arXiv:2512.00850v2 [cs.CV] 27 Mar 2026

Smol-GS: Compact Representations for Abstract 3D Gaussian Splatting

Haishan Wang, Mohammad Hassan Vali, and Arno Solin

ELLIS Institute Finland and Aalto University, Espoo, Finland
{haishan.wang, mohammad.vali, arno.solin}@aalto.fi

Abstract. We present Smol-GS, a novel method for learning compact representations for 3D Gaussian Splatting (3DGS). Our approach learns highly efficient splat-wise features to model 3D space which capture abstracted cues, including color, opacity, transformation, and material properties. We propose octree-derived positional encoding, which explicitly models spatial locality and enhances representation efficiency. We further apply entropy-based compression to exploit feature redundancy, and compress splat coordinates using a recursive voxel hierarchy. This design enables orders-of-magnitude storage reduction while preserving representation flexibility. Smol-GS achieves state-of-the-art compression performance on standard benchmarks with high-level rendering quality.

Keywords: 3D view synthesis · Gaussian splatting · Compression

Project page: https://aaltoml.github.io/Smol-GS

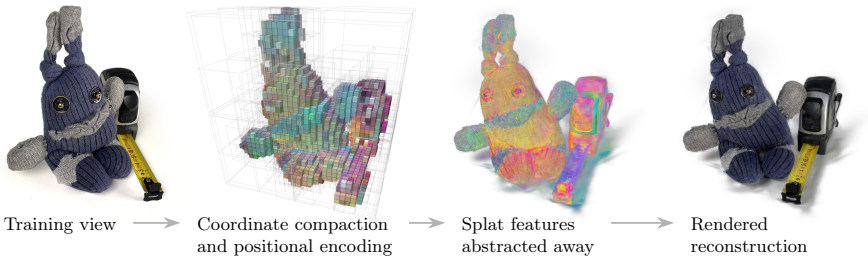

Fig. 1: Smol-GS learns a compact representation of the 3D scene that *(i)* stores coordinates in an efficient octree structure, *(ii)* learns local positional encodings, and *(iii)* abstracts away view-dependent splat features such as color, shape, and material cues. On Mip-NeRF 360, this gives state-of-the-art results in size/PSNR: 4.87 MB/27.61 dB.

1 Introduction

3D Gaussian Splatting (3DGS, [18]) is a paradigm for novel view synthesis that models a scene as a set of rasterized Gaussian splats. It achieves real-time speed

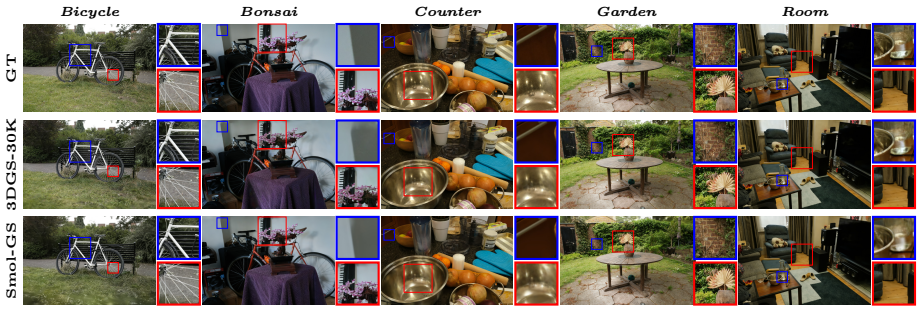

Fig. 2: Qualitative results on the Mip-NeRF 360 data set. Smol-GS provides high-fidelity reconstructions competitive with vanilla 3DGS-30K. Smol-GS preserves sharp shadow edges and material effects (specular highlights, glass, flat walls) while using an orders-of-magnitude smaller model. We include quantitative summary results in Table 1.

and high rendering quality through rasterization techniques [22]. However, 3DGS suffers from memory inefficiency; it often requires several gigabytes of storage with millions of splats to represent complex scenes. This issue related to the large memory footprint has been tackled extensively in recent works [3,4,7,8,11, 23,25,28,29,32–34,36,38,45,46,51,55]. These methods improve compactness to varying degrees, yet the coordinate data and high-dimensional appearance cues remain hard to store efficiently at scale.

Two directions dominate current 3DGS compression: *(i)* Signal-processing approaches quantize splat attributes and deliver considerable storage savings [7,23,32–34,45]. *(ii)* Learning-based approaches compress by sharing structure across splats, with anchor–offset hierarchies from Scaffold-GS [27] forming the backbone of the most competitive results [3,4,25,46]. However, signal-processing pipelines do not fully exploit local spatial correlations and repeated patterns among splats, and anchor–offset designs can also introduce transparent or invisible offsets and extra heuristics. These approaches often sidestep the coordinate compression because reconstruction is sensitive to spatial errors. We revisit this premise in our work (see Fig. 1) that hinges on the realization that keeping the spatial structure explicit (but efficiently packed) and abstracting the splat-wise visual features away can result in highly efficient compression.

In this paper, inspired by neural Gaussian generation in anchor-based systems [27], point-cloud geometry compression [39], and hash-grid assisted feature compression [3], we propose Smol-GS by introducing positional encodings to splats. The method learns a compact representation with splat-wise abstract features and a few tiny MLPs, while compressing coordinates and features. Coordinates are stored with an occupancy-octree encoding and entropy coding; features (color, opacity, transformation, and material cues) and scaling controllers are quantized and entropy-coded with distributions predicted from hashed spatial descriptors. Our design removes anchor–offset overhead, reduces redundancy further, and enables splat-level operations such as 3D editing. Qualitative (Fig. 2)

and quantitative (Fig. 7 and Table 1) comparisons show that Smol-GS achieves higher rendering quality while pushing model size down to the smallest among strong baselines.

Contributions Our contributions can be summarized as:

- We introduce Smol-GS, a novel neural splat model for 3DGS. It provides a simple yet effective strategy for representing 3D scenes with highly compact, splat-wise abstract features and lightweight MLPs. Compared to anchor-based methods, our design reduces structural overhead and enables direct splat-level manipulation.
- We propose a novel positional encoding $\boldsymbol{f}_{oct}$ derived from the octree hierarchy that enhances neural splat representation efficiency.
- We demonstrate that Smol-GS substantially reduces the storage requirements for 3DGS while maintaining high rendering quality, as validated by both quantitative and qualitative evaluations compared to the state-of-the-art approaches.

2 Related Work

3D Gaussian Splatting The 3DGS paradigm represents a scene as a collection of Gaussian splats initialized via structure-from-motion (SfM, [40]), based on computer graphics research [47, 58]. Each splat is associated with a location, covariance (via scale and rotation), opacity, color, and view-dependent material properties (spherical harmonics, SH). The explicit primitive and Gaussian rasterization allow real-time rendering, while the continuous representation yields high image quality. These advantages make 3DGS a promising framework for broad applications, such as SLAM [13, 14, 17, 50], 3D editing [5, 41, 49, 56], and VR [16, 42, 54, 57].

Compaction vs. Attribute Compression Resolving the challenging memory footprint of storing the splat-wise parameters has mainly focused on two lines of research. *Compaction* reduces the number of splats via pruning and regularization, often guided by opacity, scale, importance estimation [8, 28, 36, 38, 55]. This can save memory but risks losing thin structures and requires heuristics. *Attribute compression* keeps the splat set but encodes attributes more efficiently, balancing rate and fidelity. For example, vector-quantization [10, 43, 44] based methods utilize clustering algorithms [24, 26] to compress per-splat attributes (*e.g.*, color/SH, opacity, scale/rotation) using codebooks [7, 23, 32–34]. However, traditional tools cannot eliminate the information redundancy hidden in continuity and correlation in 3D space.

Learned Compact Representations Anchor–offset hierarchies introduced by Scaffold-GS [27] condense local neighborhoods into shared anchor embeddings and reconstruct splat features with small MLPs. Follow-ups add stronger coding: HAC++ [4] and HEMGS [25] use entropy models, and ContextGS [46] leverages autoregressive anchors to reduce storage further. These methods are

effective because they share structure across splats and model statistics. However, the anchors always introduce a certain ratio of transparent offsets, and the recovery of splat coordinates requires the composition of three factors: anchor location, relative coordinates, and scaling control. These design drawbacks, stemming from heuristic complications, limit the compression ratio and flexibility for downstream applications. Additionally, the methods typically avoid compressing coordinates due to quality sensitivity, which limits the efficiency of compression.

Coordinate Compression and Geometry Coding Point-cloud geometry compression shows that hierarchical occupancy enables compact, often lossless storage of 3D locations via octrees [6, 9, 37, 39]. This perspective suggests that 3DGS coordinates—despite their importance for fidelity—can also be stored efficiently using an occupancy-octree with entropy coding [15], leaving appearance and view-dependent effects to be learned by low-dimensional features.

Smol-GS is different from previous anchor–offset designs by *(i)* adopting an individual neural splat formulation that eliminates anchor-offsets bookkeeping; *(ii)* encoding splat coordinates with an occupancy-octree combined with entropy coding; and *(iii)* introducing an octree-derived positional encoding to enhance splat representation efficiency. Smol-GS aims to enhance the data exploitation capability of learning-based methods.

3 Methods

Smol-GS consists of five parts that work together (see Fig. 3 for an overview): *(i) Coordinate compression* (Sec. 3.1): Splat coordinates are stored with an occupancy-octree and entropy coding, leveraging spatial sparsity while avoiding anchor–offset overhead. *(ii) Positional encoding* (Sec. 3.2): Each splat inherits a unique identifier derived from the corresponding finest-level octree cell. This octree-derived positional encoding improves representation compactness and structural consistency. *(iii) Tiny decoders* (Sec. 3.3): Small MLPs reconstruct view-dependent color and transformation from abstract features, viewing direction and positional embeddings, keeping rendering fast. *(iv) Feature compression* (Sec. 3.4): Low-dimensional per-splat features capture color, opacity, transformation, and material cues. These features are quantized using learned step sizes and encoded arithmetically. *(v) Adaptive density control and training* (Secs. 3.5 and 3.6): Gradient-based densification and pruning, together with stage-wise optimization, balance reconstruction fidelity and model size.

3.1 Coordinate Compression: Occupancy-octree Encoding

Many 3DGS compression methods avoid coordinate quantization due to the sensitivity of reconstruction to geometry. Our example in Fig. 4 shows that this precision requirement could be relaxed: 16-bit per-axis quantization yields an imperceptible quality drop. Therefore, inspired by point cloud geometry compression methods [6, 9, 37, 39], we use an octree structure to capture the occupancy of splats in 3D space such that the tree is built recursively (see Fig. 5).

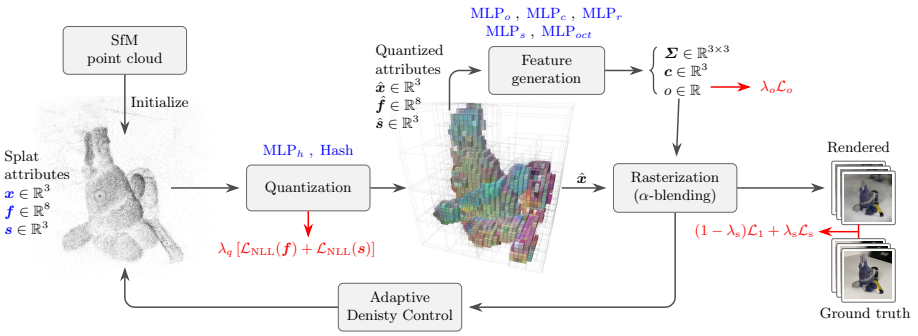

Fig. 3: Method overview of Smol-GS: We train (trained parameters in blue) neural splats with tiny MLP-decoders for view-dependent rendering (Sec. 3.3). The coordinates are compressed with occupancy-octree coordinate coding (Sec. 3.1). We also learn the quantization of splat features $\boldsymbol{f}$ and scaling controllers $\boldsymbol{s}$ with NLL rate terms (Sec. 3.4), and employ adaptive density control with stage-wise training (loss-terms in red) to balance fidelity and size (Secs. 3.5 and 3.6).

We begin from the axis-aligned bounding box covering all splats and, at recursion $r = 1$, subdivide it into 8 equal sub-boxes. At each subsequent recursion, only *non-empty* boxes (those containing at least one splat) are subdivided again into 8 equal children. The boundary of the coarsest-level box is defined by the minimum and maximum of all splat coordinates: $\boldsymbol{x}_{\min}$ and $\boldsymbol{x}_{\max}$. A box at depth $r \in \mathbb{N}$ has size $b_r = (\boldsymbol{x}_{\max} - \boldsymbol{x}_{\min})/2^r$. Each subdivision emits an 8-bit *occupancy byte* indicating which children are non-empty (1) or empty (0), with child order given by the Morton space-filling curve [30]. After R recursions, the resulting occupancy-octree explicitly represents the spatial layout of all splats such that the centre of each finest-level box at depth R provides the quantized coordinate, with at most one splat in each finest-level box. This corresponance quantizes the coordinate of the splat $\boldsymbol{x}$ to be

$$\hat{\boldsymbol{x}} = b_R \cdot \mathrm{round}((\boldsymbol{x} - \boldsymbol{x}_{\min})/b_R) + \boldsymbol{x}_{\min}. \tag{1}$$

Coordinate Storage then reduces to recording the sequence of occupancy bytes for all non-empty internal tree nodes, plus the boundaries of the coarsest bounding box and recursion depth R. Because these bytes are highly sparse (most nodes have few occupied children, see Sec. B.4), we apply Huffman coding [15] for an entropy-based compression, yielding compact *lossless-within-grid* storage of coordinates relative to the chosen quantization grid.

3.2 Positional Encoding: Index from Space Division

During the spatial partitioning described in Sec. 3.1, each splat is assigned to a unique finest-level bounding box. At each recursion, every box is subdivided into eight sub-boxes. Therefore, there are at most 8^R possible finest-level boxes

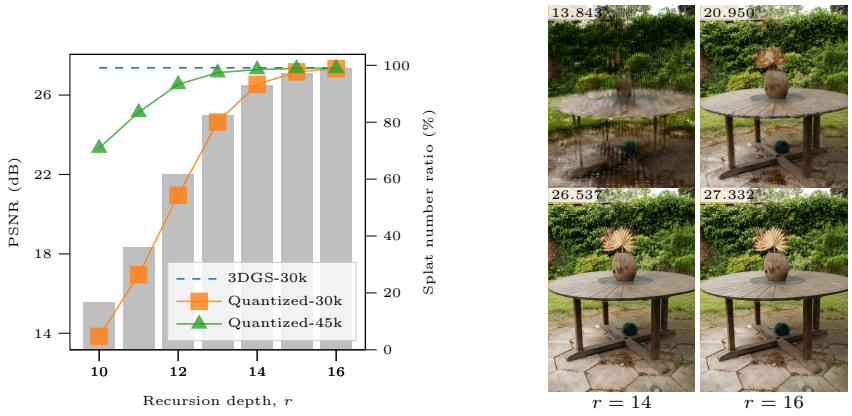

Fig. 4: Quantization recursion depth r vs. PSNR. We experiment with quantizing the coordinates of a trained 3DGS-30k model to be the 'Quantized-30k' models. These models are finetuned with quantized coordinates fixed for an additional 15k iterations to be 'Quantized-45k'. Left: Quantitative metrics (PSNR) and splat number ratio after quantization vs. quantization recursion on the *Garden* scene. Right: Qualitative results of Quantized-30k. Higher recursions yield better quality but keep more splats.

available for assignments. Then each finest-level box could be encoded by a $3R$-bit sequence, which serves as the **positional embedding** $\boldsymbol{f}_{oct} \in \{0,1\}^{3R}$ of the splat contained within that box. Mathematically, $\boldsymbol{f}_{oct}$ represents the binary data of the Morton code (Z-order index) [30] of the associated finest-level box.

Suppose a splat is located in box B at recursion depth r. Then, the $(3r-2)^{\text{th}}$ through $3r^{\text{th}}$ elements of $\boldsymbol{f}_{oct}$ indicate which sub-box at recursion depth $r+1$, contained within B, includes the splat. For example, when there is only one recursion level, $\boldsymbol{f}_{oct} = (1,0,0)$ indicates that the splat is located at the 4th sub-box of the entire space, since '100' is the binary representation of '4'.

3.3 Neural Gaussian Splats

Smol-GS models the 3D scene as a set of neural Gaussian splats with the support of a few MLPs (MLP_o, MLP_c, MLP_s, MLP_r, and MLP_{oct}). Each splat has a coordinate $\boldsymbol{x} \in \mathbb{R}^3$, a feature vector $\boldsymbol{f} \in \mathbb{R}^{n_f}$ (we use $n_f = 8$ in practice, see Sec. 4.2), and a scaling controller $\boldsymbol{s} \in \mathbb{R}^3$. The feature $\boldsymbol{f}$ encodes abstract, splat-wise cues including optical information, geometric transformation, and orientation-induced material properties.

Feature Generation Given the camera center $\boldsymbol{x}_c$ during rendering, we first construct an augmented feature $\boldsymbol{f}^*$ by concatenating the learned splat feature $\boldsymbol{f}$ with viewing information and positional encoding,

$$\boldsymbol{f}^* = \text{concat}\left(\boldsymbol{f}, \frac{\boldsymbol{x}_c - \boldsymbol{x}}{\|\boldsymbol{x}_c - \boldsymbol{x}\|}, \|\boldsymbol{x}_c - \boldsymbol{x}\|, \text{MLP}_{oct}(\boldsymbol{f}_{oct})\right), \tag{2}$$

where $\frac{\boldsymbol{x}_c-\boldsymbol{x}}{\|\boldsymbol{x}_c-\boldsymbol{x}\|}$ defines viewing direction, $\|\boldsymbol{x}_c-\boldsymbol{x}\|$ is the camera-to-splat distance and $\boldsymbol{f}_{oct}$ is the positional embedding defined in Sec. 3.2. The per-splat optical attributes are then predicted as

$$\begin{aligned} o &= \mathrm{MLP}_o(\boldsymbol{f}^*), && \text{(opacity)} \\ \boldsymbol{c} &= \mathrm{MLP}_c(\boldsymbol{f}^*), && \text{(color)} \\ \boldsymbol{r} &= \mathrm{MLP}_r(\boldsymbol{f}^*), && \text{(rotation)} \\ \boldsymbol{s}^* &= \boldsymbol{s} \odot \mathrm{sigmoid}(\mathrm{MLP}_s(\boldsymbol{f}^*)), && \text{(scaling)} \end{aligned} \tag{3}$$

where $o \in \mathbb{R}$, $\boldsymbol{c} \in \mathbb{R}^3, \boldsymbol{r} \in \mathbb{R}^4$, and $\boldsymbol{s}^* \in \mathbb{R}^3$ denote the predicted opacity, color, rotation, and scaling, respectively, and $\odot$ denotes element-wise multiplication. Generating scaling parameters $\boldsymbol{s}^*$ by solely relying on the MLP would cause instability and training collapses. To mitigate this issue, the parameter $\boldsymbol{s}$ is introduced as a splat size bound, and the MLP_s predicts a modulation factor via a sigmoid activation. The scaling matrix $\boldsymbol{S}$ and rotation matrix $\boldsymbol{R}$ defined by $\boldsymbol{s}^*$ and $\boldsymbol{r}$ reconstruct the covariance $\boldsymbol{\Sigma}$ of the splat:

$$\boldsymbol{\Sigma} = \boldsymbol{R}\boldsymbol{S}\boldsymbol{S}^\top\boldsymbol{R}^\top. \tag{4}$$

Rasterization The pixel-wise rasterization follows the α-blending in original 3DGS [18]. For each pixel $\boldsymbol{x}' \in \mathbb{R}^2$ in the 2D plane, the color $C(\boldsymbol{x}')$ is obtained by projecting all neural splats onto the image plane and compositing them along the ray by blending:

$$C(\boldsymbol{x}') = \sum_{i=1}^{N_{\text{ray}}} T_i o_i G(\boldsymbol{x}')\boldsymbol{c}_i, \quad T_i = \prod_{j=1}^{i-1}(1 - o_j G'(\boldsymbol{x}')), \tag{5}$$

where N_{ray} is the number of splats along the ray such that they are ordered by the distance to the camera center, o_i and $\boldsymbol{c}_i$ are the predicted splat opacity and color, respectively. $G(\boldsymbol{x}')$ denotes the projected Gaussian weight:

$$G(\boldsymbol{x}') = \exp\left(-\frac{1}{2}(\boldsymbol{x}'-\boldsymbol{\mu}')^\top \boldsymbol{\Sigma}'^{-1}(\boldsymbol{x}'-\boldsymbol{\mu}')\right). \tag{6}$$

Here, the $\boldsymbol{\mu}' = \boldsymbol{J}\boldsymbol{W}\boldsymbol{x}$ and $\boldsymbol{\Sigma}' = \boldsymbol{J}\boldsymbol{W}\boldsymbol{\Sigma}\boldsymbol{J}^\top\boldsymbol{W}^\top$ are the splat coordinate and covariance after projection onto the 2D plane [21, 52], where $\boldsymbol{J}$ denotes the Jacobian of projective transformation and $\boldsymbol{W}$ is the view matrix.

Overall, parametrization in Smol-GS keeps the geometry explicit, view effects compact, and shares learned spatial information via the positional embeddings.

3.4 Splat-wise Feature Compression: Arithmetic Coding

Each splat carries the scaling controller $\boldsymbol{s} \in \mathbb{R}^3$ and an abstract feature $\boldsymbol{f} \in \mathbb{R}^{n_f}$ encoding color, opacity, transformation, and material cues (see Sec. 3.3). We

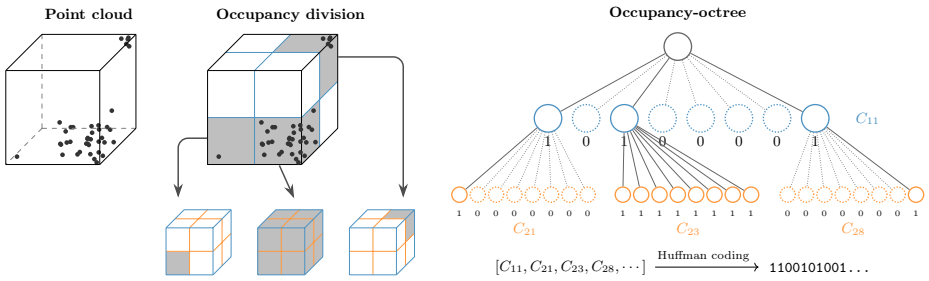

Fig. 5: Occupancy-octree coordinate coding: Given a point cloud (left), we recursively divide the bounding box into eight sub-boxes. Only the non-empty sub-boxes (gray) are further divided (middle). Each division is represented by an 8-bit binary code (1=non-empty, 0=empty). C_{ri} denotes the code of the j^{th} division at r^{th} recursion. The occupancy octree (right) is encoded by arranging all bits in a breadth-first manner. Finally, we apply Huffman coding to further compress the octree bits.

compress them by hash-grid assisted arithmetic coding and a Gaussian distribution prior, which is proposed by HAC [3]. In our setup, the features are low-dimensional, and we choose $n_f = 8$ (see Sec. B.1 for more details).

At first, we condition feature coding on spatial hash descriptors by predicting a distribution and a per-dimension step size with a tiny network MLP_h:

$$\boldsymbol{\mu_f}, \boldsymbol{\Sigma_f}, \Delta_f = \text{MLP}_h(\text{Hash}(\boldsymbol{x})), \tag{7}$$

where $\text{Hash}(\cdot)$ denotes the hash encoding function proposed in InstantNGP [31]. For better compression, the parameters of $\text{Hash}(\cdot)$ are quantized as binary. We then quantize the feature $\boldsymbol{f}$ with the learned steps,

$$\hat{\boldsymbol{f}} = \Delta_f \odot \text{round}(\boldsymbol{f}/\Delta_f), \tag{8}$$

and assume the quantized features follow a Gaussian distribution of $\mathcal{N}(\boldsymbol{f}; \boldsymbol{\mu_f}, \boldsymbol{\Sigma_f})$ (with diagonal $\boldsymbol{\Sigma_f}$). The negative log-likelihood (NLL) of the quantization bin provides a rate proxy:

$$\text{NLL}(\boldsymbol{f}) = -\log \int_{\hat{\boldsymbol{f}}-\Delta_f}^{\hat{\boldsymbol{f}}+\Delta_f} \mathcal{N}(\boldsymbol{f}'; \boldsymbol{\mu_f}, \boldsymbol{\Sigma_f})\, \mathrm{d}\boldsymbol{f}'. \tag{9}$$

The NLL is the theoretical lower bound for the number of bits needed to encode the features. The average NLL over all N splats serves as the training loss to encourage reducing the storage for encoding the features:

$$\mathcal{L}_{\text{NLL}}(\boldsymbol{f}) = \frac{1}{N} \sum_{i=1}^{N} \text{NLL}(\boldsymbol{f}_i). \tag{10}$$

We use arithmetic coding [48] to compress both features $\boldsymbol{f}$ and the scaling controller $\boldsymbol{s}$ of the splats. The loss function of the scaling controller $\boldsymbol{s}$ is defined similarly as Eq. (10) (*i.e.*, $\mathcal{L}_{\text{NLL}}(\boldsymbol{s})$), and quantized as $\hat{\boldsymbol{s}}$ similarly as in Eq. (8).

3.5 Adaptive Density Control

To balance high-fidelity while having a sparse distribution of splats, Smol-GS employs an adaptive density control (ADC) strategy that adjusts the number of splats during training.

Initialization Following 3DGS [18], we initialize coordinates $\boldsymbol{x}$ and scaling controllers $\boldsymbol{s}$ from SfM point clouds [40]. Because SfM points are sparse and non-uniformly scattered, some regions require densification. The features $\boldsymbol{f}$ are initialized randomly.

Densification Inspired by [27], splats with a high magnitude of the coordinate gradient are densified during training. We choose threshold τ_g such that the splat whose coordinate's gradient magnitude is higher than τ_g will be cloned to be a new splat. The newly introduced splat inherits the feature $\boldsymbol{f}$ from the parent splat, while its coordinate is randomly sampled from a Gaussian $\mathcal{N}(\boldsymbol{x}, \boldsymbol{\Sigma})$, where $\boldsymbol{x}$ and $\boldsymbol{\Sigma}$ are the coordinate and covariance of the parent splat, respectively. This process defines new splats and adds detail to under-represented regions of the scene, improving reconstruction quality while increasing overall density.

Pruning During training, an individual splat is pruned when either its average opacity is less than the threshold τ_o, or if it remains consistently invisible.

Opacity Regularization To encourage sparsity in the splat distribution, we apply an ℓ_1 regularization loss on the opacity of all N splats during specific stages of training (see Sec. 3.6):

$$\mathcal{L}_o = \sum_{i=1}^{N} \mathrm{MLP}_o(\boldsymbol{f}_i^*). \tag{11}$$

3.6 Training Pipeline for Smol-GS

We provide a consistent training pipeline, where Smol-GS is trained for 35k iterations with the following five stages: *(1) Warm-up stage* (0–0.5k): The splat coordinates $\boldsymbol{x}$ and scaling controllers $\boldsymbol{s}$ are initialized from SfM, and abstract features $\boldsymbol{f}$ are randomly initialized. *(2) Densification stage* (0.5k–15k): The number of splats is controlled mutually by densification and pruning as described in Sec. 3.5. *(3) Compaction stage* (15k–20k): Both opacity regularization and pruning modules further reduce the number of splats. *(4) Feature compression stage* (20k–30k): Both features $\boldsymbol{f}$ and scaling controllers $\boldsymbol{s}$ of the splats are quantized via arithmetic coding. *(5) Coordinate compression stage* (30k–35k): All splat coordinates are quantized and stored by the occupancy-octree technique. In Fig. 6, we provide training curves for an example scene (*Garden*), which gives an idea of the influence of the different stages.

Optimization Objective In stages *(1)–(2)*, we optimize a photometric objective combining the norm-1 loss $\mathcal{L}_1$ and SSIM loss $\mathcal{L}_\mathrm{s}$ between the rendered image

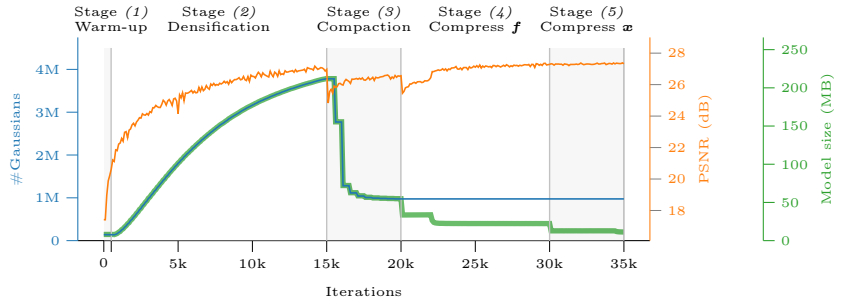

Fig. 6: Training curves over different stages: We log the number of splats (left axis), PSNR (middle), and compressed model size (right) over different stages of training, showing how the compaction and compression do not degrade the overall quality. The number of splats increases steadily during densification (0.5k–15k), then decreases rapidly during compaction (15k–20k).

and the ground truth, the opacity regularization loss $\mathcal{L}_o$, and the quantization regularization losses of $\mathcal{L}_{\text{NLL}}(\boldsymbol{f}) + \mathcal{L}_{\text{NLL}}(\boldsymbol{s})$ such that

$$\mathcal{L} = (1 - \lambda_{\text{s}})\mathcal{L}_1 + \lambda_{\text{s}}\mathcal{L}_{\text{s}} + \lambda_o\mathcal{L}_o + \lambda_q \left[\mathcal{L}_{\text{NLL}}(\boldsymbol{f}) + \mathcal{L}_{\text{NLL}}(\boldsymbol{s})\right]. \tag{12}$$

Here $\lambda_{\text{s}}, \lambda_o$, and λ_q are hyperparameters to control the optimization priorities of the SSIM loss, opacity regularization, and quantization regularization. The opacity regularization is only effective (*i.e.*, $\lambda_o > 0$) in the compaction stage *(3)*, and the quantization regularization is only active (*i.e.*, $\lambda_q > 0$) during the compression stages of *(4)* and *(5)*.

4 Experiments

We evaluate Smol-GS as a full 3DGS compression pipeline. Concretely, we demonstrate our rendering quality vs. model size both quantitatively and qualitatively under the 3DGS.zip survey benchmark setup [1]. Additionally, we demonstrate storage size per module, analyze the training/inference and encoding/decoding time, and ablate central parts of our model.

Data Sets We benchmark our Smol-GS on standard real-world 3D reconstruction data sets: nine scenes from Mip-NeRF 360 [2], two scenes from Tanks and Temples [20], and two scenes from Deep Blending [12]. The training–testing split follows the original settings of 3DGS [18] and Mip-NeRF 360 [2]: Every 8^{th} image is used for testing, and the rest are for training.

Evaluation Metrics The reconstruction quality is evaluated by three widely used metrics: the Peak Signal-to-Noise Ratio (PSNR), the Structural Similarity Index Measure (SSIM), and the Learned Perceptual Image Patch Similarity (LPIPS). PSNR computes the signal preservation level via pixel-wise mean squared error, SSIM measures the statistical structural similarity, and LPIPS

estimates the human perceptual difference based on features extracted from pre-trained neural networks [53]. The compression ratio is measured by the final model size in megabytes (MB). The reported reconstruction quality metrics represent the average value computed between the ground-truth images (from test sets) and the rendered images from the compressed model.

Smol-GS is stored as multiple components: *(i)* occupancy-octree data as bit strings to represent quantized coordinates $\hat{\boldsymbol{x}}$, *(ii)* binary arithmetic codes representing compressed features $\hat{\boldsymbol{f}}$ and scalling controllers $\hat{\boldsymbol{s}}$, *(iii)* MLP architecture/weights ($\text{MLP}_o, \text{MLP}_c, \text{MLP}_s, \text{MLP}_r, \text{MLP}_{oct}$), and *(iv)* additional metadata (*i.e.*, the lower and upper bounds of the scene, the recursion depth R). The total Smol-GS size reported in this paper is the size of a single zip file containing all these four components (following the 3DGS.zip survey [1]). Table 2 represents the size of each component in Smol-GS for the *Garden* scene.

Implementation Details Smol-GS is implemented in PyTorch [35] using the Inria 3DGS repository [18] as a starting point. Parameters of the model are trained using the Adam optimizer [19]. The loss coefficients in Eq. (12) are set to $\lambda_s = 0.2$, and $\lambda_q = 5 \times 10^{-4}$. The thresholds for densification and pruning are set to $\tau_g = 2 \times 10^{-4}$ and $\tau_o = 0.005$. We fix the feature $\boldsymbol{f}$ dimensionality to $n_f = 8$, and all scenes are discretized with $R = 16$ recursion depth for coordinate compression and positional encoding. Additional implementation details, including learning rate (lr) values, lr schedulers, and MLP architectures, are provided in Sec. A.

Smol-GS variants We include three variants of Smol-GS (small, base, big) in the results with different compaction hyperparameters λ_o (small: $\lambda_o = 4{\times}10^{-7}$, base: $\lambda_o = 3{\times}10^{-7}$, big: $\lambda_o = 1{\times}10^{-7}$).

Baselines We compare Smol-GS with the most efficient and recent 3DGS compression methods of Compact3DGS [23], SOG [29], Reducing3DGS [34], Compressed3D [33], CompGS [32], HAC [3], ContextGS [46], HEMGS [25], and HAC++ [4]. We also provide the original vanilla 3DGS results as the prototype baseline.

4.1 Results on 3DGS Compression Benchmarks

Quantitative Results We compare Smol-GS with the baselines on three benchmark data sets using the four evaluation metrics mentioned earlier. The quantitative results are summarized in Table 1 and Fig. 7. The quantitative results of the baselines are collected from the *3DGS.zip* compression survey [1]. In Table 1, Smol-GS achieves the highest compression ratio of all baselines with a considerable margin, reducing the model size by about 150× compared to vanilla 3DGS while maintaining high reconstruction fidelity. In the scatter plots in Fig. 7, Smol-GS outperforms all current state-of-the-art baselines in terms of the trade-off between compression ratio and reconstruction quality. We include further scatter plots for Tanks and Temples and Deep Blending data sets in Fig. A14 in the Appendix.

Table 1: Smol-GS provides state-of-the-art quality–size trade-off: We report PSNR/SSIM/LPIPS and the *total model size in MBs* following the 3DGS.zip survey benchmark [1]. Across Mip-NeRF 360, Tanks and Temples, and Deep Blending, Smol-GS has the smallest sizes with better or similar quality to the strongest baselines, indicating a better rate–distortion trade-off. See also Fig. 7.

Methods	Mip-NeRF 360				Tanks and Temples				Deep Blending			
	PSNR↑	SSIM↑	LPIPS↓	Size↓	PSNR↑	SSIM↑	LPIPS↓	Size↓	PSNR↑	SSIM↑	LPIPS↓	Size↓
3DGS-30K [18]	27.21	0.815	0.214	734.000	23.14	0.841	0.183	411.000	29.41	0.903	0.243	676.000
Compact3DGS [23]	27.08	0.798	0.247	48.800	23.32	0.831	0.201	39.400	29.79	0.901	0.258	43.200
SOG [29]	27.08	0.799	0.230	40.285	23.56	0.837	**0.186**	22.779	29.26	0.894	0.268	17.737
Reducing3DGS [34]	27.10	**0.809**	**0.226**	29.000	23.57	0.840	0.188	14.000	29.63	0.902	**0.249**	18.000
Compressed3D [33]	26.98	0.801	0.238	28.803	23.32	0.832	0.194	17.282	29.38	0.898	0.253	25.299
CompGS [32]	27.03	0.804	0.243	18.000	23.39	0.836	0.200	12.000	29.90	0.906	0.252	12.000
HAC [3]	27.53	0.807	0.238	16.005	24.04	0.846	0.187	8.494	29.98	0.902	0.269	4.561
ContextGS [46]	27.62	0.808	0.237	13.297	24.12	**0.849**	**0.186**	9.902	30.09	0.907	0.265	3.655
HEMGS [25]	27.75	0.806	0.248	12.539	**24.42**	0.848	0.192	6.034	**30.24**	**0.908**	0.266	2.993
HAC++ [4]	27.60	0.803	0.253	8.742	24.22	**0.849**	0.190	5.427	30.16	0.907	0.266	3.051
Smol-GS-base	**27.76**	0.807	0.248	5.860	24.25	0.843	0.199	4.784	30.12	0.905	0.262	2.860
Smol-GS-small	27.61	0.801	0.258	**4.867**	24.21	0.841	0.204	**4.209**	29.97	0.903	0.266	**2.422**

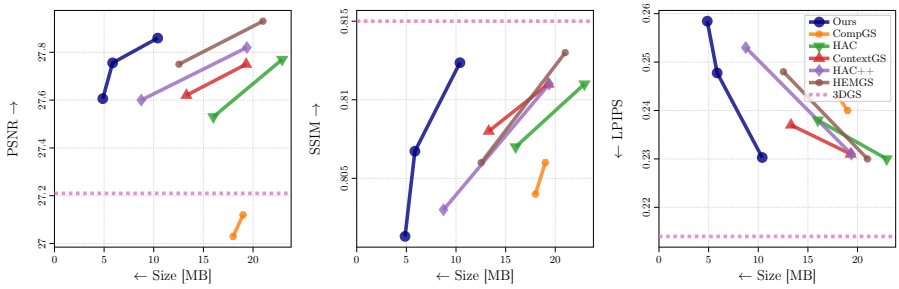

Fig. 7: Comparison of top methods on Mip-NeRF 360. Each curve refers to one method of different variants; the x-axis is the model size in MBs (smaller is better) and the y-axis reports PSNR/SSIM (higher is better) or LPIPS (lower is better). Smol-GS (small–base–big) consistently lies beyond the Pareto frontier, achieving markedly smaller sizes at better quality to strong baselines collected via the 3DGS.zip benchmark. See Fig. A14 in the Appendix for results on Tanks and Temples and Deep Blending.

Qualitative Results Fig. 2 (Mip-NeRF 360) and Fig. 8 (Tanks and Temples and Deep Blending) show qualitative comparison of Smol-GS-base, vanilla 3DGS, and the ground-truth (GT) test views for different scenes. We highlight specific regions by providing zoomed-in views for better subjective comparison. The rendered test views show that Smol-GS is capable of reconstructing scene details with high fidelity and preserving sharp edges. Especially, for challenging objectives such as the white wall in the *Bonsai* and *Playroom* scenes, the sky in the *Train* scene, and the rocket sticker in the *Playroom* scene. Smol-GS clearly generates fewer artefacts than vanilla 3DGS. We link these visual improvements to the neural representation of abstract splats, which is capable to learn the optical properties and material cues of different surfaces adaptively. This representation is substantially more efficient in storing

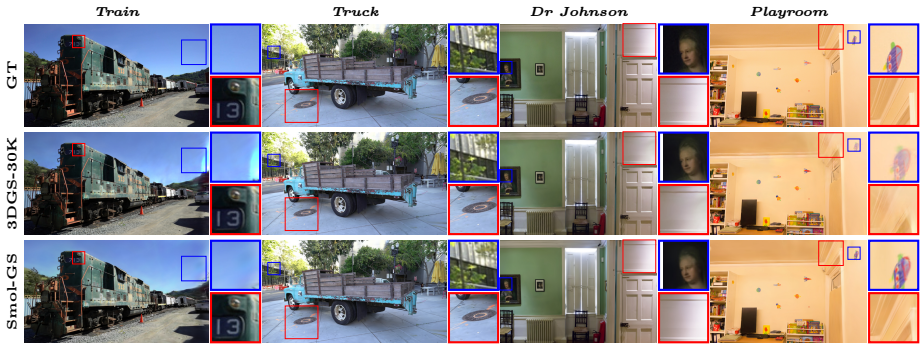

Fig. 8: Qualitative results on the Tanks and Temples and Deep Blending data sets. Smol-GS has fewer artefacts and better preservation of scene details under challenging lighting and materials. The learned splat features in Smol-GS capture view-dependent and material cues more compactly than SH, aligning with its smaller size in MBs and competitive metrics in Table 1.

view-dependent cues compared to the expensive-to-store 3rd degree spherical harmonic coefficients in 3DGS.

Storage Size The required disk space for storing each model component in Smol-GS-base is summarized in Table 2. The splat-wise features $\boldsymbol{f}$ take half of the space, while storing the coordinates $\boldsymbol{x}$ is very efficient thanks to the proposed occupancy-octree encoding. The MLP parameters solely take a negligible portion of the total size due to their tiny architectures. Compared to other methods like HAC++, we save space by eliminating the anchor-offset overhead and allowing low-dimensional features by positional encoding.

Training and Inference Speed The training time of the Smol-GS-base model is around 26.58 minutes (35k iterations) per scene on average, while HAC++ requires 25.75 minutes (30k iterations) on the same hardware. The average rendering speed of Smol-GS-base model is 241.3 fps (Mip-NeRF 360: 210.0 fps, Deep Blending: 399.9 fps, and Tanks and Temples: 223.3 fps).

Encoding and Decoding Time The encoding and decoding timings during inference of Smol-GS-base for the benchmark data sets are presented in Table 3. The fast encoding and decoding speed ensures the practicality of Smol-GS in storage and transmission scenarios. The timing and the speed were measured on NVIDIA H200 GPUs (see Sec. A.1 for details).

Table 2: The size of components of Smol-GS-base (in MB), averaged over all scenes.

Model size (MB)	Total	$\boldsymbol{x}$	$\boldsymbol{f}$	$\boldsymbol{s}$	MLPs	Others
Mip-NeRF 360	6.440	1.412	3.245	1.634	0.134	0.014
Deep Blending	3.345	0.816	1.451	0.942	0.127	0.010
Tanks and Temples	5.525	0.931	2.889	1.580	0.113	0.013

Table 3: Encoding and decoding time of each component of Smol-GS-base (in seconds), values reported averaged over all scenes.

Encoding time (s)	Total	$\boldsymbol{x}$	$\boldsymbol{f}$	$\boldsymbol{s}$	MLPs
Mip-NeRF 360	1.927	0.155	1.092	0.620	0.021
Deep Blending	1.155	0.106	0.675	0.308	0.040
Tanks and Temples	1.606	0.115	0.899	0.537	0.019

Decoding time (s)	Total	$\boldsymbol{x}$	$\boldsymbol{f}$	$\boldsymbol{s}$	MLPs
Mip-NeRF 360	2.948	0.130	1.703	1.087	0.020
Deep Blending	1.549	0.142	0.671	0.538	0.195
Tanks and Temples	2.466	0.090	1.428	0.928	0.016

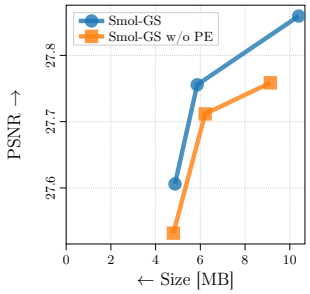

Fig. 9: Ablation study on the positional encoding (PE) module. Our proposed PE improves the trade-off between reconstruction quality (PSNR) and memory consumption on Mip-NeRF 360.

4.2 Ablation Studies

The ablation study in Fig. 9 compares our model with/without positional encoding (PE). The proposed PE module improves the reconstruction quality under the same compression ratio, therefore enhancing the representation efficiency.

Additionally, we conduct different ablation studies to evaluate the impact of each hyperparameter in Smol-GS, including the recursion depth R in occupancy-octree encoding of coordinates (see Fig. A10), the feature dimension n_f (see Fig. A11), the compaction strength λ_o (see Fig. A12), and the arithmetic coding strength λ_q (see Fig. A13). More detailed results are listed in Sec. B.1.

5 Discussion and Conclusion

Conclusion We introduced Smol-GS, a compact 3DGS representation that keeps geometry *explicit* and efficiently packed, and abstracts view-dependent appearance into low-dimensional, arithmetically coded splat features. Smol-GS achieves the highest compression ratio among existing baseline models, maintains competitive reconstruction quality, and preserves primitive structure—thus providing compatibility to various 3DGS applications. The compact storage and efficient encoding/decoding also facilitate real-time rendering and transmission of 3D scenes in resource-constrained environments. Extensive experiments on standard 3D reconstruction data sets validate the effectiveness of Smol-GS through quantitative and qualitative views.

Future Work The discrete representations of splat features in Smol-GS could serve as 3D information tokens, enabling possibilities for combination with other learning modalities and language models. The discrete primitive distribution can potentially be improved by depth estimation techniques. The octree hierarchy implies relations among splats, which can improve representation by leveraging prior knowledge of the geometry.

References

1. Bagdasarian, M.T., Knoll, P., Li, Y., Barthel, F., Hilsmann, A., Eisert, P., Morgenstern, W.: 3DGS.zip: A survey on 3D Gaussian splatting compression methods. In: Computer Graphics Forum. vol. 44, p. e70078. Wiley Online Library (2025)
2. Barron, J.T., Mildenhall, B., Verbin, D., Srinivasan, P.P., Hedman, P.: Mip-NeRF 360: Unbounded anti-aliased neural radiance fields. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 5470–5479 (2022)
3. Chen, Y., Wu, Q., Lin, W., Harandi, M., Cai, J.: HAC: Hash-grid assisted context for 3D Gaussian splatting compression. In: Proceedings of the European Conference on Computer Vision (ECCV). pp. 422–438. Springer (2024)
4. Chen, Y., Wu, Q., Lin, W., Harandi, M., Cai, J.: HAC++: Towards 100x compression of 3D Gaussian splatting. arXiv preprint arXiv:2501.12255 (2025)
5. Chen, Y., Chen, Z., Zhang, C., Wang, F., Yang, X., Wang, Y., Cai, Z., Yang, L., Liu, H., Lin, G.: GaussianEditor: Swift and controllable 3D editing with Gaussian splatting. In: Proceedings of the IEEE/CVF conference on Computer Vision and Pattern Recognition (CVPR). pp. 21476–21485 (2024)
6. Cui, M., Long, J., Feng, M., Li, B., Kai, H.: Octformer: Efficient octree-based transformer for point cloud compression with local enhancement. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 37, pp. 470–478 (2023)
7. Fan, Z., Wang, K., Wen, K., Zhu, Z., Xu, D., Wang, Z.: LightGaussian: Unbounded 3D Gaussian compression with 15x reduction and 200+ fps. In: Advances in Neural Information Processing Systems (NeurIPS). pp. 140138–140158. Curran Associates, Inc. (2024)
8. Fang, G., Wang, B.: Mini-splatting: Representing scenes with a constrained number of Gaussians. In: Proceedings of the European Conference on Computer Vision (ECCV). pp. 165–181. Springer (2024)
9. Fu, C., Li, G., Song, R., Gao, W., Liu, S.: Octattention: Octree-based large-scale contexts model for point cloud compression. In: Proceedings of the AAAI conference on artificial intelligence. vol. 36, pp. 625–633 (2022)
10. Gersho, A., Gray, R.M.: Vector Quantization and Signal Compression, vol. 159. Springer Science & Business Media (2012)
11. Girish, S., Gupta, K., Shrivastava, A.: EAGLES: Efficient accelerated 3D Gaussians with lightweight encodings. In: Proceedings of the European Conference on Computer Vision (ECCV). pp. 54–71. Springer (2024)
12. Hedman, P., Philip, J., Price, T., Frahm, J.M., Drettakis, G., Brostow, G.: Deep blending for free-viewpoint image-based rendering. ACM Transactions on Graphics (ToG) **37**(6) (Dec 2018)
13. Hu, J., Chen, X., Feng, B., Li, G., Yang, L., Bao, H., Zhang, G., Cui, Z.: CG-SLAM: Efficient dense RGB-D slam in a consistent uncertainty-aware 3D Gaussian field. In: Proceedings of the European Conference on Computer Vision (ECCV). pp. 93–112. Springer (2024)
14. Huang, H., Li, L., Cheng, H., Yeung, S.K.: Photo-SLAM: Real-time simultaneous localization and photorealistic mapping for monocular stereo and RGB-D cameras. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 21584–21593 (2024)
15. Huffman, D.A.: A method for the construction of minimum-redundancy codes. Proceedings of the IRE **40**(9), 1098–1101 (2007)

16. Jiang, Y., Yu, C., Xie, T., Li, X., Feng, Y., Wang, H., Li, M., Lau, H., Gao, F., Yang, Y., Chenfanfu, J.: VR-GS: A physical dynamics-aware interactive Gaussian splatting system in virtual reality. In: ACM SIGGRAPH (Conference Paper Track) (2024)
17. Keetha, N., Karhade, J., Jatavallabhula, K.M., Yang, G., Scherer, S., Ramanan, D., Luiten, J.: SplaTAM: Splat track & map 3D Gaussians for dense RGB-D SLAM. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 21357–21366 (2024)
18. Kerbl, B., Kopanas, G., Leimkühler, T., Drettakis, G.: 3D Gaussian splatting for real-time radiance field rendering. ACM Transactions on Graphics (ToG) **42**(4) (2023)
19. Kingma, D.P.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
20. Knapitsch, A., Park, J., Zhou, Q.Y., Koltun, V.: Tanks and temples: Benchmarking large-scale scene reconstruction. ACM Transactions on Graphics (ToG) **36**(4) (Jul 2017)
21. Kopanas, G., Philip, J., Leimkühler, T., Drettakis, G.: Point-based neural rendering with per-view optimization. In: Computer Graphics Forum. vol. 40, pp. 29–43. Wiley Online Library (2021)
22. Lassner, C., Zollhofer, M.: Pulsar: Efficient sphere-based neural rendering. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. pp. 1440–1449 (2021)
23. Lee, J.C., Rho, D., Sun, X., Ko, J.H., Park, E.: Compact 3D Gaussian representation for radiance field. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 21719–21728 (2024)
24. Linde, Y., Buzo, A., Gray, R.: An algorithm for vector quantizer design. IEEE Transactions on Communications **28**(1), 84–95 (2003)
25. Liu, L., Chen, Z., Jiang, W., Wang, W., Xu, D.: HEMGS: A hybrid entropy model for 3D Gaussian splatting data compression. arXiv preprint arXiv:2411.18473 (2024)
26. Lloyd, S.: Least squares quantization in pcm. IEEE Transactions on Information Theory **28**(2), 129–137 (1982)
27. Lu, T., Yu, M., Xu, L., Xiangli, Y., Wang, L., Lin, D., Dai, B.: Scaffold-GS: Structured 3D Gaussians for view-adaptive rendering. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 20654–20664 (2024)
28. Mallick, S.S., Goel, R., Kerbl, B., Steinberger, M., Carrasco, F.V., De La Torre, F.: Taming 3DGS: High-quality radiance fields with limited resources. In: SIGGRAPH Asia 2024 Conference Papers. pp. 1–11 (2024)
29. Morgenstern, W., Barthel, F., Hilsmann, A., Eisert, P.: Compact 3D scene representation via self-organizing Gaussian grids. In: Proceedings of the European Conference on Computer Vision (ECCV). pp. 18–34. Springer (2024)
30. Morton, G.M.: A Computer Oriented Geodetic Data Base and a New Technique in File Sequencing. International Business Machines Company (1966)
31. Müller, T., Evans, A., Schied, C., Keller, A.: Instant neural graphics primitives with a multiresolution hash encoding. ACM Transactions on Graphics (ToG) **41**(4), 1–15 (2022)
32. Navaneet, K., Pourahmadi Meibodi, K., Abbasi Koohpayegani, S., Pirsiavash, H.: CompGS: Smaller and faster Gaussian splatting with vector quantization. In: Proceedings of the European Conference on Computer Vision (ECCV). pp. 330–349. Springer (2024)

33. Niedermayr, S., Stumpfegger, J., Westermann, R.: Compressed 3D Gaussian splatting for accelerated novel view synthesis. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 10349–10358 (2024)
34. Papantonakis, P., Kopanas, G., Kerbl, B., Lanvin, A., Drettakis, G.: Reducing the memory footprint of 3D Gaussian splatting. Proceedings of the ACM on Computer Graphics and Interactive Techniques **7**(1), 1–17 (2024)
35. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al.: Pytorch: An imperative style, high-performance deep learning library. Advances in Neural Information Processing Systems (NeurIPS) **32** (2019)
36. Pateux, S., Gendrin, M., Morin, L., Ladune, T., Jiang, X.: BOGausS: Better optimized Gaussian splatting. arXiv preprint arXiv:2504.01844 (2025)
37. Que, Z., Lu, G., Xu, D.: Voxelcontext-net: An octree based framework for point cloud compression. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 6042–6051 (2021)
38. Ren, K., Jiang, L., Lu, T., Yu, M., Xu, L., Ni, Z., Dai, B.: Octree-GS: Towards consistent real-time rendering with lod-structured 3D Gaussians. arXiv preprint arXiv:2403.17898 (2024)
39. Schnabel, R., Klein, R.: Octree-based point-cloud compression. In: Proceedings of the 3rd Eurographics / IEEE VGTC Conference on Point-Based Graphics. p. 111–121. SPBG'06, Eurographics Association, Goslar, DEU (2006)
40. Schonberger, J.L., Frahm, J.M.: Structure-from-motion revisited. In: Proceedings of the IEEE conference on Computer Vision and Pattern Recognition (CVPR). pp. 4104–4113 (2016)
41. Shao, R., Sun, J., Peng, C., Zheng, Z., Zhou, B., Zhang, H., Liu, Y.: Control4D: Efficient 4D portrait editing with text. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 4556–4567 (2024)
42. Tang, J., Ren, J., Zhou, H., Liu, Z., Zeng, G.: DreamGaussian: Generative Gaussian splatting for efficient 3D content creation. arXiv preprint arXiv:2309.16653 (2023)
43. Vali, M.H., Bäckström, T.: Stochastic optimization of vector quantization methods in application to speech and image processing. In: International Conference on Acoustics, Speech and Signal Processing (ICASSP). pp. 1–5 (2023)
44. Van Den Oord, A., Vinyals, O., Kavukcuoglu, K.: Neural discrete representation learning. In: Advances in Neural Information Processing Systems 30 (NIPS). Curran Associates, Inc. (2017)
45. Wang, H., Vali, M.H., Solin, A.: Compressing 3D Gaussian splatting by noise-substituted vector quantization. In: Scandinavian Conference on Image Analysis (SCIA). pp. 338–352. Springer (2025)
46. Wang, Y., Li, Z., Guo, L., Yang, W., Kot, A., Wen, B.: ContextGS: Compact 3D Gaussian splatting with anchor level context model. In: Advances in Neural Information Processing Systems (NeurIPS). vol. 37, pp. 51532–51551. Curran Associates, Inc. (2024)
47. Westover, L.A.: Splatting: A Parallel, Feed-Forward Volume Rendering Algorithm. The University of North Carolina at Chapel Hill (1991)
48. Witten, I.H., Neal, R.M., Cleary, J.G.: Arithmetic coding for data compression. Communications of the ACM **30**(6), 520–540 (1987)
49. Wu, J., Bian, J.W., Li, X., Wang, G., Reid, I., Torr, P., Prisacariu, V.A.: GaussCtrl: Multi-view consistent text-driven 3D Gaussian splatting editing. In: Proceedings of the European Conference on Computer Vision (ECCV). pp. 55–71. Springer (2024)

50. Yan, C., Qu, D., Xu, D., Zhao, B., Wang, Z., Wang, D., Li, X.: GS-SLAM: Dense visual SLAM with 3D Gaussian splatting. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 19595–19604 (2024)
51. Ye, V., Li, R., Kerr, J., Turkulainen, M., Yi, B., Pan, Z., Seiskari, O., Ye, J., Hu, J., Tancik, M., et al.: gsplat: An open-source library for Gaussian splatting. Journal of Machine Learning Research (JMLR) **26**(34), 1–17 (2025)
52. Yifan, W., Serena, F., Wu, S., Öztireli, C., Sorkine-Hornung, O.: Differentiable surface splatting for point-based geometry processing. ACM Transactions on Graphics (ToG) **38**(6), 1–14 (2019)
53. Zhang, R., Isola, P., Efros, A.A., Shechtman, E., Wang, O.: The unreasonable effectiveness of deep features as a perceptual metric. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2018)
54. Zhang, T., Yu, H.X., Wu, R., Feng, B.Y., Zheng, C., Snavely, N., Wu, J., Freeman, W.T.: PhysDreamer: Physics-based interaction with 3d objects via video generation. In: Proceedings of the European Conference on Computer Vision (ECCV). pp. 388–406. Springer (2024)
55. Zhang, Y., Jia, W., Niu, W., Yin, M.: GaussianSpa: An "optimizing-sparsifying" simplification framework for compact and high-quality 3D Gaussian splatting. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 26673–26682 (2025)
56. Zhuang, J., Kang, D., Cao, Y.P., Li, G., Lin, L., Shan, Y.: Tip-editor: An accurate 3D editor following both text-prompts and image-prompts. ACM Transactions on Graphics (ToG) **43**(4), 1–12 (2024)
57. Zielonka, W., Bagautdinov, T., Saito, S., Zollhöfer, M., Thies, J., Romero, J.: Drivable 3D Gaussian avatars. In: International Conference on 3D Vision (3DV). pp. 979–990. IEEE (2025)
58. Zwicker, M., Pfister, H., Van Baar, J., Gross, M.: Ewa splatting. IEEE Transactions on Visualization and Computer Graphics **8**(3), 223–238 (2002)

A Implementation Details

A.1 Hardware and Software

Our experiments are conducted on NVIDIA H200 GPUs with 141 GB memory. The codes are implemented in Python (python=2.11) using the PyTorch framework (version 2.4.1) [35] with CUDA 12.4. The Gaussian rasterization module is built on top of the Inria 3DGS repository [18]. The arithmetic coding and hash encoding functions modules are built upon the implementations from HAC [3].

A.2 Optimization Details

Learning rates and schedulers The model parameters are optimized using the Adam optimizer [19]. We use an exponential decay scheduler on learning rates of abstract features $\boldsymbol{f}$, coordinates $\boldsymbol{x}$, scaling controller $\boldsymbol{s}$, all MLPs $(\text{MLP}_o, \text{MLP}_c, \text{MLP}_r, \text{MLP}_s, \text{MLP}_h, \text{MLP}_{oct})$, and the hash encoder $\text{Hash}(\cdot)$. For each parameter, the learning rate η_i at iteration i is defined as

$$\eta_i = \exp((1-t)\log(\eta_0) + t\log(\eta_{\text{end}})), \text{ where } \quad t = \text{clamp}(\frac{i}{I}, \text{min} = 0, \text{max} = 1). \tag{13}$$

Here I is the maximum iteration for the learning rate scheduler, η_0 and η_{end} are the initial and final learning rates, respectively. We use $I = 30,000$, and the details of η_0 and η_{end} for different model parameters are shown in Table A4.

Table A4: Learning schedulers of model parameters. Here η_0 and η_{end} denote the initial and final learning rates, and the learning rate follows an exponential decay schedule defined in Eq. (13).

	$\boldsymbol{x}$	$\boldsymbol{f}$	$\boldsymbol{s}$	MLP_o	MLP_c	MLP_r	MLP_s	MLP_h	MLP_{oct}	Hash
η_0	0.0002	0.0075	0.01	0.002	0.008	0.004	0.004	0.005	0.008	0.005
η_{end}	1×10^{-5}	0.0075	0.002	2×10^{-5}	5×10^{-5}	0.004	0.004	1×10^{-5}	0.008	1×10^{-5}

A.3 Adaptive Density Control Settings

During the *Densification stage* (0.5k–15k iterations), the pruning threshold on opacity is set to $\tau_o = 0.005$. The densification threshold on the magnitude of coordinate gradients is defined as $\tau_g = 2\times10^{-4}$. During the *Compaction stage* (15k–20k iterations), there is no densification module applied, and the pruning threshold on opacity is set to $\tau_o = 0.005$.

A.4 Model Hyperparameters

The feature dimension is fixed to $n_f = 8$, and the number of recursions of occupancy-octree for coordinate compression is set to $R = 16$. The parameter selection is based on the ablation experiments at Sec. B.1

MLP architectures The MLP architecture metadata, including the input dim., output dim., hidden dim., number of layers, and activation types of the hidden layer, and output layer is summarized in Table A5. The input dimensions of $\text{MLP}_o, \text{MLP}_c, \text{MLP}_r, \text{MLP}_s$ are $n_f + n_o + 4$, which is defined by the concatenation of the abstract feature $\boldsymbol{f}$ (dimension $n_f = 8$), the output dimension n_{oct} of the octree encoder MLP_{oct}, the 3D viewing direction, and the viewing distance. Here n_{oct} is fixed to be 8.

Hash encoder The hash encoder $\text{Hash}(\cdot)$ contains both 2D and 3D hash tables, with size of 2^{15} and 2^{13}, respectively. There are 4-dimensional features per level. The resolution lists of 2D and 3D hash table are defined as $(130, 258, 514, 1026)$ and $(18, 24, 33, 44, 59, 80, 108, 148, 201, 275, 376, 514)$.

Table A5: Details of the MLP architectures in Smol-GS. Here n_f denotes the feature dimension.

	Input Dim	Layers	Hidden Dim	Output Dim	Activation(Hidden/Output)
MLP_o	$n_f + n_{oct} + 4$	3	128	1	ReLU/Tanh
MLP_c	$n_f + n_o oct + 4$	3	128	3	ReLU/Sigmoid
MLP_r	$n_f + n_{oct} + 4$	3	128	4	ReLU/None
MLP_s	$n_f + n_{oct} + 4$	3	128	3	ReLU/None
MLP_h	96	3	128	$8 + 2n_f$	ReLU/None
MLP_{oct}	$3r$	3	64	n_{oct}	ReLU/None

Regularization hyperparameters For the default settings (Smol-GS-base), the regularization hyperparameters are set to $\lambda_o = 3 \times 10^{-7}$, $\lambda_q = 5 \times 10^{-4}$ and $\lambda_s = 0.01$, the recursion depth $r = 16$, where λ_o, λ_q and λ_s denote the weights for opacity regularization, quantization regularization, and SSIM-loss regularization, respectively.

B Additional Experiments

In this section, we provide ablation studies on different model hyperparameters and optimization hyperparameters in Sec. B.1, additional quantitative results in Sec. B.2, additional qualitative results in Sec. B.3, occupancy-octree analysis in Sec. B.4, encoding and decoding time on each model component per scene in Sec. B.5, and the interpretation of learned abstract features in Sec. B.6.

Table A6: Ablation study on recursion level R of occupancy-octree encoding. The results show that $R = 16$ provides sufficient reconstruction quality.

	Mip-NeRF 360				Tanks and Temples				Deep Blending			
Recursion Level	PSNR↑	SSIM↑	LPIPS↓	Size↓	PSNR↑	SSIM↑	LPIPS↓	Size↓	PSNR↑	SSIM↑	LPIPS↓	Size↓
$R = 14$	27.36	0.787	0.265	5.132	23.58	0.808	0.244	3.769	29.69	0.901	0.272	2.789
$R = 15$	27.60	0.801	0.252	5.513	24.07	0.829	0.220	4.463	30.17	0.906	0.260	2.798
$R = 16$	27.69	0.806	0.248	5.896	24.23	0.842	0.201	4.8	30.05	0.905	0.263	2.747
$R = 17$	27.68	0.807	0.248	6.084	24.32	0.846	0.199	4.813	30.05	0.905	0.262	3.114
$R = 18$	27.70	0.808	0.248	6.207	24.28	0.847	0.195	4.958	30.07	0.904	0.264	2.851

B.1 Ablation Studies

We conduct ablation studies on four different hyperparameters in Smol-GS, including feature dimension n_f, recursion level R, the loss weights on opacity regularization λ_o, and quantization regularization λ_q.

Recursion level R The value of the total recursion depth R controls the compression ratio of splat coordinates. The ablation experiment on recursion level R is presented in Table A6 and Fig. A10, where $R \in \{14, 15, 16, 17, 18\}$ in the experiments. Larger R requires more storage memory of coordinate in a linear manner, and leads to better reconstruction quality. Recursion level $R = 16$ provides a good trade-off between compression ratio and reconstruction quality.

Feature dimension n_f The value of n_f controls the dimensionality of abstract features. The ablation experiment on feature dimension n_f is presented in Table A7 and Fig. A11, where $n_f \in \{4, 6, 8, 16, 24\}$ in the experiments. Larger n_f requires more storage space for features, while improving the reconstruction quality. We choose feature dimension $n_f = 8$ for compact storage and competitive reconstruction quality, as higher dimensionality yields marginal reconstruction gains while substantially increasing storage costs.

Opacity regularization λ_o The value of λ_o controls the strength of opacity regularization. The ablation experiment on opacity regularization strength λ_o is presented in Table A8 and Fig. A12. And the value of λ_o is set to $\{1\times10^{-7}, 2\times10^{-7}, 3\times10^{-7}, 4\times10^{-7}, 5\times10^{-7}\}$ in the experiments. The larger λ_o encourages the model to use fewer splats to represent the scene, resulting in worse reconstruction quality but smaller storage size. And $\lambda_o \in [3\times10^{-7}, 4\times10^{-7}]$ remains proper reconstruction quality. Thus, we select our model variants in this range.

Quantization regularization λ_q The value of λ_q controls the strength of quantization regularization, which encourages the model to use fewer bits to represent the abstract features $\boldsymbol{f}$ and scaling controller $\boldsymbol{s}$. The ablation experiment on quantization regularization strength λ_q is presented in Table A9 and Fig. A13, where $\lambda_q \in \{2\times10^{-4}, 5\times10^{-4}, 1\times10^{-3}, 2\times10^{-3}, 3\times10^{-3}\}$ in the experiments. Larger λ_q encourages the model to use fewer bits for feature representation $\boldsymbol{f}$ and scaling controller $\boldsymbol{s}$, which sacrifices reconstruction quality and reduces the model storage size slowly when $\lambda_q \geq 5\times10^{-4}$. Therefore, we choose $\lambda_q = 5\times10^{-4}$ for our default model Smol-GS-base.

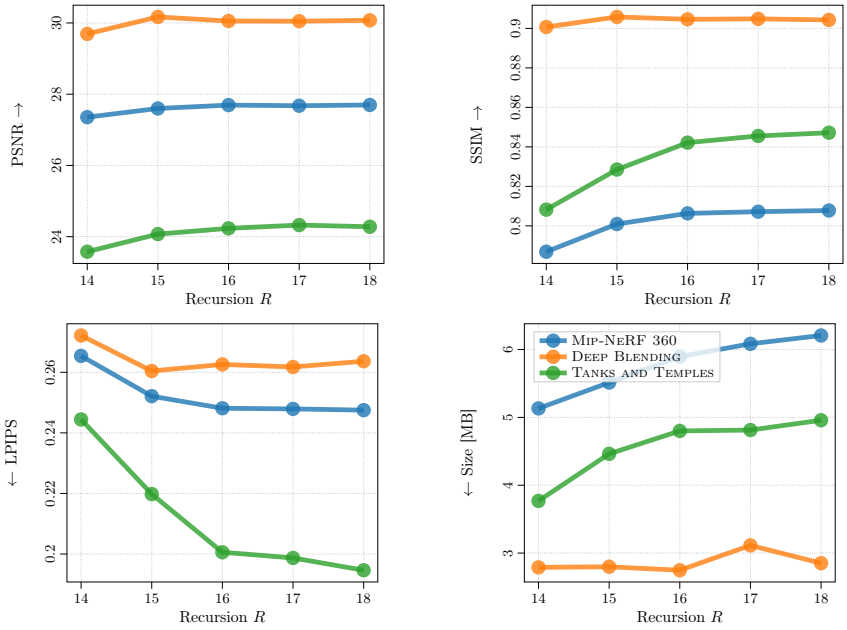

Fig. A10: Ablation study on recursion level R of occupancy-octree encoding. In each subfigure, the x-axis represents different recursion levels R, and the y-axis represents the corresponding metric values: PSNR, SSIM, LPIPS, and Size (in MBs).

Table A7: Ablation study on feature dimension n_f. $n_f = 8$ provides a good trade-off between size and quality.

	Mip-NeRF 360				Tanks and Temples				Deep Blending			
Feature Dimension	PSNR↑	SSIM↑	LPIPS↓	Size↓	PSNR↑	SSIM↑	LPIPS↓	Size↓	PSNR↑	SSIM↑	LPIPS↓	Size↓
$n_f = 4$	27.37	0.799	0.255	5.038	23.91	0.835	0.207	4.054	30.07	0.905	0.263	2.64
$n_f = 6$	27.61	0.803	0.252	5.036	24.06	0.835	0.212	4.2	30.03	0.903	0.267	2.316
$n_f = 8$	27.69	0.806	0.248	5.896	24.23	0.842	0.201	4.8	30.05	0.905	0.263	2.747
$n_f = 16$	27.80	0.809	0.245	8.398	24.30	0.845	0.200	6.663	30.11	0.905	0.262	3.89
$n_f = 24$	27.84	0.809	0.245	10.24	24.19	0.838	0.209	6.354	30.10	0.905	0.264	3.775

Table A8: Ablation study on opacity regularization strength λ_o. We choose $\lambda_o = 3 \times 10^{-7}$ for the trade-off on reconstruction quality and model size.

	Mip-NeRF 360				Tanks and Temples				Deep Blending			
Opacity Regularization	PSNR↑	SSIM↑	LPIPS↓	Size↓	PSNR↑	SSIM↑	LPIPS↓	Size↓	PSNR↑	SSIM↑	LPIPS↓	Size↓
$\lambda = 1e-7$	27.85	0.812	0.231	10.31	24.30	0.846	0.189	7.11	30.16	0.906	0.253	4.678
$\lambda = 2e-7$	27.78	0.810	0.239	7.563	24.16	0.842	0.196	5.596	30.16	0.906	0.258	3.396
$\lambda = 3e-7$	27.69	0.806	0.248	5.896	24.23	0.842	0.201	4.8	30.05	0.905	0.263	2.747
$\lambda = 4e-7$	27.59	0.801	0.258	4.825	24.17	0.840	0.204	4.19	30.07	0.904	0.265	2.458
$\lambda = 5e-7$	27.44	0.794	0.270	4.033	24.07	0.837	0.209	3.951	30.00	0.903	0.268	2.23

B.2 Additional Quantitative Results

The overall quantitative comparison among all baselines on three benchmark data sets is presented as a scatter plot in Fig. A14. The results demonstrate that

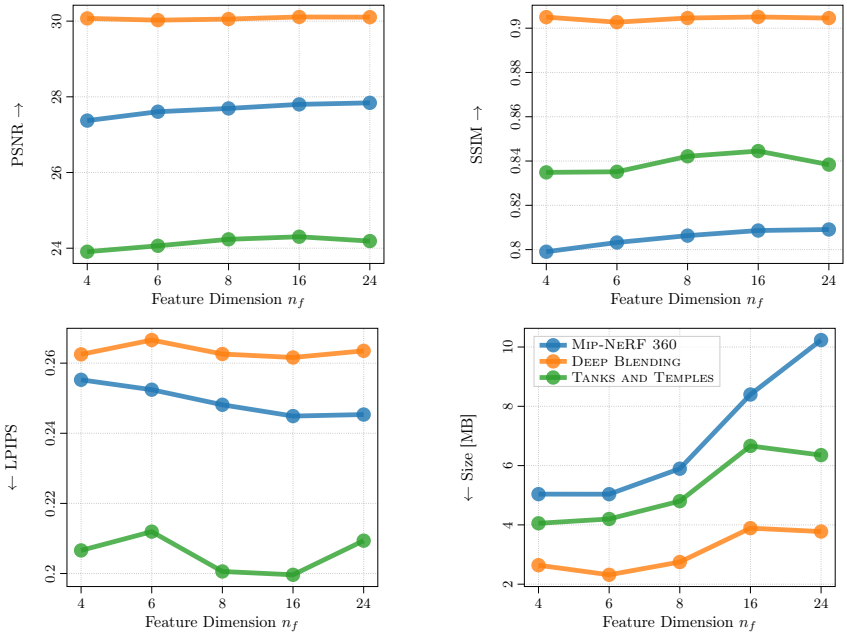

Fig. A11: Ablation study on feature dimension n_f. In each subfigure, the x-axis represents different feature dimensions n_f, and the y-axis represents the corresponding metric values: PSNR, SSIM, LPIPS, and Size (in MBs).

Table A9: Ablation study on quantization regularization strength λ_q. We select $\lambda_q = 5 \times 10^{-4}$ for the best margin effect on overall metrics.

Quantization Regularization	Mip-NeRF 360				Tanks and Temples				Deep Blending			
	PSNR↑	SSIM↑	LPIPS↓	Size↓	PSNR↑	SSIM↑	LPIPS↓	Size↓	PSNR↑	SSIM↑	LPIPS↓	Size↓
$\lambda_q = 2e-4$	27.73	0.807	0.248	6.516	23.67	0.821	0.233	4.666	30.10	0.905	0.262	3.034
$\lambda_q = 5e-4$	27.69	0.806	0.248	5.896	24.23	0.842	0.201	4.8	30.05	0.905	0.263	2.747
$\lambda_q = 1e-3$	27.67	0.806	0.249	5.454	24.22	0.841	0.201	4.467	30.06	0.904	0.262	2.651
$\lambda_q = 2e-3$	27.53	0.804	0.251	5.065	24.08	0.839	0.203	4.089	29.97	0.903	0.265	2.407
$\lambda_q = 3e-3$	27.42	0.802	0.253	4.797	24.08	0.838	0.205	3.93	29.52	0.898	0.281	2.664

Smol-GS achieves competitive reconstruction quality with the lowest memory footprints compared to other baselines.

B.3 Additional Qualitative Results

Additional qualitative comparison among ground truth, 3DGS-30K, Smol-GS-base, and Smol-GS-small is presented in Fig. A15. The results provide more details that Smol-GS preserves scene details with fewer artefacts than 3DGS-30K, even with smaller model sizes. Smol-GS-small loses some fine details compared to Smol-GS-base, but still maintains competitive visual quality.

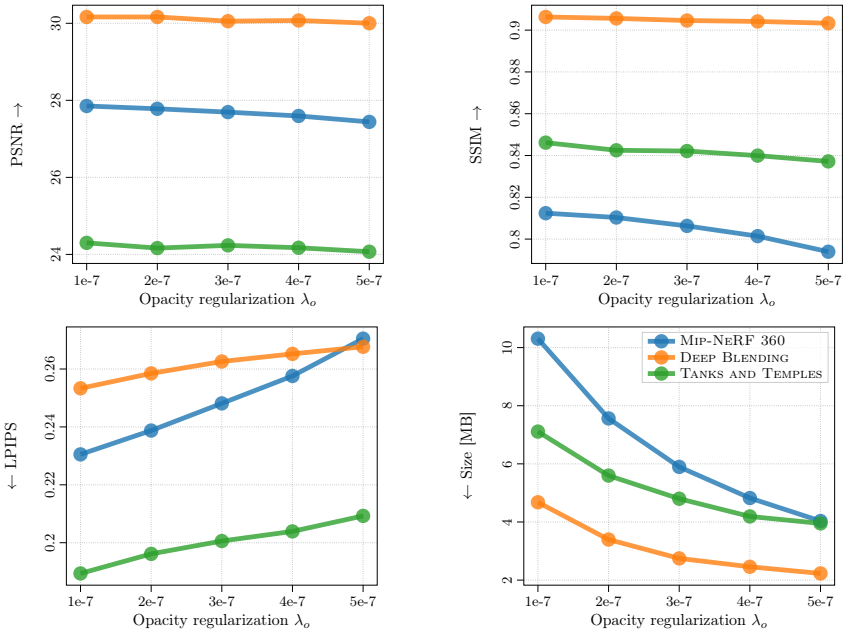

Fig. A12: Ablation study on opacity regularization strength λ_o. In each subfigure, the x-axis represents different opacity regularization strength λ_o, and the y-axis represents the corresponding metric values: PSNR, SSIM, LPIPS, and Size (in MBs).

B.4 Occupancy-Octree Analysis

The occupancy-octree learned in Sec. 3.1 generates a set of byte-codes during the encoding process. The statistical characteristics of the byte-codes are analyzed in Fig. A16. The results show that most divisions generate fewer non-empty sub-boxes, indicating that the byte-codes are highly sparse and thus compressible by entropy coding.

B.5 Encoding and Decoding Details

Per-scene encoding time, decoding time, and reconstruction quality for both Smol-GS-base and Smol-GS-small are summarized in Tables A10 to A15.

B.6 Interpretation of Features

The abstract features learned by Smol-GS-base are visualized in Fig. A17. The features are visualized by the dimension reduction technique and transformation. Given feature $\boldsymbol{f}$, the color visualized is defined as $\text{RGB} = \text{sigmoid}(\text{PCA}(\boldsymbol{f})) \in [0,1]^3$, where PCA denotes the principal component analysis. We can observe that the learned features capture meaningful optical captures and semantics.

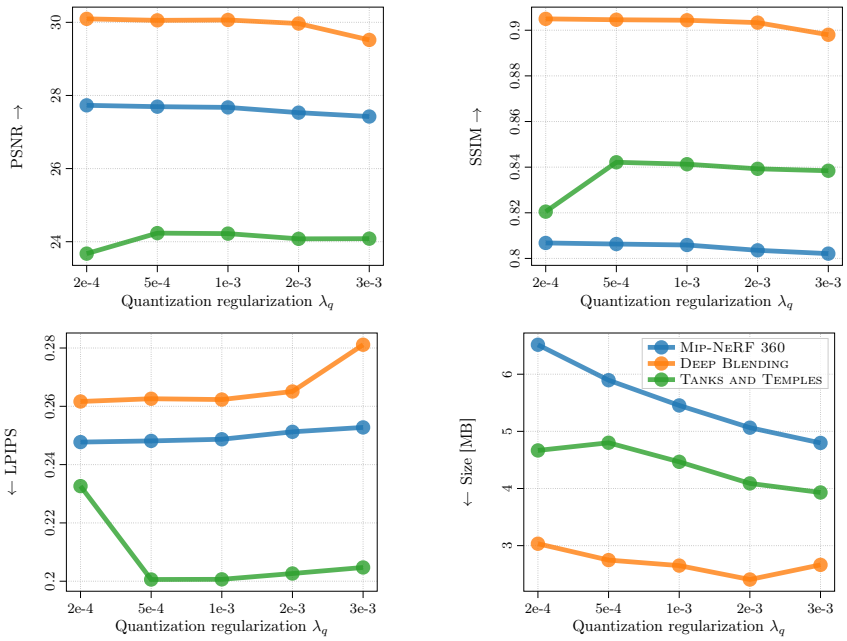

Fig. A13: Ablation study on quantization regularization strength λ_q. In each subfigure, the x-axis represents different quantization regularization strength λ_q, and the y-axis represents the corresponding metric values: PSNR, SSIM, LPIPS, and Size (in MBs).

Table A10: Encoding time (seconds) of each component of Smol-GS-base per scene.

Encoding time (s)		Total	$\boldsymbol{x}$	$\boldsymbol{f}$	$\boldsymbol{s}$	MLPs
Mip-NeRF 360	*Garden*	3.654	0.262	2.158	1.149	0.026
	Bicycle	1.845	0.152	1.044	0.595	0.022
	Stump	2.297	0.156	1.314	0.761	0.022
	Bonsai	1.401	0.162	0.755	0.432	0.020
	Counter	1.442	0.128	0.815	0.454	0.017
	Kitchen	1.473	0.107	0.839	0.479	0.014
	Room	0.961	0.102	0.495	0.320	0.020
	Treehill	1.277	0.101	0.704	0.420	0.022
	Flowers	2.988	0.223	1.704	0.970	0.024
Deep Blending	*Drjohnson*	1.315	0.120	0.764	0.354	0.048
	Playroom	0.996	0.093	0.587	0.262	0.031
Tanks and Temples	*Train*	1.417	0.089	0.822	0.465	0.015
	Truck	1.794	0.142	0.977	0.609	0.023

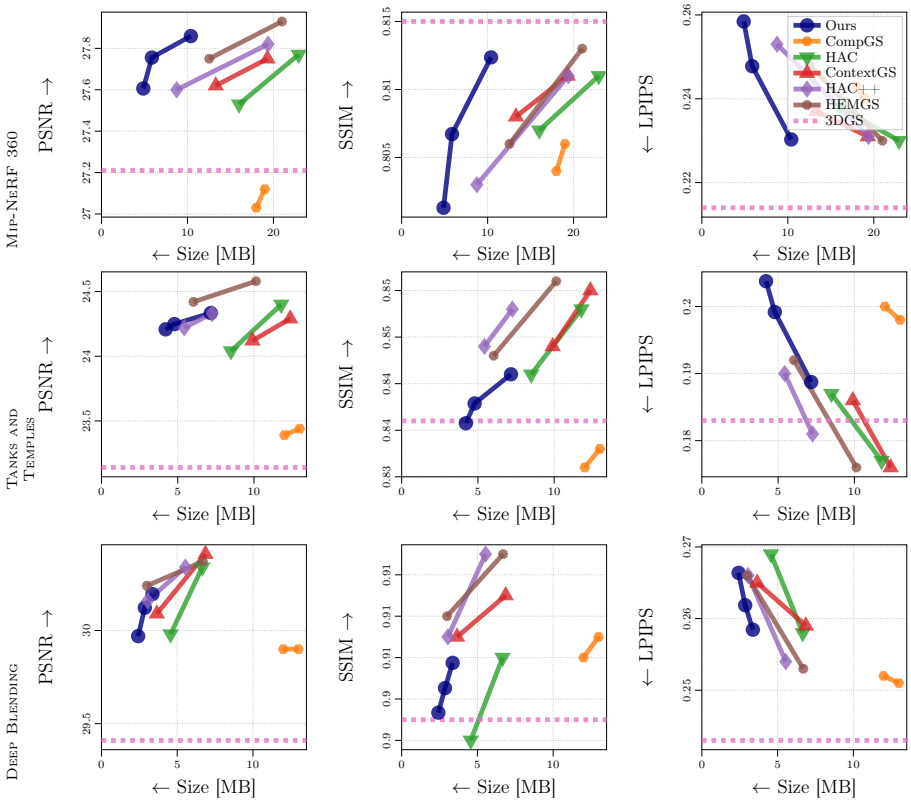

Fig. A14: Comparison of reconstruction quality across baselines. In each subfigure, the x-axis represents the model size (in MBs), and the y-axis represents the corresponding metric values: PSNR (higher is better), SSIM (higher is better), LPIPS (lower is better). Different rows correspond to different benchmark data sets: Mip-NeRF 360, Tanks and Temples, and Deep Blending. Each line connects the results of a specific model over different hyperparameters (variants).

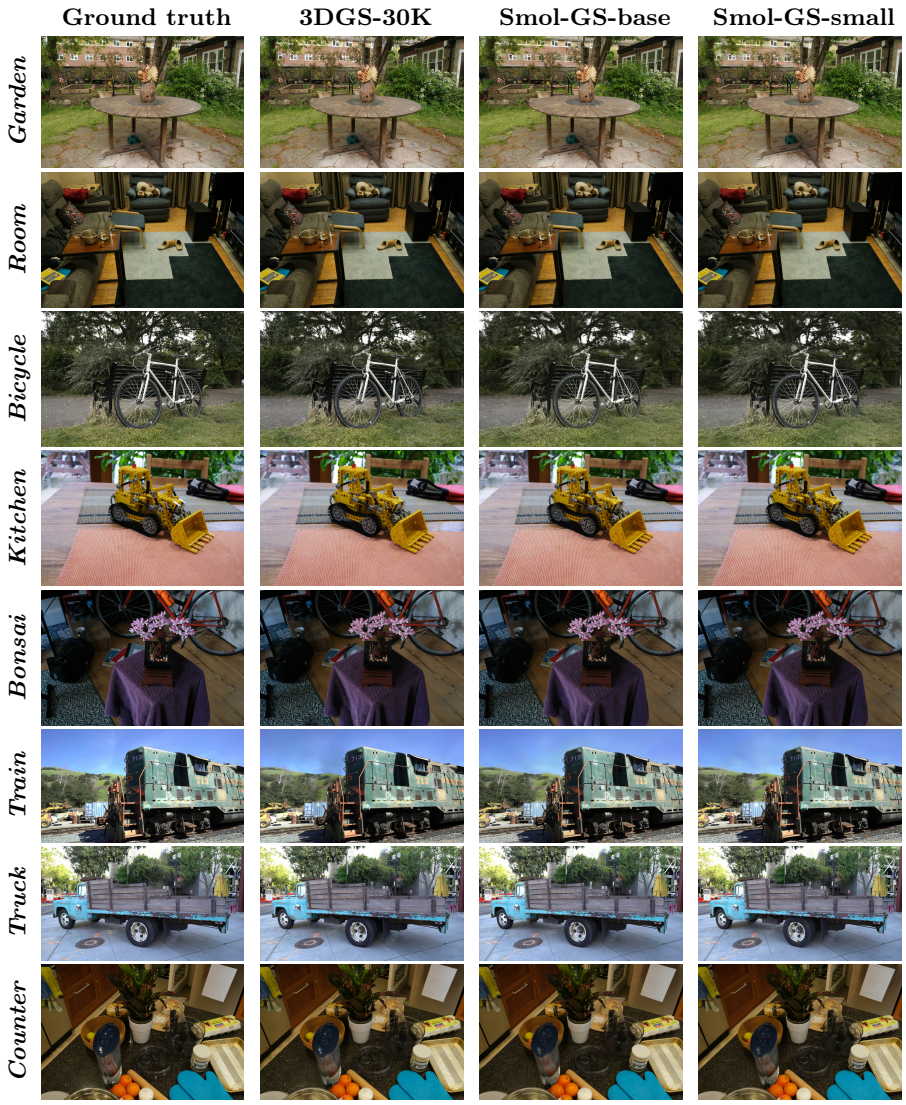

Fig. A15: Comparison of qualitative results among the ground truth, 3DGS-30K, Smol-GS-base, and Smol-GS-small.

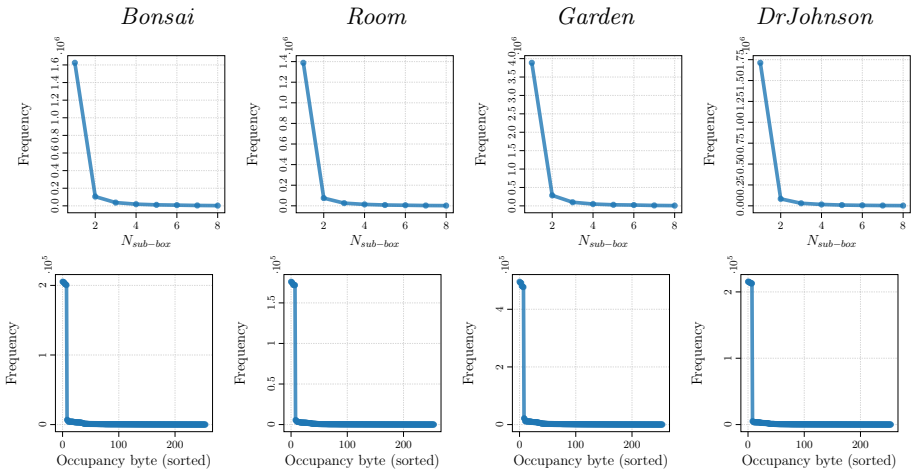

Fig. A16: Statistical analysis of the occupancy-octree byte-codes on four example scenes: *Bonsai, Room, Garden, DrJohnson*. The occupancy-octree generates a series of byte-codes during the encoding process. Each byte-code denotes the existence of a plat in the sub-boxes of the corresponding division. The first row shows the histogram of the number of non-empty sub-boxes per division, and the second row shows the histogram of byte-codes (sorted by frequency). Each column corresponds to a specific scene. The histograms indicate the strong sparsity of the occupancy-octree byte-codes shared by different scenes.

Table A11: Encoding time (seconds) of each component of Smol-GS-small per scene.

Encoding time (s)		Total	$\boldsymbol{x}$	$\boldsymbol{f}$	$\boldsymbol{s}$	MLPs
Mip-NeRF 360	*Garden*	2.938	0.219	1.690	0.943	0.024
	Bicycle	1.558	0.125	0.869	0.510	0.021
	Stump	1.687	0.116	0.958	0.558	0.019
	Bonsai	1.218	0.115	0.678	0.384	0.016
	Counter	1.317	0.120	0.731	0.411	0.022
	Kitchen	1.420	0.135	0.794	0.434	0.022
	Room	0.831	0.088	0.419	0.277	0.023
	Treehill	0.856	0.077	0.481	0.266	0.012
	Flowers	2.411	0.199	1.424	0.728	0.017
Deep Blending	*Drjohnson*	0.883	0.100	0.451	0.295	0.018
	Playroom	0.706	0.095	0.334	0.230	0.023
Tanks and Temples	*Train*	1.297	0.085	0.746	0.428	0.013
	Truck	1.510	0.124	0.836	0.500	0.018

Table A12: Decoding time (seconds) of each component of Smol-GS-base per scene.

Decoding time (s)		Total	$\boldsymbol{x}$	$\boldsymbol{f}$	$\boldsymbol{s}$	MLPs
Mip-NeRF 360	*Garden*	5.507	0.230	3.216	2.030	0.019
	Bicycle	2.882	0.126	1.692	1.042	0.017
	Stump	3.478	0.132	2.001	1.320	0.019
	Bonsai	2.118	0.114	1.210	0.757	0.026
	Counter	2.227	0.105	1.286	0.810	0.021
	Kitchen	2.288	0.095	1.314	0.859	0.015
	Room	1.439	0.095	0.758	0.552	0.028
	Treehill	1.955	0.079	1.117	0.741	0.015
	Flowers	4.635	0.195	2.736	1.676	0.019
Deep Blending	*Drjohnson*	1.657	0.152	0.785	0.608	0.108
	Playroom	1.441	0.132	0.557	0.468	0.281
Tanks and Temples	*Train*	2.180	0.064	1.284	0.814	0.014
	Truck	2.752	0.115	1.572	1.041	0.018

Table A13: Decoding time (seconds) of each component of Smol-GS-small per scene.

Decoding time (s)		Total	$\boldsymbol{x}$	$\boldsymbol{f}$	$\boldsymbol{s}$	MLPs
Mip-NeRF 360	*Garden*	4.485	0.192	2.634	1.630	0.019
	Bicycle	2.420	0.105	1.400	0.890	0.019
	Stump	2.527	0.094	1.438	0.972	0.018
	Bonsai	1.869	0.099	1.068	0.681	0.016
	Counter	1.980	0.095	1.133	0.730	0.018
	Kitchen	2.097	0.104	1.202	0.758	0.025
	Room	1.237	0.072	0.660	0.484	0.017
	Treehill	1.257	0.052	0.717	0.469	0.014
	Flowers	3.624	0.151	2.185	1.265	0.017
Deep Blending	*Drjohnson*	1.285	0.081	0.654	0.530	0.017
	Playroom	0.972	0.068	0.480	0.403	0.018
Tanks and Temples	*Train*	2.004	0.059	1.184	0.744	0.013
	Truck	2.343	0.097	1.368	0.854	0.017

Table A14: Reconstruction quality per scene for Smol-GS-base.

Reconstruction		PSNR↑	SSIM↑	LPIPS↓	Size↓
Mip-NeRF 360	*Garden*	27.327	0.844	0.157	11.283
	Bicycle	24.881	0.720	0.310	5.644
	Stump	26.658	0.767	0.271	7.015
	Bonsai	32.971	0.945	0.188	4.010
	Counter	29.707	0.914	0.191	4.498
	Kitchen	31.440	0.925	0.133	4.470
	Room	32.088	0.924	0.206	2.858
	Treehill	23.210	0.635	0.396	3.782
	Flowers	21.519	0.587	0.379	9.178
Deep Blending	*Drjohnson*	29.627	0.902	0.263	3.331
	Playroom	30.617	0.907	0.261	2.390
Tanks and Temples	*Train*	22.608	0.806	0.241	4.198
	Truck	25.889	0.879	0.157	5.369

Table A15: Reconstruction quality per scene for Smol-GS-small.

Reconstruction		PSNR↑	SSIM↑	LPIPS↓	Size↓
Mip-NeRF 360	*Garden*	27.158	0.836	0.171	8.976
	Bicycle	24.735	0.713	0.321	4.725
	Stump	26.640	0.763	0.284	5.147
	Bonsai	32.854	0.944	0.191	3.669
	Counter	29.602	0.913	0.194	4.098
	Kitchen	31.208	0.923	0.136	4.063
	Room	31.912	0.922	0.210	2.530
	Treehill	22.868	0.616	0.430	2.696
	Flowers	21.477	0.581	0.389	7.900
Deep Blending	*Drjohnson*	29.565	0.901	0.267	2.777
	Playroom	30.374	0.906	0.265	2.067
Tanks and Temples	*Train*	22.593	0.805	0.245	3.827
	Truck	25.823	0.877	0.163	4.590

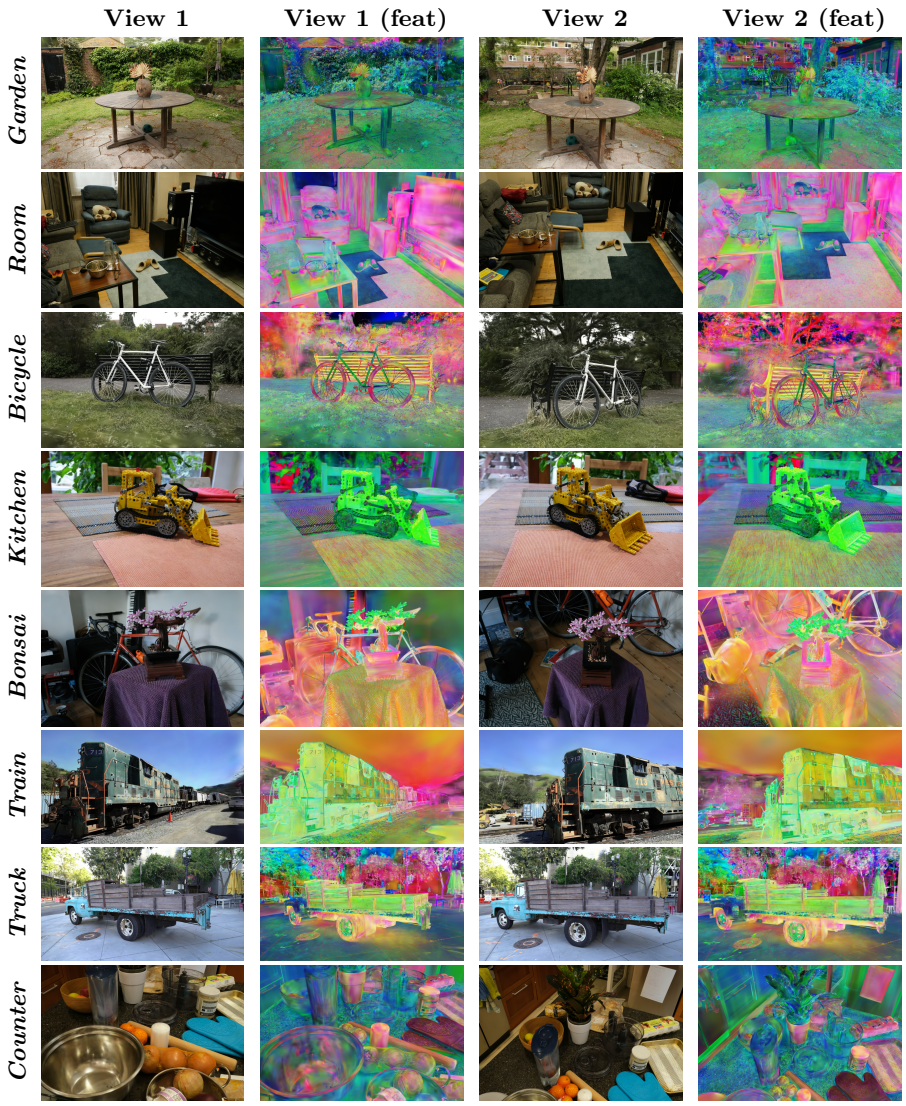

Fig. A17: Visualization of learned abstract features $\boldsymbol{f}$. For each scene, we show two different views and their corresponding rendering image and feature visualizations. The features are visualized by mapping the 8-dimensional features to RGB colors using RGB = sigmoid(PCA($\boldsymbol{f}$)).