value of the Gaussian: $G(x) = e^{-\frac{1}{2}(x-\mu)^\top \mathbf{\Sigma}^{-1}(x-\mu)}$. To maintain the positive definiteness of the covariance matrix $\mathbf{\Sigma}$, 3DGS decomposes $\mathbf{\Sigma}$ into a scaling matrix $\mathbf{S} = \text{diag}(\mathbf{s}), \mathbf{s} \in \mathbb{R}^3$ and a rotation matrix $\mathbf{R}$: $\mathbf{\Sigma} = \mathbf{RSS}^\top\mathbf{R}^\top$. The rotation matrix $\mathbf{R}$ is parameterized by a rotation quaternion $\mathbf{q} \in \mathbb{R}^4$. The backpropagation process is illustrated in [18].

When rendering novel views, the technique of splatting [47, 52] is employed for the Gaussians within the camera planes. As introduced by [53], using a viewing transform denoted as $\mathbf{W}$ and the affine transform $\mathbf{J}$, the covariance matrix $\mathbf{\Sigma}'$ in camera coordinates system can be computed by $\mathbf{\Sigma}' = \mathbf{JW\Sigma W}^\top\mathbf{J}^\top$.

In summary, each element of 3D Gaussians has the following parameters: (1) a 3D center $\mu \in \mathbb{R}^3$; (2) a rotation quaternion $\mathbf{q} \in \mathbb{R}^4$; (3) a scale vector $\mathbf{s} \in \mathbb{R}^3$; (4) a color feature defined by spherical harmonics coefficients $\mathbf{SH} \in \mathbb{R}^h$, with $h = 3(d+1)^2$, where d is the harmonics degree; and (5) an opacity logit $o \in \mathbb{R}$. Specifically, for each pixel, the color and opacity of Gaussians are computed using $G(x)$. The blending of N ordered points that overlap the pixel is given by: $C = \sum_{i \in N} c_i \alpha_i \prod_{j=1}^{i-1}(1-\alpha_j)$. Here, c_i and α_i represent the density and color of this point computed by a Gaussian with covariance $\mathbf{\Sigma}$ multiplied by a per-point opacity and SH color coefficients.

Scaffold-GS [28] is a variant of 3DGS, widely adopted in 3DGS compression due to its low storage requirements. It introduces *anchor points* to capture common attributes of local 3D Gaussians. Specifically, the *anchor points* are initialized from neural Gaussians by voxelizing the 3D scenes. Each anchor point has a context feature $\mathbf{f} \in \mathbf{R}^{32}$, a location $\mathbf{x} \in \mathbf{R}^3$, a scaling factor $\mathbf{l} \in \mathbf{R}^6$ and k learnable offset $\mathbf{O} \in \mathbf{R}^{k\times 3}$. Given a camera at $\mathbf{x}_c$, anchor points are used to predict the view-dependent neural Gaussians in their corresponding voxels as follows,

$$\{\mathbf{c}^i, \mathbf{r}^i, \mathbf{s}^i, \alpha^i\}_{i=0}^k = \text{MLP}(\mathbf{f}, \boldsymbol{\sigma}_c, \vec{\mathbf{d}}_c), \tag{1}$$

where $\boldsymbol{\sigma}_c = ||\mathbf{x} - \mathbf{x}_c||_2$, $\vec{\mathbf{d}}_c = \frac{\mathbf{x}-\mathbf{x}_c}{||\mathbf{x}-\mathbf{x}_c||_2}$, the superscript i represents the index of neural Gaussian in the voxel, $\mathbf{s}^i, \mathbf{c}^i \in \mathbf{R}^3$ are the scaling and color respectively, and $\mathbf{r}^i \in \mathbf{R}^4$ is the quaternion for rotation. In the left side of Fig. 1, the positions of neural Gaussians are then calculated as

$$\{\boldsymbol{\mu}^0, ..., \boldsymbol{\mu}^{k-1}\} = \mathbf{x} + \{\mathbf{O}^0, ..., \mathbf{O}^{k-1}\} \cdot \mathbf{l}_{:3}, \tag{2}$$

where $\mathbf{x}$ is the learnable positions of the anchor and $\mathbf{l}_{:3}$ is the base scaling of its associated neural Gaussians. After decoding the properties of neural Gaussians from anchor points, the remaining steps are the same as 3DGS [18]. By predicting the properties of neural Gaussians from the anchor features and saving the properties of anchor points only, Scaffold-GS greatly eliminates the redundancy among 3D neural Gaussians and decreases the storage demand.

4. Methodology

In this section, we introduce the compression pipeline and the size estimator, followed by inter-attribute and intra-attribute mixed-precision quantization. Building on these components, we propose an algorithm to optimize hyperparameter settings that satisfy size constraints while maximizing visual quality.

4.1. Compression Pipeline and Size Estimator

Estimating the file size for compressed 3DGS is nontrivial. First, the difficulty of size estimation varies across different frameworks. We observe that the more modules that require training, the harder it becomes to estimate the final file size, especially for the online methods. For example, the learnable masking module proposed by [20] requires training; in HAC [6], context information extracted from the coordinates needs to be trained from scratch, as well as the hyperparameters used for entropy coding. These representations, which need to be learned from scratch, result in a compressed file size that is difficult to predict. In contrast, frameworks like MesonGS, which allow for offline compression, are easier to predict since none of the modules require training. As shown in Fig. 2a, the final file size of HAC fluctuates continuously during training, while the size of MesonGS remains stable with almost no variation. Therefore, we adopt the MesonGS framework for compression. Besides, we select ScaffoldGS as the base model as it has been adopted by multiple state-of-the-art 3DGS compression works [2, 6, 42].

We briefly introduce the proposed compression pipeline under a given configuration. In Fig. 1, assuming there are N anchor points, we first calculate the importance of each anchor point by averaging the importance of generated Gaussian splats. Then, we prune the anchor points based on the percentage τ, which means that τN anchor points are reserved. The coordinates of the anchor points are voxelized to form an octree. For anchor points within the same voxel, we average the attributes to ensure that each voxel corresponds to only one anchor point. Based

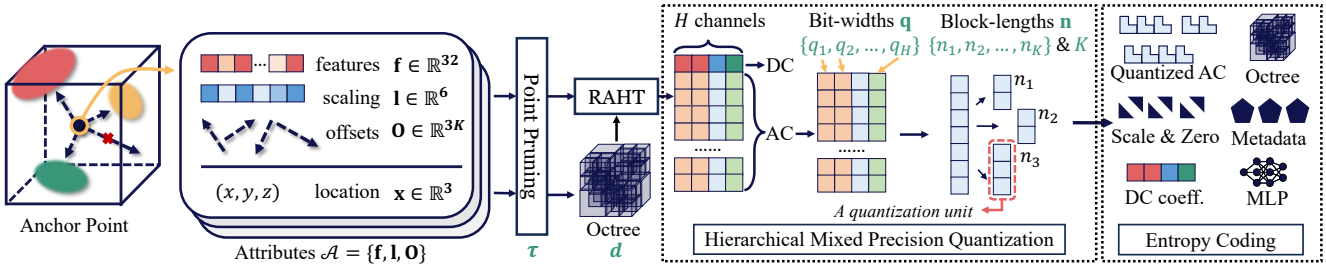
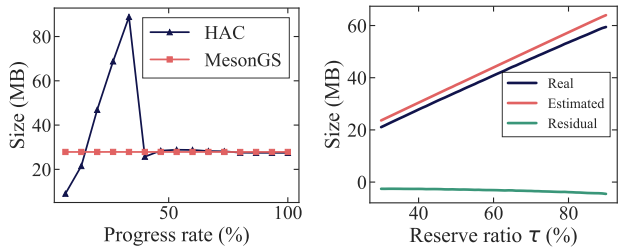


Figure 1. **Compression pipeline.** An anchor point contains a location and multiple attributes, including features, scaling, and offsets. Then, the figure shows the components of the pipeline, which involves pruning unimportant anchor points, performing voxelization on the locations to generate an octree, applying Region Adaptive Hierarchical Transform (RAHT) to the attributes to generate DC and AC coefficients, quantizing the AC coefficients using hierarchical mixed-precision quantization, and finally packing all the elements.



(a) Size fluctuation. (b) Size vs. τ.

Figure 2. **Motivations behind the size estimator.** (a): Compared to online methods (i.e., HAC), offline methods (i.e., MesonGS) have a more stable size during the finetuning. (b) τ vs. real / estimated size. Data are collected from the *bicycle* scene.

on the voxelized coordinates, we apply region-adaptive hierarchical transform (RAHT) to attributes $\mathcal{A} \in \mathbb{R}^{(\tau N) \times H}$, which produces the DC and AC coefficients. DC coefficients are stored in float format while the AC coefficients are then quantized by a hierarchical MPQ scheme. Specifically, each channel of the AC coefficients is partitioned into K blocks, whose lengths are configured by the start indices of blocks for the convenience of implementation, i.e. $\mathbf{n} = \{n_1, n_2, ..., n_K\}$. After that, each block, an individual quantization unit, is quantized with a given bit-width configured by $\mathbf{q} = \{q_1, q_2, ..., q_H\}$. Here, blocks in the same channel use the same bit-width. Finally, all components are compressed by LZ77 [14, 49, 50] codec. The metadata includes the octree depth d, the number of blocks K, and block-length configs for each channel.

As shown by the green symbols in Fig 1, the final file size is determined on the following hyperparameters, including the pruning percentage $\tau \in [0, 1]$, octree depth d, the number of blocks K, the bit-widths $\mathbf{q}$, and the block-lengths $\mathbf{n}$. Hence, we can estimate the size of the compressed file by

$$\begin{aligned} \mathcal{S} = \frac{1}{8}(\underbrace{(\tau N - 1)|\mathbf{q}|}_{\text{AC}} + \underbrace{64\tau N}_{\text{Octree}} + \underbrace{32H}_{\text{DC}} + \underbrace{32KH}_{\text{DP}} \\ + \underbrace{HK \times 32 \times 2}_{\text{Scales \& Zero points}} + \underbrace{2 \times 32}_{\text{depth}, K}) + \underbrace{50 \times 1024}_{\text{MLP}}. \end{aligned} \tag{3}$$

Here, $(\tau N - 1)|\mathbf{q}|$ refers to the estimated size of the compressed AC coefficients. As we use a 64-bit integer to store the 3D coordinate of an octree voxel, the total required storage is estimated as $64\tau N$. $32KH$ refers to the required storage for block-lengths. Other terms follow the same logic.

Observing Eq. 3, if the other parameters are fixed, it can be seen that estimated size S is a linear function of τ:

$$\mathcal{S}(\tau) = \frac{1}{8}(N|\mathbf{q}| + 64N)\tau + \mathcal{C}, \tag{4}$$

where $\mathcal{C}$ is the remainder in Eq. 3. To verify this linear relationship, we plot the τ and the corresponding actual file size and the estimated file size in Fig. 2b by fixing other hyperparameters. We can see these two lines exhibit a linear relationship with the same slope. However, there is still a constant difference between the two lines, which can be further corrected. Specifically, we complete a compression process to obtain the actual size, then compute and use the residual Δ to calibrate the size estimator: $\mathcal{S}^* = \mathcal{S} + \Delta$.

4.2. Inter-Attribute Mixed Precision Quantization

Based on the size estimator, we model the search problem for setting the mixed precision of attributes as a 0-1 programming problem to help search for other hyperparameters. We introduce MPQ here because although offline methods are more suitable for estimating the final size, prefixing hyperparameters hinder the model from achieving optimal performance. MesonGS also uses a uniform quantization precision, which prevents it from obtaining better compression results. Besides, attributes occupy over 90% of the final storage, further compressing them offers the greatest potential benefit.

Uniformly quantizing all the attributes to low-bitwidth (e.g. INT4) could lead to significant quality degradation. However, it is possible to benefit from low-precision quantization by keeping a subset of sensitive attributes at high precision. The basic idea is to keep sensitive channels at higher precision and insensitive layers at lower precision. An important component is that we directly consider the *size* metric, to select the bit-precision configuration. Previous methods have been unable to control the file size of the GS model. For example, given a desired file size, it has

been difficult to identify the bit configuration that is closest to this size while achieving the optimal quality. In this work, we formalize the problem as an Integer Linear Programming (ILP) problem.

Assume that for each attribute channel, there are B quantization options (e.g., 2 options for INT4 or INT8). For the AC coefficients with H attribute channels, the search space of the ILP is B^H. The objective of solving the ILP is to find the best bit configuration among these B^H possibilities that optimally balances information loss Ω and the user-specified file size. Each bit-precision setting can lead to a different model perturbation. To simplify the problem, we assume that the perturbations of each channel are independent of one another. This allows us to precompute the information loss of each channel separately, and it only requires BH computations. For the information loss metric, we use the mean square error between the original attributes and the restored attributes[1]. Formally, we can precompute the information loss matrix $\Omega \in \mathbb{R}^{H\times B}$ with:

$$\Omega(i,j) = \|\hat{\mathcal{A}}_i^j - \mathcal{A}_i\|^2. \tag{5}$$

We decompose $\mathbf{q}$ into $\mathbf{QB}$, where $\mathbf{Q} \in \{0,1\}^{H\times B}$ and $\mathbf{B} = [1, 2, ..., B]^\top$. B is set as 16. Then the 0-1 ILP problem tries to find the right bit precision $\mathbf{Q}$ that minimizes the information loss, as follows:

$$\begin{aligned} &\text{Objective: } \Omega \odot \mathbf{Q}, \\ &\text{Subject to: } \mathcal{S}(\mathbf{Q}) \leq \text{ Model Size Limit}, \\ &\qquad \forall i \in \{0, ..., H-1\}, \sum_{j=0}^{B} \mathbf{Q}_{i,j} = 1. \end{aligned} \tag{6}$$

Here, $S(\mathbf{Q})$ denotes the estimated file size, and $\odot$ means the Hadamard product. Note that we cannot solve the bit-width setting if there is no size estimator. Since the number of blocks and the bit-width setting influence each other, we fix the number of blocks and only adjust the bit-width. We solve this 0-1 ILP using open source PULP library [35].

4.3. Intra-Attribute Mixed Precision Quantization

Given a desired model size limitation, the ILP solver in Sec. 4.2 generates optimal intra-attribute bit precision configurations for different channels of attributes. In this part, we develop methods at the single attribute level to optimally slice a channel of attribute into K blocks. Our optimization goal is to minimize the permutation loss for each channel of the attribute when slicing a channel of attributes into multiple blocks. Hence, the size of each block does not have to be the same. Note that incorporating mixed block lengths does not significantly affect the final file size. For mixed block length quantization, we only need to record the starting index of each block to ensure decompression. The size of these indices is $32KH$.

[1] Similar assumption can be found in [9, 10].

Algorithm 1: Hyperparameter Searching.

Data: Size Budget and a pre-trained GS file.
Result: Optimal hyperparameters set Φ^*.

all_qbits[] = Carefully selected values of $|\mathbf{q}|$;
$\Omega^* = +\infty$;
for $|\mathbf{q}|$ *in* all_qbits[] **do**
  `/* Initialize hyperparameter set */`
  $\Phi = \{\}$;
  Solve out the τ based on Eq. 4: $\Phi = \{\tau\}$;
  Determine the octree depth d and number of blocks K based on τN: $\Phi = \{\tau, d, K\}$;
  Solve the 0-1 ILP to obtain the $\mathbf{q}$ and overall information loss Ω: $\Phi = \{\tau, d, K, \mathbf{q}, \mathbf{n}\}$;
  if $\Omega < \Omega^*$ **then**
    $\Omega^* = \Omega$;
    $\Phi^* = \Phi$;
`/* Calibrate the size estimator */`
Use Φ^* to finish encoding and obtain residual Δ;
Calibrate size estimator: $\mathcal{S}^* = \mathcal{S} + \Delta$;
Solve out $\mathbf{q}^*$ and $\mathbf{n}^*$ based on the calibrated estimator $\mathcal{S}^*$;
return $\Phi^* = \{\tau, d, K, \mathbf{q}^*, \mathbf{n}^*\}$;

Assume that we want to split a channel of attributes with length N into K blocks, noted as $\{b_1, b_2, ..., b_K\}$, and then quantize each block to q bits. Here q bits is the optimal bit-width setting for this channel of attributes that is solved by the 0-1 ILP. We denote the start index of each block as $\{n_1, n_2, ..., n_K\}$. Our goal is to minimize the information loss caused by block-wise quantization. Following the metrics proposed by mixed precision quantization for deep learning models [9, 10, 46], the minimal information loss for quantizing this channel of attribute is written as:

$$\mathcal{L}^* = \min_{n_1,n_2,...,n_K} \left\{ \sum_i \mathcal{L}(n_i, n_{i+1}) \right\}, \tag{7}$$

$$\mathcal{L}(n_i, n_{i+1}) = \frac{\|\hat{b}_i - b_i\|^2}{n_{i+1} - n_i}, \tag{8}$$

where $\hat{b}_i$ refers to the vector that dequantized from the quantized b_i. We measure the information loss of quantizing a block vector as the mean square error between $\hat{b}_i$ and b_i.

DP formulation. To find $\mathcal{L}^*$, we develop a dynamic programming (DP) algorithm. We use the function $\mathcal{F}(k, l)$ to represent the minimal total information loss when slicing the l elements into k blocks. We start with $\mathcal{F}(0, 0) = 0$, and derive the optimal substructure of F as follows:

$$\mathcal{F}(k, l) = \min_{0<i\leq l-k} \{\mathcal{L}(l-i, l) + \mathcal{F}(k-1, l-i)\}. \tag{9}$$

Complexity. Our DP algorithm first iterates over all possible k and l. Then, for each $\mathcal{F}(k, l)$, the DP algorithm traverses through $l - k$ combinations to select the one with the

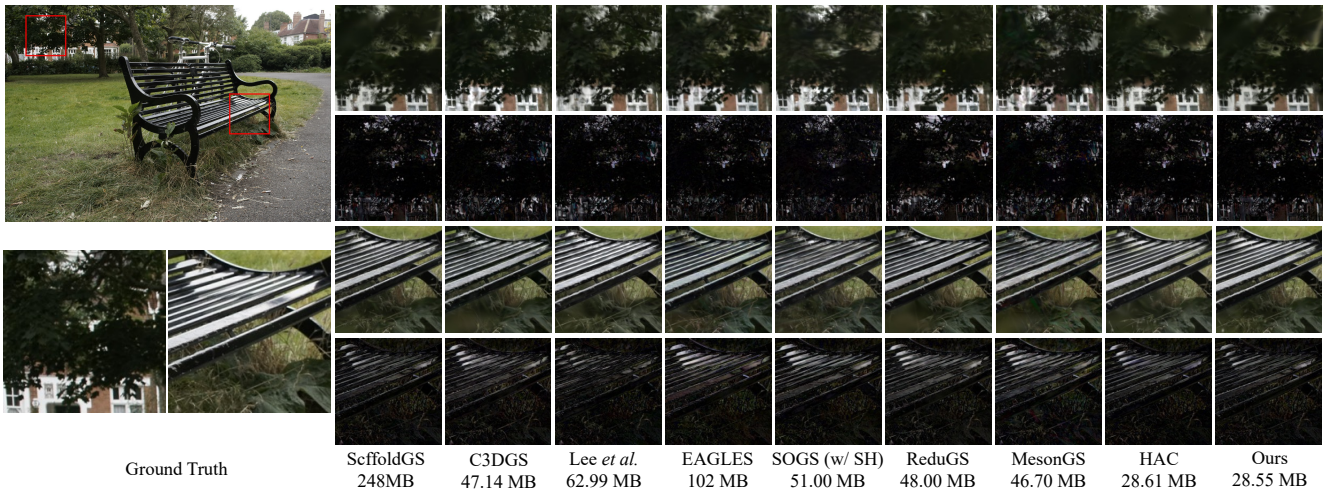


Figure 3. **Qualitative results**. We present the rendering results (rows 1 and 3) along with the corresponding error maps (rows 2 and 4) from two randomly selected viewpoints for the *bicycle* scene.

minimum loss. For each combination, we iterate through blocks of length i to compute the information loss. The overall complexity is $\mathcal{O}(KN_\tau(N_\tau - K)^2)$. Since $K \ll N_\tau$, the final complexity becomes $\mathcal{O}(K(N_\tau)^3)$. Here, $N_\tau = \tau N$, which is about $80,000$. Hence, this complexity is not feasible in practice. To accelerate this process, we limit the step size for each DP iteration to multiples of U, reducing the complexity to $\mathcal{O}(K(N_\tau/U)^3)$.

4.4. Searching Algorithm

The hyperparameter searching algorithm is shown in Algo. 1. We first iterate the carefully selected candidates of $|\mathbf{q}|$ and retain the set of hyperparameters with the smallest information loss. We use the rule of allowing only a 1% reduction in anchor points to calculate the octree depth. Since the number of blocks and the bit-width setting influence each other, and the model is relatively robust to the choice of K, we fix K to a constant value. Finally, for the selected hyperparameter set, we perform a full encoding process to calibrate the size estimator, then re-solve the 0-1 ILP and DP to update $\mathbf{q}$ and $\mathbf{n}$, yielding the final set of hyperparameters. After the search, we can use finetuning to enhance the quality, in which we fix the pruning mask and 3D coordinates. To accelerate block-wise quantization, we implement a parallel CUDA kernel. More details are proposed in the supplementary materials.

5. Experiments

Datasets. We conduct experiments on four datasets: 1) Mip-NeRF 360 [3]. This dataset contains five outdoor and four indoor scenes. Each scene contains 100 to 300 images. We use the images at 1600×1063. 2) Tank & Temples [19]. This dataset contains the *train* and *truck* scenes. 3) Deep Blending [17]. This dataset contains the *drjohnson* and *playroom* scenes. 4) Synthetic-NeRF [29]. This is a view synthesis dataset consisting of 8 synthetic scans, with 100 views used for training and 200 views for testing.

Baselines. We compare our method with the following baselines: 3DGS [18], ScaffoldGS [28], C3DGS [32], Lee *et al.* [20], LightGaussian [11], EAGLES [15], SOGS [30], Compact3D [31], ReduGS [33], MesonGS [44], HAC [6], DVGO [36], VQRF [22], and ACRF [13]. Quantitative results of baselines are derived from HAC [6], 3DGS.zip [2], and MesonGS [44], while the visual results are produced from our experiments.

5.1. Experimental Results

End-to-end Performance. Our method aims to automatically select hyperparameters to compress 3D Gaussians under a size budget while maximizing visual quality. We evaluate this ability of our method via latency and quality metrics. In Tab. 1, our method is at most $1.69\times$ faster than the baseline and achieves better or comparable quality across three datasets. For comparison with the baseline method, we fine-tuned for 6000 steps (approximately 8000 seconds). In practical applications, however, fine-tuning for only 500 steps is expected to achieve satisfactory visual quality. We perform a binary search on the hyperparameter λ of HAC to find a configuration that meets the size budget. The search stops when the difference between the obtained size and the size budget is within 1%. HAC requires significant time to fine-tune each potential hyperparameter configuration to determine the final file size. In contrast, we only need the time for a round of search plus one fine-tuning process. For more details, please refer to the supplementary materials.

Searching Time Decomposition. We divide the search algorithm in Algo. 1 into three stages for timing, as shown in Tab. 3: Stage 1 (lines 3–10), Stage 2 (lines 11–12), and Stage 3 (line 13). Additionally, we record the DP time and the total search time. It can be observed that a feasible

Method	Mip-NeRF 360 - 18.33 MB					Tank&Temples - 11 MB					Deep Blending - 8 MB				
	PSNR (dB)↑	SSIM ↑	LPIPS ↓	Size (MB) ↓	Time (s) ↓	PSNR (dB)↑	SSIM ↑	LPIPS ↓	Size (MB) ↓	Time (s) ↓	PSNR (dB)↑	SSIM ↑	LPIPS ↓	Size (MB) ↓	Time (s) ↓
HAC	27.17	0.789	0.261	18.29	16627	24.45	0.854	0.179	10.92	10978	30.27	0.910	0.254	8.29	9993
Our	27.44	0.804	0.242	18.28	9823	23.97	0.835	0.201	10.95	8701	30.22	0.906	0.261	7.99	8434

Table 1. **End-to-end performance in size-aware compression**. "Mip-NeRF 360 - 18.33 MB" refers to "dataset - size budget".

Method	Mip-NeRF 360				Tank&Temples				Deep Blending			
	PSNR (dB)	SSIM	LPIPS	Size (MB)	PSNR (dB)	SSIM	LPIPS	Size (MB)	PSNR (dB)	SSIM	LPIPS	Size (MB)
3DGS	27.49	0.813	0.222	744.7	23.69	0.844	0.178	431.0	29.42	0.899	0.247	663.9
ScaffoldGS	27.50	0.806	0.252	253.9	23.96	0.853	0.177	86.50	30.21	0.906	0.254	66.00
Lee *et al.*	27.08	0.798	0.247	48.80	23.32	0.831	0.201	39.43	29.79	0.901	0.258	43.21
C3DGS	26.98	0.801	0.238	28.80	23.32	0.832	0.194	17.28	29.38	0.898	0.253	25.30
EAGLES	27.15	0.808	0.238	68.89	23.41	0.840	0.200	34.00	29.91	0.910	0.250	62.00
LightGaussian	27.00	0.799	0.249	44.54	22.83	0.822	0.242	22.43	27.01	0.872	0.308	33.94
SOGS	26.01	0.772	0.259	23.90	22.78	0.817	0.211	13.05	28.92	0.891	0.276	8.40
Compact3d	27.16	0.808	0.228	50.30	23.47	0.840	0.188	27.97	29.75	0.903	0.247	42.77
ReduGS	27.10	0.809	0.226	29.00	23.57	0.840	0.188	14.00	29.63	0.902	0.249	18.00
MesonGS	26.98	0.801	0.233	28.77	23.32	0.837	0.193	16.99	29.51	0.901	0.251	24.76
HAC	27.53	0.807	0.238	15.26	24.04	0.846	0.187	8.10	29.98	0.902	0.269	4.35
Our-big	27.65	0.809	0.235	21.63	24.01	0.835	0.200	11.99	30.25	0.904	0.271	5.99
Our-middle	27.45	0.806	0.241	17.62	23.94	0.835	0.198	10.24	30.14	0.901	0.276	7.42
Our-small	27.21	0.799	0.251	13.54	23.80	0.834	0.202	8.28	29.82	0.896	0.287	4.53

Table 2. **Quantitative results.** The best and 2nd best results are highlighted in red and yellow cells. Note that we do not consider 3DGS and ScaffoldGS when highlighting this. The size is measured in MB.

Dataset	Stage1	Stage2	Stage3	DP	E2E (w/o DP)	E2E
Mip-NeRF 360	409	5	436	195	596	791
Tank&Temples	214	4	283	33	468	501
Deep Blending	176	5	253	117	434	551

Table 3. **Decomposition of searching time.** The unit of time is second. "E2E (w/o DP)" represents the total search time excluding the DP phase.

Method	Budget (B)	Searched (B)	Δ size (B)	Information loss
GA	3×10^7	21,833,128	8,166,872	42,821,038
Vanilla ILP		28,934,805	1,065,195	1,258,394
0-1 ILP (Our)		29,831,203	168,797	11,826

Table 4. **Superiority of 0-1 ILP.** "GA": Genetic Algorithm. With up to 16 bit-width choices, the Vanilla ILP and GA that are widely adopted in model quantization methods are unable to quickly search for suitable mixed-precision settings.

hyperparameter configuration can be found in just 10 minutes. We utilize a virtual machine equipped with 14 vCPU of Intel(R) Xeon(R) Platinum 8362 CPU @ 2.80GHz and an NVIDIA 4090 RTX GPU to measure the time.

Qualitative Evaluation. In Fig. 3, we present the rendering results and the corresponding error maps. From the error maps, it is evident that our method handles chair reflections better than other methods while achieving rendering results that are comparable to ScaffoldGS.

Compared to SOTA methods. The quantitative compression results of different methods are presented in Tab. 2 and Fig. 4. Our method outperforms most others across all three datasets and achieves performance comparable to the SOTA method – HAC, demonstrating that MPQ effectively addresses the shortcomings of post-training compression methods. We also provide a comparison with NeRF compression, as shown in Tab. 5. Our method achieves better performance with a smaller file size.

Method	PNSR (dB) ↑	SSIM ↑	LPIPS ↓	Size (MB) ↓
DVGO	31.90	0.956	0.035	105.92
VQRF	31.77	0.954	0.036	1.43
ACRF	31.79	0.954	0.037	1.15
Our	32.41	0.960	0.043	1.10

Table 5. **Comparison with NeRF-based methods.** The best and 2nd best results are highlighted in red and yellow cells.

K	Budget (MB)	PSNR (dB)↑	SSIM ↑	LPIPS ↓	Searched (MB) ↓
40	30	25.13	0.7410	0.2684	29.85
30		25.15	0.7411	0.2685	29.91
50		25.14	0.7413	0.2686	29.86
40	20	25.07	0.7353	0.2752	19.83
30		25.07	0.7357	0.2757	19.85
50		25.12	0.7368	0.2743	19.92

Table 6. **Robustness Evaluation.** Mixed-precision quantization can adapt to different values of the number of blocks K, ensuring that the visual quality within a given size is not affected by K.

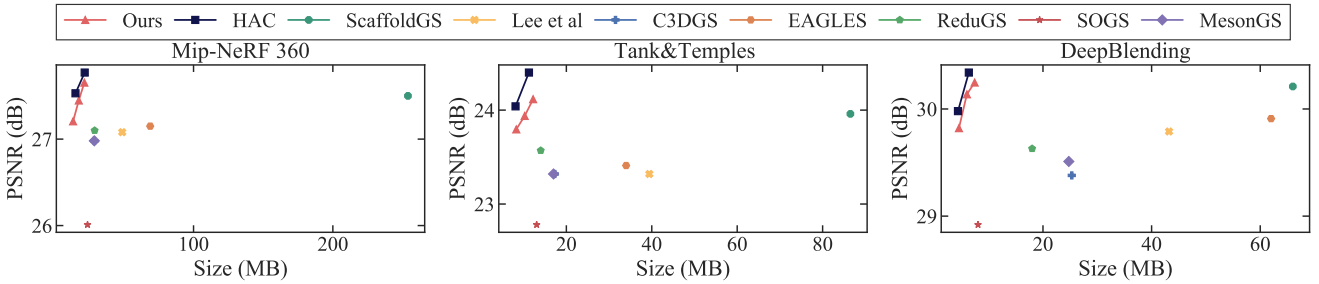
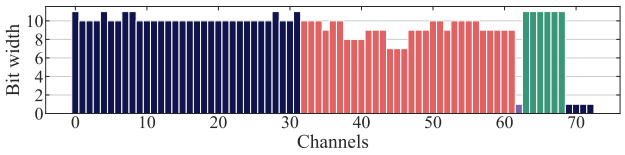
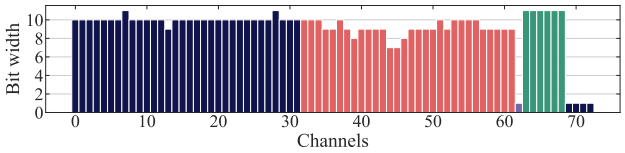


Figure 4. **Rate-distortion curves for quantitative comparisons.** Note that our goal is not to improve the marginal performance and defeat the existing compression works. Instead, we aim to design a hyperparameter parameter searching algorithm to compress the 3DGS model into the desired size while maximizing visual quality.



(a) Budget: 28 MB, Scene: *bicycle*.



(b) Budget: 12 MB, Scene: *bicycle*.

Figure 5. **Bit widths of each channel.** We use different colors to represent different kinds of attributes. From left to right are: features $\mathbf{f}$, offsets $\mathbf{O}$, opacity o, scaling $\mathbf{l}$, and rotation $\mathbf{r}$.

5.2. Ablation Study

Unless specified otherwise, all following experiments use the *bicycle* scene from the Mip-NeRF 360 dataset.

0-1 ILP Superiority in Searching Bit-widths. In solving the optimal bit-width setting for different attribute channels, we also demonstrate the superiority of the 0-1 ILP. As shown in Tab. 4, we experiment with widely-used General ILP [46] and genetic algorithms [16, 38], both of which proved inferior. The 0-1 ILP fully utilizes the size budget while minimizing information loss. General ILP involves variables ranging from 1 to 16. In contrast, 0-1 ILP's binary values offer finer control, easier integration of constraints, and more efficient solution techniques. Genetic algorithms, though suited for non-linear or black-box problems, handle constraints less efficiently, making them unsuitable for our linear programming structure.

Bit-widths. In Fig. 5, we show the bit-widths for different attribute channels, where different colors represent different attributes, from left to right in order: features $\mathbf{f}$, offsets $\mathbf{O}$, opacity o, scaling $\mathbf{l}$, and rotation $\mathbf{r}$. It can be seen that under different size budgets, the choices of bit-widths are generally the same; however, for certain channels, the 28MB budget selects a larger bit-width for specific channels.

Effectiveness of Hierarchical MPQ. In Fig. 6, we evaluate the effectiveness of each component of our proposed framework. The "baseline" refers to applying MesonGS to ScaffoldGS. We start with the baseline method and progressively incorporate inter-attribute MPQ and intra-attribute MPQ. The results show that the performance is consistently improved with the addition of each module, which proves the effectiveness of hierarchical MPQ.

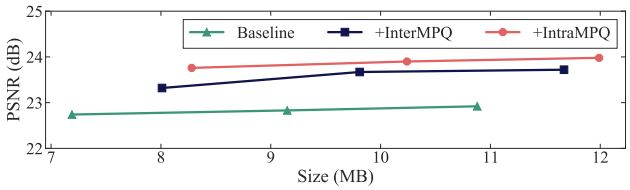


Figure 6. **Effectiveness of hierarchical MPQ.**

Robustness Evaluation. We evaluated the file size and corresponding performance of the searching algorithm under different numbers of blocks. As shown in Tab. 6, for varying numbers of blocks and different target sizes, our method consistently finds appropriate bit-width settings, ensuring that the final file size is close to the target while maintaining optimal visual quality. Regardless of the block number setting, the final file size and performance are similar, indicating that our method is robust to the number of blocks.

Why Fixing the Coordinates? We investigate the necessity of updating coordinates during training. Specifically, we rewrite the backpropagation rules for the voxelization process of the Octree. If, during the forward pass, points within a voxel are deduplicated by averaging, the gradient of that voxel will be evenly distributed to the corresponding points during backpropagation. We find that once backpropagation is enabled for the Octree, the loss fluctuates significantly, ultimately leading to worse fine-tuning results compared to keeping the coordinates fixed. Please refer to the supplementary material for details.

6. Conclusion

In this paper, we present *SizeGS*, a method for automatically selecting hyperparameters to compress 3D Gaussians to a target file size while maximizing visual qual-

ity. Key components of our approach include the selection of the base model—ScaffoldGS, the compression framework—MesonGS, the size estimator, and hierarchical mixed precision quantization (MPQ). Extensive experiments demonstrate the effectiveness of our size-aware compression methodology, achieving significant improvements in controlling the file size of 3D Gaussian compression. This work introduces a new task in 3D Gaussian compression and expands the application of MPQ within this area, paving the way for further advancements that integrate MPQ with 3D Gaussian compression.

References

[1] Muhammad Salman Ali, Maryam Qamar, Sung-Ho Bae, and Enzo Tartaglione. Trimming the fat: Efficient compression of 3d gaussian splats through pruning. In *BMVC*, 2024. 2

[2] Milena T. Bagdasarian, Paul Knoll, Florian Barthel, Anna Hilsmann, Peter Eisert, and Wieland Morgenstern. 3dgs.zip: A survey on 3d gaussian splatting compression methods, 2024. 1, 3, 6

[3] Jonathan T Barron, Ben Mildenhall, Dor Verbin, Pratul P Srinivasan, and Peter Hedman. Mip-nerf 360: Unbounded anti-aliased neural radiance fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5470–5479, 2022. 6

[4] Anpei Chen, Haofei Xu, Stefano Esposito, Siyu Tang, and Andreas Geiger. Lara: Efficient large-baseline radiance fields. In *European Conference on Computer Vision*, pages 338–355. Springer, 2025. 1

[5] Yihang Chen, Qianyi Wu, Mengyao Li, Weiyao Lin, Mehrtash Harandi, and Jianfei Cai. Fast feedforward 3d gaussian splatting compression. *arXiv preprint arXiv:2410.08017*, 2024. 2

[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. 1, 2, 3, 6

[7] Ruizhi Cheng, Nan Wu, Vu Le, Eugene Chai, Matteo Varvello, and Bo Han. Magicstream: Bandwidth-conserving immersive telepresence via semantic communication. In *Proceedings of the 22nd ACM Conference on Embedded Networked Sensor Systems*, page 365–379, New York, NY, USA, 2024. Association for Computing Machinery. 1

[8] Mallesham Dasari, Kumara Kahatapitiya, Samir R Das, Aruna Balasubramanian, and Dimitris Samaras. Swift: Adaptive video streaming with layered neural codecs. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 103–118, 2022. 1

[9] Zhen Dong, Zhewei Yao, Amir Gholami, Michael W. Mahoney, and Kurt Keutzer. HAWQ: Hessian AWare Quantization of neural networks with mixed-precision. In *The IEEE International Conference on Computer Vision (ICCV)*, 2019. 2, 5

[10] Zhen Dong, Zhewei Yao, Daiyaan Arfeen, Amir Gholami, Michael W. Mahoney, and Kurt Keutzer. HAWQ-V2: Hessian aware trace-weighted quantization of neural networks. In *Advances in neural information processing systems (NeurIPS)*, 2020. 2, 5

[11] Zhiwen Fan, Kevin Wang, Kairun Wen, Zehao Zhu, Dejia Xu, and Zhangyang Wang. LightGaussian: Unbounded 3D Gaussian Compression with 15x Reduction and 200+ FPS. In *Advances in neural information processing systems (NeurIPS)*, 2024. 1, 2, 6

[12] Guangchi Fang and Bing Wang. Mini-splatting: Representing scenes with a constrained number of gaussians. In *European Conference on Computer Vision*, 2024. 2

[13] Guangchi Fang, Qingyong Hu, Longguang Wang, and Yulan Guo. ACRF: Compressing explicit neural radiance fields via attribute compression. In *International Conference on Learning Representations(ICLR)*, 2024. 6

[14] Jean-loup Gailly and Mark Adler. Zlib general purpose compression library. *User manual for zlib version*, 1(4), 2003. 4

[15] Sharath Girish, Kamal Gupta, and Abhinav Shrivastava. Eagles: Efficient accelerated 3d gaussians with lightweight encodings. In *European Conference on Computer Vision*, 2024. 2, 6

[16] Zichao Guo, Xiangyu Zhang, Haoyuan Mu, Wen Heng, Zechun Liu, Yichen Wei, and Jian Sun. Single path one-shot neural architecture search with uniform sampling. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XVI 16*, pages 544–560. Springer, 2020. 8

[17] Peter Hedman, Julien Philip, True Price, Jan-Michael Frahm, George Drettakis, and Gabriel Brostow. Deep blending for free-viewpoint image-based rendering. *ACM Transactions on Graphics (ToG)*, 37(6):1–15, 2018. 6

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

[19] Arno Knapitsch, Jaesik Park, Qian-Yi Zhou, and Vladlen Koltun. Tanks and temples: Benchmarking large-scale scene reconstruction. *ACM Transactions on Graphics (ToG)*, 36 (4):1–13, 2017. 6

[20] Joo Chan Lee, Daniel Rho, Xiangyu Sun, Jong Hwan Ko, and Eunbyung Park. Compact 3d gaussian representation for radiance field. *CVPR*, 2024. 1, 2, 3, 6

[21] Ke Li, Reinhard Bacher, Susanne Schmidt, Wim Leemans, and Frank Steinicke. Reality fusion: Robust real-time immersive mobile robot teleoperation with volumetric visual data fusion. *arXiv preprint arXiv:2408.01225*, 2024. 1

[22] Lingzhi Li, Zhen Shen, Zhongshu Wang, Li Shen, and Liefeng Bo. Compressing volumetric radiance fields to 1 mb. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4222–4231, 2023. 6

[23] Sicheng Li, Hao Li, Yiyi Liao, and Lu Yu. Nerfcodec: Neural feature compression meets neural radiance fields for memory-efficient scene representation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 21274–21283, 2024. 1

[24] Junhua Liu, Boxiang Zhu, Fangxin Wang, Yili Jin, Wenyi Zhang, Zihan Xu, and Shuguang Cui. Cav3: Cache-assisted

viewport adaptive volumetric video streaming. In *2023 IEEE Conference Virtual Reality and 3D User Interfaces (VR)*, pages 173–183. IEEE, 2023. 1

[25] Weihang Liu, Xue Xian Zheng, Jingyi Yu, and Xin Lou. Content-aware radiance fields: Aligning model complexity with scene intricacy through learned bitwidth quantization. In *European Conference on Computer Vision*, pages 239–256. Springer, 2024. 3

[26] Xiangrui Liu, Xinju Wu, Pingping Zhang, Shiqi Wang, Zhu Li, and Sam Kwong. Compgs: Efficient 3d scene representation via compressed gaussian splatting. In *Proceedings of the 32nd ACM International Conference on Multimedia*, page 2936–2944, New York, NY, USA, 2024. Association for Computing Machinery. 1

[27] Yu Liu, Bo Han, Feng Qian, Arvind Narayanan, and Zhi-Li Zhang. Vues: practical mobile volumetric video streaming through multiview transcoding. In *ACM MobiCom '22: The 28th Annual International Conference on Mobile Computing and Networking, Sydney, NSW, Australia, October 17 - 21, 2022*, pages 514–527. ACM, 2022. 1

[28] 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, 3, 6

[29] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. *Communications of the ACM*, 65(1):99–106, 2021. 2, 6

[30] Wieland Morgenstern, Florian Barthel, Anna Hilsmann, and Peter Eisert. Compact 3d scene representation via self-organizing gaussian grids. In *European Conference on Computer Vision*. Springer, 2024. 1, 2, 6

[31] KL Navaneet, Kossar Pourahmadi Meibodi, Soroush Abbasi Koohpayegani, and Hamed Pirsiavash. Compgs: Smaller and faster gaussian splatting with vector quantization. *ECCV*, 2024. 6

[32] Simon Niedermayr, Josef Stumpfegger, and Rüdiger Westermann. Compressed 3d gaussian splatting for accelerated novel view synthesis. In *CVPR*, 2024. 1, 6

[33] Panagiotis Papantonakis, Georgios Kopanas, Bernhard Kerbl, Alexandre Lanvin, and George Drettakis. Reducing the memory footprint of 3d gaussian splatting. *Proceedings of the ACM on Computer Graphics and Interactive Techniques*, 7(1), 2024. 1, 2, 6

[34] Kerui Ren, Lihan Jiang, Tao Lu, Mulin Yu, Linning Xu, Zhangkai Ni, and Bo Dai. Octree-gs: Towards consistent real-time rendering with lod-structured 3d gaussians. *arXiv preprint arXiv:2403.17898*, 2024. 2

[35] J.S. Roy and S.A. Mitchell. PuLP is an LP modeler written in Python. 2020. 5

[36] Cheng Sun, Min Sun, and Hwann-Tzong Chen. Direct voxel grid optimization: Super-fast convergence for radiance fields reconstruction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5459–5469, 2022. 6

[37] Chen Tang, Kai Ouyang, Zhi Wang, Yifei Zhu, Yaowei Wang, Wen Ji, and Wenwu Zhu. Mixed-precision neural network quantization via learned layer-wise importance. In *European Conference on Computer Vision*, 2022. 2

[38] Chen Tang, Yuan Meng, Jiacheng Jiang, Shuzhao Xie, Rongwei Lu, Xinzhu Ma, Zhi Wang, and Wenwu Zhu. Retraining-free model quantization via one-shot weight-coupling learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15855–15865, 2024. 8

[39] 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. 1, 2

[40] Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. HAQ: Hardware-aware automated quantization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019. 2

[41] Penghao Wang, Zhirui Zhang, Liao Wang, Kaixin Yao, Siyuan Xie, Jingyi Yu, Minye Wu, and Lan Xu. V^3: Viewing volumetric videos on mobiles via streamable 2d dynamic gaussians. *CoRR*, abs/2409.13648, 2024. 1

[42] Yufei Wang, Zhihao Li, Lanqing Guo, Wenhan Yang, Alex C Kot, and Bihan Wen. Contextgs: Compact 3d gaussian splatting with anchor level context model. In *Advances in neural information processing systems (NeurIPS)*, 2024. 1, 2, 3

[43] Minye Wu and Tinne Tuytelaars. Implicit gaussian splatting with efficient multi-level tri-plane representation, 2024. 1, 2

[44] 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. 1, 2, 6

[45] Runyi Yang, Zhenxin Zhu, Zhou Jiang, Baijun Ye, Xiaoxue Chen, Yifei Zhang, Yuantao Chen, Jian Zhao, and Hao Zhao. Spectrally pruned gaussian fields with neural compensation, 2024. 1

[46] Zhewei Yao, Zhen Dong, Zhangcheng Zheng, Amir Gholami, Jiali Yu, Eric Tan, Leyuan Wang, Qijing Huang, Yida Wang, Michael Mahoney, et al. Hawq-v3: Dyadic neural network quantization. In *International Conference on Machine Learning*, pages 11875–11886. PMLR, 2021. 2, 5, 8

[47] Wang Yifan, Felice Serena, Shihao Wu, Cengiz Öztireli, and Olga Sorkine-Hornung. Differentiable surface splatting for point-based geometry processing. *ACM Transactions on Graphics (TOG)*, 38(6):1–14, 2019. 3

[48] Jiakai Zhang, Liao Wang, Xinhang Liu, Fuqiang Zhao, Minzhang Li, Haizhao Dai, Boyuan Zhang, Wei Yang, Lan Xu, and Jingyi Yu. Neuvv: Neural volumetric videos with immersive rendering and editing. *CoRR*, abs/2202.06088, 2022. 1

[49] Jacob Ziv and Abraham Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on information theory*, 23(3):337–343, 1977. 4

[50] Jacob Ziv and Abraham Lempel. Compression of individual sequences via variable-rate coding. *IEEE transactions on Information Theory*, 24(5):530–536, 1978. 4

[51] Chen Ziwen, Hao Tan, Kai Zhang, Sai Bi, Fujun Luan, Yicong Hong, Li Fuxin, and Zexiang Xu. Long-lrm: Long-sequence large reconstruction model for wide-coverage gaussian splats. *arXiv preprint arXiv:2410.12781*, 2024. 1

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

[53] Matthias Zwicker, Hanspeter Pfister, Jeroen Van Baar, and Markus Gross. Surface splatting. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 371–378, 2001. 3

SizeGS: Size-aware Compression of 3D Gaussians with Hierarchical Mixed Precision Quantization

Supplementary Material

Algorithm 2: Hyperparameter Searching.

Data: Size Budget and a pre-trained GS file.
Result: Optimal hyperparameters set Φ^*.
1 all_qbits[] = Carefully selected values of $|\mathbf{q}|$;
2 $\Omega^* = +\infty$;
3 **for** $|\mathbf{q}|$ *in* all_qbits[] **do**
  `/* Initialize hyperparameter set */`
4 $\Phi = \{\}$;
5 Solve out the τ: $\Phi = \{\tau\}$;
6 Determine the octree depth d and number of blocks K based on τN: $\Phi = \{\tau, d, K\}$;
7 Solve the 0-1 ILP to obtain the $\mathbf{q}$ and overall information loss Ω: $\Phi = \{\tau, d, K, \mathbf{q}, \mathbf{n}\}$;
8 **if** $\Omega < \Omega^*$ **then**
9 $\Omega^* = \Omega$;
10 $\Phi^* = \Phi$;
`/* Calibrate the size estimator */`
11 Use Φ^* to finish encoding and obtain residual Δ;
12 Calibrate size estimator: $\mathcal{S}^* = \mathcal{S} + \Delta$;
13 Solve out $\mathbf{q}^*$ and $\mathbf{n}^*$ based on the calibrated estimator $\mathcal{S}^*$;
14 **return** $\Phi^* = \{\tau, d, K, \mathbf{q}^*, \mathbf{n}^*\}$;

A. More Details of the Search Algorithm

In line 6 of Algo. 2, we determine the number of blocks, K, based on τN. For all experiments, we fix K to 60. The rationale for this choice is that the 0-1 Integer Linear Programming (ILP) formulation adjusts the bit-width settings to satisfy the specified K. The 0-1 ILP formulation, proposed in Sec. 4.2, is defined as:

$$\begin{aligned} &\text{Objective: } \Omega \odot \mathbf{Q}, \\ &\text{Subject to: } \mathcal{S}(\mathbf{Q}) \leq \text{ Model Size Limit}, \\ &\forall i \in \{0, ..., H-1\}, \sum_{j=0}^{B} \mathbf{Q}_{i,j} = 1. \end{aligned} \tag{10}$$

To find the optimal bit-width settings, we precompute the information loss matrix $\Omega \in \mathbb{R}^{H \times B}$, where

$$\Omega(i, j) = \|\hat{\mathcal{A}}_i^j - \mathcal{A}_i\|^2. \tag{11}$$

Here, $\hat{\mathcal{A}}_i^j$ is obtained by applying a round of quantization and dequantization to $\mathcal{A}_i$, using the block quantization scheme.

Determining both block length and bit-width settings introduces a "chicken-or-the-egg" dilemma: one must be fixed as a basis before the other can be solved. To address this, we first solve for block length settings under a uniform bit-width assumption. Then, based on the resolved block lengths, we determine the mixed precision bit-width settings. The bit-width settings automatically adapt to the value of K as they are derived from block lengths.

K	Before Fine-tuning			After Fine-tuning		
	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓
60	23.59	0.666	0.303	25.16	0.743	0.266
20	23.52	0.661	0.305	25.15	0.742	0.266
15	23.56	0.663	0.304	25.10	0.741	0.266
10	23.57	0.663	0.303	25.09	0.738	0.269
5	23.59	0.666	0.302	25.07	0.739	0.269
3	23.50	0.659	0.304	25.01	0.737	0.271
1	23.55	0.664	0.302	25.00	0.734	0.273

Table 7. **Robustness of K.** Experiments are conducted on the *bicycle* scene. We set the size budget as 28 MB.

B. Robustness of K

To evaluate the robustness of the proposed search algorithm with respect to K, we conducted a comprehensive experimental analysis. As shown in Tab. 7, we set K to different values and finetune the compressed model for 6000 steps to collect the quality metrics. Remarkably, even when K was set to 1, the search algorithm produced satisfactory results, especially without fine-tuning. This is because the bit-width settings dynamically adapt to K. Besides, we draw the biwidth settings of different K in Fig. 7. We observe that the features $\mathbf{f}$ and the scaling $\mathbf{l}$ gain more bit-widths when K increases, while offsets $\mathbf{O}$ are given less bit-widths.

C. Experiment Settings of HAC

This section describes the experimental settings used to measure the hyperparameter search latency for HAC.

As shown in the Fig. 8, the size of the compressed file is a monotonically decreasing function of the λ. Consequently, we employ binary search on the hyperparameter λ to adjust the compressed file size in HAC. Other parameters follow the official implementation.

To perform the binary search, we first determine the *left* and *right* bounds for λ. Based on prior experience with HAC, we set the *left* bound to 1×10^{-6} and the *right* bound to 4×10^{-3} across all scenes. The search terminates when the absolute difference between the compressed file size and the size budget is less than 10% of the size budget. All measurements were conducted on a machine with an Intel(R) Xeon(R) Gold 6230 CPU and an NVIDIA RTX 4090 GPU.

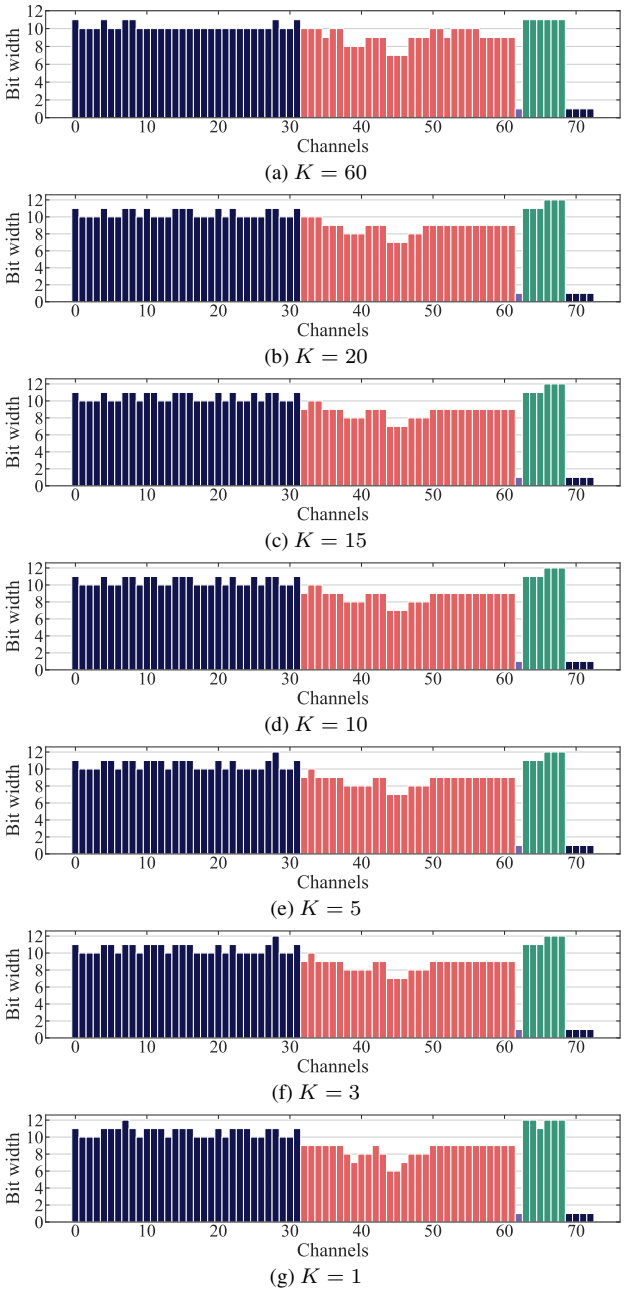


Figure 7. **Bit-width setting under different values of K.** We use different colors to represent different attributes. From left to right are: features **f**, offsets **O**, opacity o, scaling **l**, and rotation **r**.

To summarize, the hyperparameter search latency for HAC comprises two parts: **1) Initial Bound Search Time.** This is the time required to determine the *left* and *right* bounds. This is set as a constant of 2000 seconds, based on the typical time consumption of compression. **2) Binary Search Time.** This is the time required to finish the binary search.

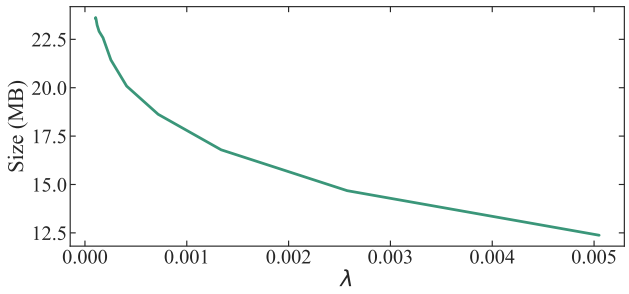


Figure 8. **Correlation between the size and λ.** The file size is a monotonically decreasing function of λ, so it is reasonable to use the binary search. Experiments are conducted on the *bicycle* scene.

D. Additional Experiments

Why Fixing the Coordinate? Fig. 9 illustrates the training loss curves under four scenarios: **(a)** the baseline (no anchor coordinate updates), **(b)** updating anchor coordinates during iterations 0 to 1000, **(c)** updating anchor coordinates during iterations 1000 to 2000, and **(d)** a comparison of (a) and (b). The results demonstrate that updating anchor coordinates during training causes a rapid increase in loss and finally leads to worse convergence compared to baseline training.

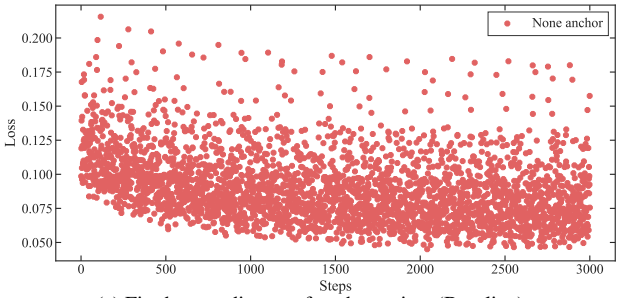
Per-scene Results. Tab. 8, Tab. 9, Tab. 10, and Tab. 11 present the per-scene performance of our method across four datasets: MipNeRF 360 dataset, Tank&Temples dataset, Deep Blending dataset, and Synthetic-NeRF dataset. Tab. 12, Tab. 13, Tab. 14, and Tab. 15 detail the storage composition of our method on these datasets.

E. Granularity of Our MPQ Scheme

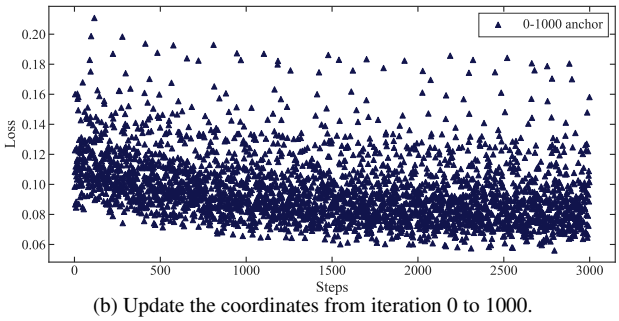
Fig. 10 compares HAC [6] and our method. The mixed-precision quantization (MPQ) scheme used in our approach achieves a finer granularity.

F. Notations

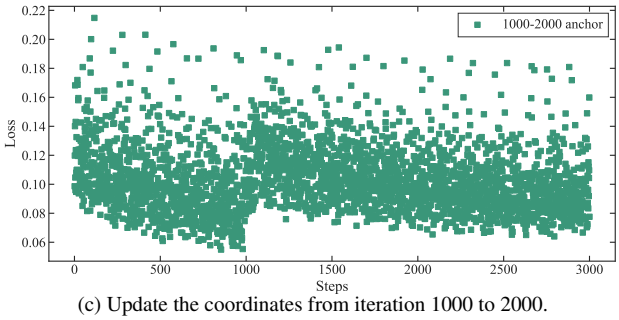
Tab. 16 summarizes all notations used in this paper along with their definitions. The notations are divided into 6 groups, separated by lines: 1) 3DGS-related terms; 2) ScaffoldGS-related terms; 3) Size estimator; 4) Hyperparameters; 5) 0-1 integer linear programming terms; 6) Dynamic programming terms.



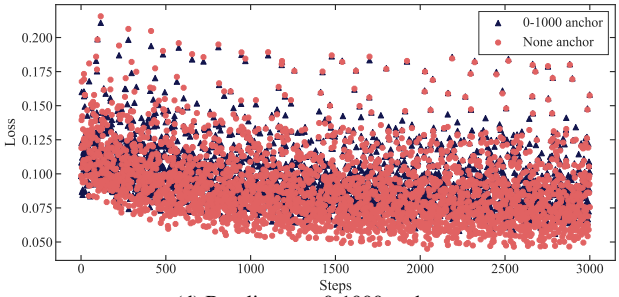
(a) Fix the coordinates of anchor points (Baseline).



(b) Update the coordinates from iteration 0 to 1000.



(c) Update the coordinates from iteration 1000 to 2000.



(d) Baselines vs. 0-1000 update.

Figure 9. **Fixing the coordinates of anchor points produces lower loss.**

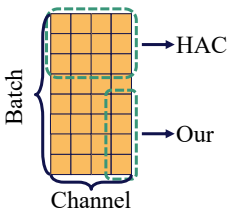


Figure 10. **Granularity of MPQ: HAC vs. Our.**

Scene	PSNR	SSIM	LPIPS	Size (MB)
bicycle	25.02	0.728	0.286	20.71
bonsai	32.01	0.939	0.197	10.13
counter	28.77	0.897	0.221	6.97
kitchen	30.03	0.913	0.153	7.53
garden	26.46	0.813	0.186	17.90
room	31.32	0.917	0.217	6.63
stump	26.79	0.766	0.263	17.84
flowers	21.30	0.577	0.379	16.62
treehill	23.16	0.644	0.353	17.51
Average	**27.21**	**0.799**	**0.251**	**13.54**
bicycle	25.16	0.745	0.261	28.56
bonsai	32.47	0.944	0.189	12.48
counter	29.21	0.908	0.205	9.22
kitchen	30.44	0.917	0.145	9.58
garden	26.91	0.827	0.167	23.59
room	31.42	0.920	0.212	7.22
stump	26.86	0.766	0.262	23.28
flowers	21.35	0.579	0.375	22.08
treehill	23.20	0.644	0.351	22.60
Average	**27.45**	**0.806**	**0.241**	**17.62**
bicycle	25.14	0.741	0.263	35.32
bonsai	32.87	0.948	0.182	16.62
counter	29.45	0.913	0.195	11.39
kitchen	30.87	0.923	0.136	11.43
garden	27.24	0.842	0.150	29.72
room	32.02	0.928	0.194	11.75
stump	26.77	0.767	0.262	29.19
flowers	21.33	0.577	0.377	25.73
treehill	23.20	0.644	0.352	23.49
Average	**27.65**	**0.809**	**0.235**	**21.63**

Table 8. Per-scene results of the MipNeRF-360 dataset.

Scene	PSNR	SSIM	LPIPS	Size (MB)
truck	25.61	0.873	0.159	10.06
train	21.97	0.794	0.244	6.49
Average	**23.79**	**0.833**	**0.201**	**8.27**
truck	25.79	0.878	0.152	12.47
train	22.08	0.792	0.243	8.01
Average	**23.93**	**0.835**	**0.197**	**10.24**
truck	25.87	0.879	0.148	14.59
train	22.14	0.790	0.251	9.37
Average	**24.01**	**0.835**	**0.199**	**11.98**

Table 9. Per-scene results of the Tank&Temples dataset.

Scene	PSNR	SSIM	LPIPS	Size (MB)
drjohnson	29.10	0.888	0.295	4.82
playroom	30.54	0.904	0.278	4.24
Average	**29.82**	**0.896**	**0.286**	**4.53**
drjohnson	29.42	0.894	0.282	6.39
playroom	30.85	0.908	0.268	5.59
Average	**30.13**	**0.901**	**0.275**	**5.99**
drjohnson	29.54	0.899	0.273	7.9
playroom	30.95	0.907	0.269	6.93
Average	**30.24**	**0.903**	**0.271**	**7.42**

Table 10. Per-scene results of the Deep Blending dataset.

Scene	PSNR	SSIM	LPIPS	Size (MB)
chair	33.53	0.979	0.020	1.18
drumps	25.81	0.945	0.049	1.73
ficus	34.29	0.982	0.016	1.09
hotdogs	36.67	0.979	0.030	0.73
lego	33.93	0.972	0.030	1.39
materials	29.94	0.956	0.046	1.65
mic	35.44	0.989	0.010	0.89
ship	30.90	0.896	0.127	1.78
Average	**32.56**	**0.962**	**0.041**	**1.30**
chair	34.10	0.981	0.017	1.58
drumps	25.96	0.947	0.047	2.32
ficus	34.64	0.983	0.015	1.45
hotdogs	37.03	0.981	0.027	0.97
lego	34.20	0.975	0.026	1.80
materials	30.21	0.958	0.044	2.13
mic	35.97	0.990	0.009	1.18
ship	31.10	0.898	0.123	2.39
Average	**32.90**	**0.964**	**0.038**	**1.73**
chair	34.71	0.983	0.014	1.92
drumps	26.05	0.947	0.047	2.82
ficus	34.86	0.984	0.014	1.78
hotdogs	37.49	0.982	0.025	1.17
lego	35.08	0.978	0.021	2.24
materials	30.41	0.959	0.042	2.69
mic	36.39	0.991	0.008	1.42
ship	31.29	0.900	0.118	2.89
Average	**33.28**	**0.965**	**0.036**	**2.12**

Table 11. Per-scene results of the Synthetic-NeRF dataset.

Scene	Octree	Metadata+Attributes	MLP
bicycle	1.360	27.19	0.048
bonsai	0.717	12.65	0.048
counter	0.491	8.69	0.048
garden	1.190	22.66	0.048
kitchen	0.331	9.21	0.048
room	0.431	9.01	0.048
stump	1.010	22.22	0.048
treehill	0.917	21.65	0.046
flowers	0.941	20.89	0.046

Table 12. Composition of storage sizes (MB) for different parts in the Mip-NeRF360 dataset.

Scene	Octree	Metadata+Attributes	MLP
train	0.243	6.21	0.046
truck	0.402	9.62	0.046

Table 13. Composition of storage sizes (MB) for different parts in the Tank&Temples dataset.

Scene	Octree	Metadata+Attributes	MLP
drjohnson	0.269	6.08	0.048
playroom	0.245	5.31	0.048

Table 14. Composition of storage sizes (MB) for different parts in the Deep Blending dataset.

Scene	Octree	Metadata+Attributes	MLP
chair	0.073	1.47	0.046
drums	0.098	2.19	0.046
ficus	0.151	1.36	0.046
hotdog	0.101	0.89	0.046
lego	0.187	1.67	0.046
materials	0.213	1.99	0.046
mic	0.118	1.09	0.046
ship	0.248	2.25	0.046

Table 15. Composition of storage sizes (MB) for different parts in the Synthetic-NeRF dataset.

Notation	Definition
Σ	Covariance matrix of a Gaussian distribution
$G(x)$	Gaussian function
$\mathbf{S}$	Scaling matrix
s	Scaling vector
$\mathbf{R}$	Rotation matrix
$\mathbf{q}$	Rotation quaternion
$\mathbf{W}$	Viewing transform
$\mathbf{J}$	Jacobian matrix
$\mathbf{\Sigma}'$	Covariance matrix of in camera space
$\mathbf{SH}$	Spherical harmonics co-efficient
o	Opacity
μ	3D center of a Gaussian function
$\mathbf{x}$	Location of anchor point in ScaffoldGS
$\mathbf{f}$	A context feature of the anchor point
$\mathbf{l}$	Scaling factor of ScaffoldGS
$\mathbf{O}$	Learnable offsets of ScaffoldGS
$\mathcal{A}$	Attributes of ScaffoldGS, including features $\mathbf{f}$, scaling factor $\mathbf{l}$, and learnable offsets $\mathbf{O}$
$\mathcal{S}$	Estimated file size
$\mathcal{S}^*$	Calibrated file size
$\mathcal{S}(\cdot)$	Function of size estimator
$\mathcal{C}$	Constant
Δ	The residual used to calibrate the size estimator
τ	Reserving percentage
N	Number of anchor points after applying pruning and voxelization
H	Number of channels of the attributes
d	Depth of octree
K	Number of blocks
$\mathbf{q}$	Bit-width settings in the form of vector
$\lvert\mathbf{q}\rvert$	Sum of the vector of bit-width settings
$\mathbf{n}$	Block-length settings
n_i	The start index of the block b_i
B	Quantization options, usually set as 16
$\mathbf{Q}$	One-hot bit precision matrix
$\mathbf{B}$	Vector, $[0, 1, ..., B]$, to transform $\mathbf{Q}$ into bit vector $\mathbf{q}$
Ω	Information loss (a matrix or a value)
b_i	A block of attribute
$\hat{b_i}$	The vector that de-quantized from the quantized b_i
$F(k, l)$	The minimal total information loss when slicing the l elements into k blocks
U	Step unit of dynamic programming
$\mathcal{L}(n_i, n_{i+1})$	Information loss in quantizing block b_i

Table 16. **Notation table.**