Compact3D: Compressing Gaussian Splat Radiance Field Models with Vector Quantization

K L Navaneet* Kossar Pourahmadi Meibodi* Soroush Abbasi Koohpayegani Hamed Pirsiavash

University of California, Davis

Abstract

3D Gaussian Splatting is a new method for modeling and rendering 3D radiance fields that achieves much faster learning and rendering time compared to SOTA NeRF methods. However, it comes with a drawback in the much larger storage demand compared to NeRF methods since it needs to store the parameters for several 3D Gaussians. We notice that many Gaussians may share similar parameters, so we introduce a simple vector quantization method based on K-means algorithm to quantize the Gaussian parameters. Then, we store the small codebook along with the index of the code for each Gaussian. Moreover, we compress the indices further by sorting them and using a method similar to run-length encoding. We do extensive experiments on standard benchmarks as well as a new benchmark which is an order of magnitude larger than the standard benchmarks. We show that our simple yet effective method can reduce the storage cost for the original 3D Gaussian Splatting method by a factor of almost $20\times$ *with a very small drop in the quality of rendered images. Our code is available here: https://github.com/UCDvision/compact3d.*

1. Introduction

Recently, we have seen great progress in radiance field methods to reconstruct a 3D scene using multiple images captured from multiple viewpoints. NeRF [38] is probably the most well-known method that employs an implicit neural representation to learn the radiance field by a deep model implicitly. Although very successful, NeRF methods are very slow in training and rendering. There are various methods to solve this problem, however, they usually come with some cost in the quality of the rendered images.

The Gaussian Splatting method [31] for radiance field rendering is a new paradigm in learning radiance fields. The idea is to model the scene using many Gaussian shapes. Each Gaussian has several parameters including its position in 3D space, covariance matrix, opacity, color, and spherical harmonics of the color that need to be learned from multiple view images. The main advantage of this method over NeRF methods is that the training and rendering are much faster. This is mainly due to the simplicity of projecting 3D Gaussians to the 2D image space and then rendering the view by combining several projected Gaussians along with their opacity using rasterizing. This results in real-time rendering of the scenes on a single GPU (ref. Fig. 1). Another advantage is that unlike NeRF, the 3D structure of the scene is explicitly stored in the parameter space rather than implicit storage in NeRF models. This property enables many operations including editing the 3D scene directly in the parameter space.

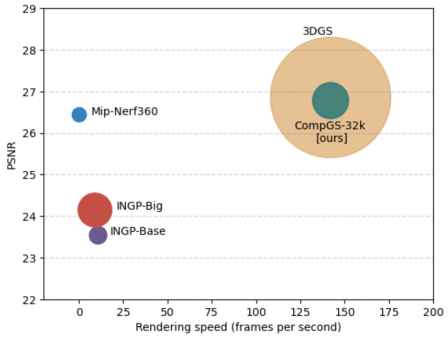

Figure 1. **Comparison of performance, inference speed, and memory time.** The size of the points is proportional to the size of the trained models. Our compressed version of 3DGS , termed CompGS , maintains the speed and performance of 3DGS while reducing its size to the levels of NeRF based approaches.

One of the main drawbacks of the Gaussian Splatting method compared to NeRF variations is that Gaussian Splatting needs at least an order of magnitude more parameters compared to NeRF. This increases the storage and communication requirements of the model and also the memory at the inference time, which can be very limiting in many real-world applications involving smaller IoT devices. For instance, the large memory consumption may be prohibitive in storing, communicating, and rendering several radiance field models on AR/VR headsets.

*Equal contribution

In this paper, we are interested in compacting Gaussian Splatting representations without sacrificing their rendering speed to enable their usage in various applications including low-storage or low-memory IoT devices and AR/VR headsets. Our main intuition is that several Gaussians may be able to share some of their parameters (e.g. covariance matrix) with each other. Hence, we simply vector-quantize parameters and store the codebook along with the index for each Gaussian. This can result in a huge reduction in the storage of the learned radiance fields. Also, it can reduce the memory footprint at the rendering time since the index can act as a pointer to the correct code freeing the memory needed to replicate those parameters for all Gaussians.

To this end, we use simple K-means algorithm to vector quantize the parameters at the learning time. Inspired by various quantization-aware learning methods in deep learning [45], we use the quantized model at the forward pass while updating the non-quantized model at the backward pass. To reduce the computation overhead of running K-means , we update the centroids at every iteration since it is cheap, but update the assignments less frequently (e.g., every 100 iterations) since it is costly. Moreover, since the Gaussians are a set of non-ordered elements, we compress the representation further by sorting the Gaussians based on one of the quantized parameters and storing the indices using the Run-Length-Encoding (RLE) method.

Unlike visual recognition community that uses large scale benchmarks, interestingly, radiance field modeling community traditionally has used small datasets including handful of 3D scenes: maximum of 13 total real world scenes in several papers (e.g.3DGS [31]). We believe this is due to the computational cost of NeRF-based methods (several hours of training time for each scene). Hence, with the recent advancements in this field including Gaussian Splatting that takes only a few minutes to learn a scene, it may be the time to move beyond small benchmarks and evaluate methods on larger scale benchmarks. Therefore, we introduce using an already existing dataset [6] as a new benchmark for radiance field modeling that is an order of magnitude larger than the traditional datasets (200 vs 13 scenes). We believe the community will benefit from using this larger benchmark to evaluate future radiance field methods with a reasonable computational demand.

2. Related Work

Novel-view synthesis methods: Early deep learning techniques for novel-view synthesis used CNNs to estimate blending weights or texture-space solutions [16, 25, 48, 54, 62]. However, the use of CNNs faced challenges with MVS-based geometry and caused temporal flickering. Volumetric representations began with Soft3D [42], and subsequent techniques used deep learning with volumetric ray-marching [27, 50]. Mildenhall et al. introduced Neural Radiance Fields (NeRFs) [38] to improve the quality of synthesized novel views, but the use of a large Multi-Layer Perceptron (MLP) as the backbone and dense sampling slowed down the process a lot. Successive methods sought to balance quality and speed, with Mip-NeRF360 achieving top image quality [4]. Recent advances prioritize faster training and rendering via spatial data structures, encodings, and MLP adjustments [10, 17, 18, 26, 39, 47, 52, 59, 61]. Notable methods, like InstantNGP [39], use hash grids and occupancy grids for accelerated computation with a smaller MLP, while Plenoxels [17] entirely forgo neural networks, relying on Spherical Harmonics for directional effects. Despite impressive results, challenges in representing empty space, limitations in image quality, and rendering speed persist in NeRF methods. In contrast, 3D Gaussian Splatting [31] achieves superior quality and faster rendering without implicit learning [4]. However, the main drawback of 3D Gaussian Splatting is its increased storage compared to NeRF methods which may limit its usage in many applications such as edge devices. We are able to keep the quality and fast rendering speed of 3D Gaussian Splatting method while providing reduced model storage by applying a vector quantization scheme to Gaussian parameters.

Bit Quantization: Reducing the number of bits to represent each parameter in a deep neural network is a commonly used method to quantize models [24, 30, 33] that result in smaller memory footprints. Representing weights in 64 or 32-bit formats may not be crucial for a given task, and a lower-precision quantization can lead to memory and speed improvements. Dettmers et al. [14] show 8-bit quantization is sufficient for large language models. In the extreme case, weights of neural networks can be quantized using binary values. XNOR [44] examines this extreme case by quantization-aware training of a full-precision network that is robust to quantization transformations.

Vector Quantization: Vector quantization (VQ) [15, 19–21] is a lossy compression technique that converts a large set of vectors into a smaller codebook and represents each vector by one of the codes in the codebook. As a result, one needs to store only the code assignments and the codebook instead of storing all vectors. This compression technique has been used in many applications including image compression [12], video and audio codec [34, 37], compressing deep networks [11, 20], and generative models [22, 46, 55]. We apply a similar method to compressing 3DGS models.

Compression for 3D scene representation methods. Since NeRF relies on dense sampling of color values and opacity, the computational costs are significant. To efficiently represent 3D scenes and objects, methods adopt different data structures such as trees [57, 61], point clouds [41, 60], and grids [8, 17, 39, 49, 51, 52]. With grid structures training iterations can be completed in a matter of minutes. However, dense 3D grid structures may require sub-

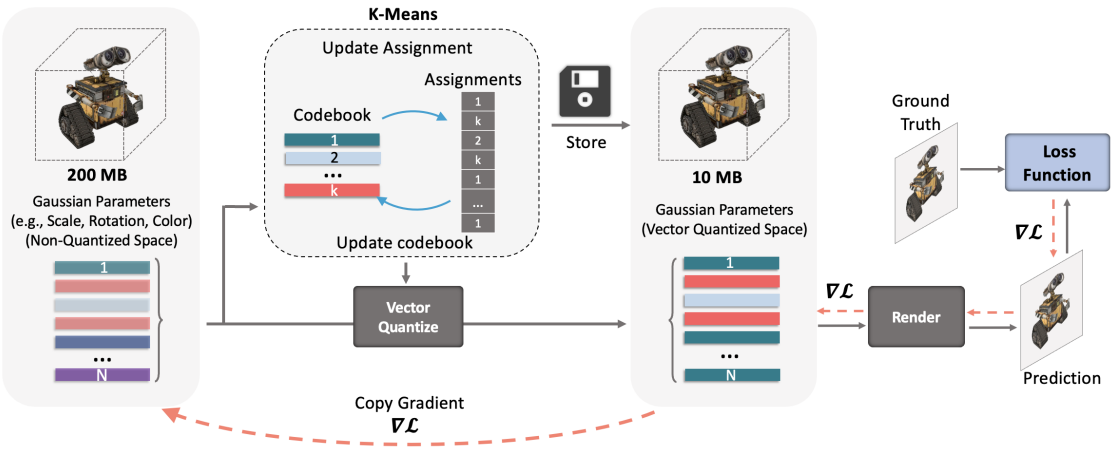

Figure 2. **Overview of CompGS :** We compress 3DGS using vector quantization of the parameters of the Gaussians. The quantization is performed along with the training of the Gaussian parameters. Considering each Gaussian as a vector, we perform K-means clustering to represent the N Gaussians in the model with k cluster centers (codes). Each Gaussian is then replaced by its corresponding code for rendering and loss calculation. The gradients wrt centers are copied to all the elements in the corresponding cluster and the non-quantized versions of the parameters are updated. Only the codebook and code assignments for each Gaussian are stored and used for inference. CompGS maintains the real-time rendering property of 3DGS while compressing it by an order of magnitude.

stantial amounts of memory. Several methods have worked on reducing the size of such volumetric grids [8, 39, 52, 53]. Instant-NGP [39] uses hash-based multi-resolution grids. VQAD [52] replaces the hash function with codebooks and vector quantization.

Another line of work decomposes 3D grids into lower dimensional components, such as planes and vectors, to reduce the memory requirements [8, 29, 53]. Despite reducing the time and space complexity of the 3D scenes, their sizes are still larger than MLP-based methods. VQRF [36] compresses volumetric grid-based radiance fields by adopting the VQ strategy to encode color features into a compact codebook.

While we also employ vector quantization, we differ from the above approaches in the method employed for novel view synthesis. Unlike the NeRF based approaches described above, we aim to compress 3DGS which uses a collection of 3D Gaussians to represent the 3D scene and does not contain grid like structures or neural networks.

Deep Model Compression. Model compression tries to reduce the storage size without changing the accuracy of original models. Model compression techniques can be divided to 1) model pruning [23, 24, 56, 58] that aims to remove redundant layers of neural networks; 2) weight quantization [30, 33, 40], and 3) knowledge distillation [2, 3, 9, 28, 43], in which a compact student model is trained to mimic the original teacher model. Some works have applied these techniques to volumetric radiance fields [13, 35, 61]. For instance, TensoRF [8] decompose volumetric representations via low-rank approximation.

3. Method

Here, we briefly describe the 3DGS [31] method for learning and rendering 3D scenes and explain our vector quantization approach for compressing it.

Overview of 3DGS: 3DGS models a scene using a collection of 3D Gaussians. A 3D Gaussian is parameterized by its position and covariance matrices in the 3D space.

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

where $x-\mu$ is the position vector, μ is the position, and Σ is the 3D covariance matrix of the Gaussian. Since the covariance matrix needs to be positive definite, it is factored into its scale (S) and rotation (R) matrices as $\Sigma = RSS^TR^T$ for easier optimization. In addition, each Gaussian has an opacity parameter α. Since the color of the Gaussians may depend on the viewing angle, the color of each Gaussian is modeled by a Spherical Harmonics (SH) of order 3 in addition to a DC component for the color.

Given a view-point, the collection of 3D Gaussians is efficiently rendered in a differentiable manner to get a 2D image by α-blending of anisotropic splats, sorting, and using a tile-based rasterizer. At the training time, 3DGS renders the training view points and minimizes the loss between the groundtruth and rendered images in the pixel space. The loss is ℓ_1 loss plus an SSIM loss in the pixel space. 3DGS initializes the optimization by a point cloud achieved by a standard SfM method and iteratively prunes the ones with small opacity parameter and adds new ones when the gradient is large. 3DGS paper shows that it is extremely fast to train and is capable of real-time rendering while matching

or outperforming SOTA NeRF approaches in terms of 3D model quality.

Compression of 3DGS: 3DGS requires a few million Gaussians to model a typical scene. With 59 parameters per Gaussian, the storage size of the trained model is an order of magnitude larger than most NeRF approaches (e.g., Mip-NeRF360 [4]). This makes it inefficient for some applications including edge devices. We are interested in reducing the number of parameters. Our main intuition is that many Gaussians may have similar parameter values (e.g., covariance). Hence, we use simple vector quantization using K-means algorithm to compress the parameters. Fig. 2 provides an overview of our approach.

Consider a 3DGS model has n Gaussians with a d dimensional parameter vector for each. We run K-means algorithm to cluster the vectors into k clusters. Then, one can store the model using k vectors of size d and n integer indices (one for each Gaussian). Since $n >> k$, this method can result in a large compression ratios. In a typical scene, n is a few millions while k is a few thousands.

In learning the parameters of 3DGS model, we store the non-quantized parameters. In the forward pass of learning 3DGS, we quantize the parameters and replace them with the quantized version (centroids) to do the rendering and calculate the loss. Then, we do the backward pass to get the gradients for the quantized parameters and copy the gradients to the non-quantized parameters to update them. We use straight-through estimator proposed in STE [7]. After learning, we discard the non-quantized parameters and keep only the codebook and indices of the codes for Gaussians.

Running K-means after every iteration of the gradient descent may be costly. K-means has two steps: updating centroids given assignments, and updating assignments given centroids. We note that the latter is more expensive while the former is a simple averaging. Hence, we update the centroids after each iteration and update the assignments once every t iterations. We use $t = 100$ in our experiments.

Performing a single K-means for the whole d dimensional parameters requires a huge codebook since the different parameters of the Gaussian are not necessarily correlated. Hence, we group similar parameters together and cluster them independently to learn a separate codebook for each. This requires storing multiple indices for each Gaussian. In our main method, we quantize DC component of color, spherical harmonics, scale, and rotation parameters separately, resulting in 4 codebooks. We do not quantize opacity parameter since it is a single scalar and do not quantize the position of the Gaussians since sharing them results in overlapping Gaussians which does not make sense.

Since the indices are integer values, we use fewer number of bits compared to the original parameters to store each. Moreover, 3DGS models the scene as a set of Gaussians where the ordering does not matter. Hence, we sort the Gaussians based on one of the indices so that Gaussians using the same code appear together in the list. Then, for that index, instead of storing n integers, we store the index of the Gaussian that the index of its code in the codebook increases by one. This is similar to run-length-coding for data compression. This method reduces the size of one of the indices from n integers to k integers.

4. Experiments

Implementation details: For all our experiments, we use the publicly available official code repository [1] of 3D Gaussian Splat [31] provided by its authors. There are no changes in the hyperparameters used for training compared to 3D Gaussian Splat. The Gaussian parameters are trained without any vector quantization till $15K$ iterations and K-means quantization is used for the remaining $15K$ iterations. A standard K-means iteration involves distance calculation between all elements (Gaussian parameters) and all cluster centers followed by assignment to the closest center. The centers are then updated using new cluster assignments and the loop is repeated. We use 10 such K-means iterations in our experiments once every 100 iterations till iteration $25K$ and keep the assignments constant thereafter till the last iteration, $30K$ The K-means cluster centers are updated using the non-quantized Gaussian parameters after each iteration of training. The covariance (scale and rotation) and color (DC and harmonics) components of each Gaussian is vector quantized while position (mean) and opacity parameters are not quantized. Additional results with different parameters being quantized are provided in Table 6. Unless mentioned differently, we use a codebook of size 512 for the color and 4016 for the covariance parameters. The scale parameters of covariance are quantized before applying the exponential activation on them. Similarly, quaternion based rotation parameters are quantized before normalization.

Datasets: We primarily show results on three challenging real world datasets - Tanks&Temples[32], Deep Blending [25] and Mip-NeRF360 [4] containing two, two and nine scenes respectively. Additionally, we provide results on our large scale ARKit [6] dataset created using the ARKit dataset. ARKit is an order of magnitude larger than the other datasets with 200 scenes.

ARKit [6] is an extensive indoor scene understanding dataset comprising $5,048$ scans encompassing $1,661$ distinct scenes. Each sequence includes camera poses and utilizes LiDAR scanner-based ARKit scene reconstruction. The videos are recorded using the 2020 iPad Pro and have a resolution of 1920×1440. We exclusively utilize the RGB frames from each video. To construct our dataset, we randomly select 200 raw videos from the ARKit dataset, extracting a uniform sample of 300 frames from each.

Table 1. **Comparison with SOTA methods for novel view synthesis.** 3DGS [31] performs comparably or outperforms the best of the NeRF based approaches while maintaining a high rendering speed during inference. Trained NeRF models are significantly smaller than 3DGS since NeRFs are parameterized using neural networks while 3DGS requires storage of parameters of millions of 3D Gaussians. CompGS is a vector quantized version of 3DGS that maintains the speed and performance advantages of 3DGS while being an order of magnitude smaller. We report the averaged FPS and memory over all datasets. CompGS is identical to 3DGS during inference and thus has the same FPS. *Reproduced using official code. † Reported from 3DGS [31].

Dataset	Mip-NeRF360			Tanks&Temples			Deep Blending			Avg	Avg
Method	SSIM↑	PSNR↑	LPIPS↓	SSIM↑	PSNR↑	LPIPS↓	SSIM↑	PSNR↑	LPIPS↓	FPS	Mem
Plenoxels†	0.626	23.08	0.463	0.719	21.08	0.379	0.795	23.06	0.510	10.3	2.4 GB
INGP-Base†	0.671	25.30	0.371	0.723	21.72	0.330	0.797	23.62	0.423	10.7	13 MB
INGP-Big †	0.699	25.59	0.331	0.745	21.92	0.305	0.817	24.96	0.390	8.86	48 MB
M-Nerf360†	0.792	27.69	0.237	0.759	22.22	0.257	0.901	29.40	0.245	0.09	8.6 MB
3DGS †	0.815	27.21	0.214	0.841	23.14	0.183	0.903	29.41	0.243	142	607 MB
3DGS *	0.813	27.42	0.217	0.844	23.68	0.178	0.899	29.49	0.246	142	607 MB
CompGS 4k	0.804	26.97	0.234	0.836	23.31	0.194	0.904	29.76	0.248	142	51.6 MB
CompGS 32k	0.808	27.16	0.228	0.840	23.47	0.188	0.903	29.75	0.247	142	54.6 MB

Subsequently, we employ the code provided by 3DGS [31] to extract undistorted images and Structure-from-Motion (SfM) information from the input images. Scenes containing fewer than 100 frames with SfM information are omitted from our dataset compilation. This process is reiterated until we assemble 200 scenes, each with more than 100 frames containing SfM information. The dataset can easily be extended in the future by including more of the remaining scenes from the ARKit dataset.

Baselines: As we propose a method (termed CompGS) for compacting 3DGS, we focus our comparisons with 3DGS and different baseline methods for compressing it. We consider bit quantization (denoted as Int-16/8/4 in results) and 3DGS without the harmonic components for color (denoted as 3DGS-No-SH) as an alternative compression methods. Bit-quantization is performed using the standard Absmax quantization [14] technique. Additionally, Table 1 shows comparison with state-of-the-art NeRF approaches [4, 17, 39]. Mip-NeRF360 [4] achieves high performance comparable to 3DGS while Plenoxels [17] and InstantNGP[39] have high frame-rate for rendering and very low training time. InstantNGP and Mip-NeRF360 are also comparable in model size to our compressed model.

Evaluation: For a fair comparison, we use the same train-test split as Mip-NeRF360 [4] and 3DGS [31] and directly report the metrics for other methods from [31]. We also report our reproduced metrics for 3DGS since we observe slightly better results compared to the ones reported in [31] on some of the scenes when we run their code ourselves. We report the standard evaluation metrics of SSIM, PSNR and LPIPS. The common practice is to report the average of PSNR across a set of images and scenes. We do report this as PSNR. However, this metric may be dominated by very accurate reconstructions (smaller errors) since it is based on the geometric average of the errors due to the log operation in PSNR calculation. Hence, for our larger dataset, we also report PSNR-AM for which we average the error across all images and scenes before calculating the PSNR. In comparing model sizes, we normalize all methods by dividing them by the size of our method. The number of Gaussians in a scene affect the memory calculation for 3DGS based methods. When comparing them, we report the average memory over all datasets calculated by setting the number of Gaussians equal to its average over all scenes and datasets.

4.1. Results

Comparison of our results with state-of-the-art (SOTA) approaches is shown in Table 1. Our vector quantized method has a small drop in performance compared to the non-quantized 3DGS but is comparable to SOTA NeRF approaches like Mip-NeRF360. The model memory footprint drastically reduces for CompGS compared to 3DGS , making it comparable to NeRF approaches. This reduces a big disadvantage of 3DGS models and makes it more practical. The compression achieved by CompGS is impressive considering that more than two-thirds of its memory is due to the non-quantized position and opacity parameters. Additionally CompGS maintains the other advantages of 3DGS such as high frame rate for real-time rendering during inference, low inference memory usage and low training time. A limitation of CompGS compared to 3DGS is the overhead in compute and training time introduced by the K-means clustering algorithm. We observe that the training time is close to double the time required for 3DGS when 10 K-means iterations is done every 100 iterations of Gaussian training (our default setting). However, the training time is still orders of magnitude smaller than the high-quality NeRF based approaches like Mip-NeRF360 .

Figure 3. **Qualitative comparison of novel view synthesis approaches**. We visualize images from different scenes across datasets for SOTA NeRF, 3DGS , our CompGS and the No-SH variant of 3DGS . All methods based on 3DGS have better reconstruction of finer details like spokes of the bicycle wheel compared to NeRF approaches. Both compressed versions CompGS and 3DGS-No-SH are similar in appearance to 3DGS with no additional visually apparent errors.

Note that we do not employ techniques for faster K-means such as tracking and updating only those points that have moved sufficiently far from the current position. This can be particularly helpful in our case since the cluster assignments are not significantly altered at the later stages of training. More optimization of the hyperparameters (frequency and number of K-means iterations) too can help reduce the training time. All the experiments were run on a single RTX-6000 GPU.

Comparison of compression methods: In Table 2, we compare the proposed vector quantization based compression against other baseline approaches for compressing 3DGS . Since the spherical harmonic components used for modeling color make up nearly three-fourths of all the parameters of each Gaussian, a trivial compression baseline is to use a variant of 3DGS with only the DC component for color and no harmonics. This baseline (3DGS-No-SH) achieves a high compression with just 23.7% of the original model size but has a drop in performance. Our CompGS approach outperforms 3DGS-No-SH while using less than half its memory. We also consider a variant of CompGS with a single codebook for both SH and DC parameters (termed SH+DC) with a larger codebook of size of 4096. This has a marginal decrease in both memory and performance compared to default CompGS suggesting that correlated parameters can be combined to reduce the number of indices to be stored.

Fig. 3 shows qualitative comparison of CompGS across multiple datasets with both SOTA approaches and compression methods for 3DGS . Both CompGS and 3DGS-No-SH are visually similar to 3DGS , preserving finer details such as the spokes of the bike and bars of dish-rack. Among NeRF approaches, Mip-NeRF360 is closest in terms of quality to 3DGS while InstantNGP trades-off quality for inference and training speed.

All the above approaches are trained using 32-bit precision for all Gaussian parameter values. Post-training

Table 2. **Comparison of compression methods for 3DGS.** We evaluate different baseline approaches for compressing 3DGS. All memory values are reported as a ratio of the method with our smallest model. CompGS performs favorably compared to all methods both in terms of novel view synthesis performance and compression. We find that K-means based quantization of a pretrained model is not effective and that is crucial to perform our quantization during the training of the Gaussian parameters. Bit-quantization approaches closely match the original method when the number of bits is high but the performance greatly degrades when it is reduced to just 4-bits per value. Not quantizing the position (Int-x no-pos) is crucial, especially with higher degrees of quantization. Since harmonics constitute 76% of each Gaussian, 3DGS-no-SH achieves a high level of compression. But CompGS with only quantized harmonics achieves similar compression with nearly no loss in performance compared to 3DGS .

	Mip-NeRF360			Tanks&Temples			Deep Blending			
Method	SSIM	PSNR	LPIPS	SSIM	PSNR	LPIPS	SSIM	PSNR	LPIPS	Mem
3DGS	0.813	27.42	0.217	0.844	23.68	0.178	0.899	29.49	0.246	20.0
3DGS-No-SH	0.802	26.80	0.229	0.833	23.16	0.190	0.900	29.50	0.247	4.8
Post-train K-means 4k	0.768	25.46	0.266	0.803	22.12	0.226	0.887	28.61	0.268	1.7
CompGS SH 4k	0.811	27.25	0.223	0.842	23.57	0.183	0.902	29.60	0.246	4.8
CompGS 4k	0.804	26.97	0.234	0.836	23.31	0.194	0.904	29.76	0.248	1.7
CompGS 32k	0.808	27.16	0.228	0.840	23.47	0.188	0.903	29.75	0.247	1.8
Int16	0.804	27.25	0.223	0.836	23.56	0.185	0.900	29.49	0.247	10.0
Int8 no-pos	0.812	27.38	0.219	0.843	23.67	0.180	0.900	29.47	0.247	5.8
Int8	0.357	14.41	0.629	0.386	12.37	0.625	0.709	21.58	0.457	5.0
Int4 no-pos	0.489	17.42	0.525	0.488	12.94	0.575	0.746	19.90	0.446	3.4
3DGS-No-SH Int16	0.789	26.59	0.237	0.826	23.04	0.198	0.900	29.50	0.248	2.4
CompGS 4k, Int16	0.796	26.83	0.239	0.830	23.21	0.199	0.904	29.76	0.248	1.0

bit quantization of 3DGS to 16-bits reduces the memory by half with very little drop in performance. However, reducing the precision to 8-bits results in a huge degradation of the model. This drop is due to the quantization of the position parameters of the Gaussians. Excluding them from quantization (denoted as Int8) results in a model comparable to the 32-bit variant. However, further reduction to 4-bits degrades the model even when the position parameters are not quantized. Note that bit quantization approaches have significantly lower compression compared to CompGS and they are a subset of the possible solutions for our vector quantization based approach. Similar to 3DGS, CompGS has a small drop in performance when 16-bit quantization is used. The size of CompGS with 32-bit precision is just 1.7 times that of 16-bit since the precision for cluster indices of quantized parameters does not change. Our $16 - bit$ model achieves a $20\times$ reduction in size compared to the original 3DGS model.

Results on ARKit dataset: Table 3 and Fig. 4 show the results on our large-scale ARKit benchmark. We also report the results using PSNR-AM since the dataset is larger than the previous benchmarks so standard PSNR may be dominated by a single easy scene. Our compressed model achieves nearly the same performance as 3DGS with ten times lesser memory. Unlike CompGS , the 3DGS-No-SH method suffers a significant drop in reconstruction quality. It fails to reconstruct large parts of the image for some views in many of the scenes.

Table 3. **Comparison of results on the large scale ARKit dataset.** We introduce ARKit with 200 scenes as a large scale benchmark for novel view synthesis. The benchmark is created using a subset of multi-view images from the ARKit [5] indoor scene understanding dataset. CompGS achieves a high level of compression with nearly identical metrics for view synthesis. We additionally report PSNR-AM as the PSNR calculated using arithmetic mean of MSE over all the scenes in the dataset to prevent the domination of high-PSNR scenes. Compressing such large scale indoor scenes can be particularly helpful for VR applications.

Method	SSIM	PSNR	PSNR-AM	LPIPS	Mem
3DGS	0.909	25.76	20.73	0.226	20.0
3DGS-No-SH	0.905	25.31	20.11	0.234	4.8
CompGS	0.909	25.70	20.73	0.229	1.7

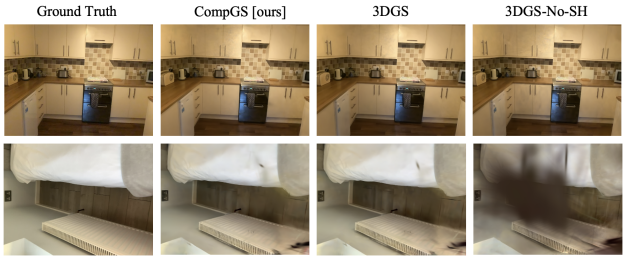

Figure 4. **Results on ARKit dataset.** 3DGS-No-SH fails to reconstruct well in several images while CompGS is nearly identical to 3DGS with a large reduction in model size.

Figure 5. **Qualitative analysis of shared codebook.** We show the generalization of codebook learned using a single scene on various scenes of the Mip-NeRF360 dataset. The codebook was trained on the 'Counter' scene (row-1) and frozen for the remaining scenes. The codebooks for all four parameters (DC, SH, Scale, Rot) are shared across scenes. Both CompGS and CompGS-Shared-Codebook are visually similar to the uncompressed 3DGS with no conspicuous differences between them. 3DGS-No-SH requires twice more memory than CompGS while 3DGS is ten times bigger than CompGS . The scenes and views used for visualization were chosen at random.

4.1.1 Generalization of codebook across scenes

We train our vector quantization approach including the codebook and the code assignments on a single scene ('Counter') of the Mip-NeRF360 dataset. We then freeze the codebook and learn only assignments for the rest of the eight scenes in the dataset and report the averaged performance metrics over all scenes. The results are in table 4. Interestingly, we observe that the shared codebook generalizes well across all scenes with a small drop in performance compared to learning a codebook for each scene. Sharing learnt codebook can further reduce the memory requirement and can help speed up the training of CompGS. The quality of the codebook can be improved by learning it over multiple scenes. Fig. 5 shows qualitative comparison of the same. There are no apparent differences between CompGS and CompGS-Shared-Codebook approaches.

4.1.2 Ablations

Here, we analyze the effect of various components of our quantization method and its hyperparameters on reconstruction performance and model size.

Memory break-down of CompGS: In Table 5, we show the contribution of various components to the final memory usage of CompGS . Out of 59 parameters of each Gaussian, we quantize 55 parameters of color and covariance while

Table 4. **Effect of shared codebook.** We train our vector quantization approach including the codebook and the code assignments on a single scene ('Counter') of the Mip-NeRF360 dataset. We then freeze the codebook and learn only assignments for the rest of the eight scenes in the dataset and report the averaged performance metrics over all scenes. Interestingly, we observe that the shared codebook generalizes well across all scenes with a small drop in performance compared to learning a codebook for each scene. Sharing learnt codebook can further reduce the memory requirement and can help speed up the training of CompGS. The quality of the codebook can be improved by learning it over multiple scenes.

Dataset	Mip-NeRF360		
Method	SSIM$^{\uparrow}$	PSNR$^{\uparrow}$	LPIPS$^{\downarrow}$
3DGS	0.815	27.21	0.214
3DGS *	0.813	27.42	0.217
CompGS 4k	0.804	26.97	0.234
CompGS Shared Codebook	0.797	26.64	0.242

	Non Quant	Quant
Num Params	4	55
Mem (16bit)	68%	32%
Mem (32bit)	81%	19%

k-Means Quant	
Index	Codebook
99%	1%
98%	2%

Table 5. **Breakdown of memory usage in CompGS.** We observe that just 4 non-quantized values of the total 59 values per Gaussian contribute to 68% and 81% of the total memory in our 16-bit and 32-bit variants respectively. For the quantized parameters, nearly the entire memory is used to store the indices while the codebook contributes less than 2%.

the 3 position and 1 opacity parameters are used as is. However, the bulk of the stored memory (68% and 81% for 16- and 32-bits) is due to the non-quantized parameters. For the quantized parameters, nearly all the memory is used to store the cluster assignment indices with less than 2% used for the codebook.

Parameter selection for quantization: Table 6 shows the effect of quantizing different subsets of the Gaussian parameters on the Tanks&Temples dataset. Quantizing the position parameters significantly reduces the performance on both the scenes. We thus do not quantize position in any of our other experiments. Quantizing only the harmonics (SH) of color parameter is nearly identical in size to the no-harmonics (3DGS-No-SH) of 3DGS . Our SH has very little drop in metrics compared to 3DGS while 3DGS-No-SH is much worse off without the harmonics. As more parameters are quantized, the performance of CompGS slowly reduces. The combination of all color and covariance parameters still results in a model with good qualitative and quantitative results.

Effect of codebook size: Fig. 6 shows the effect of codebook length on reconstruction performance for quantization of different Gaussian parameters on the Tanks&Temples dataset. The DC component of color has the smallest drop in performance upon quantization and achieves results

Quantized	Train		Truck		
Params	SSIM$^{\uparrow}$	PSNR$^{\uparrow}$	SSIM$^{\uparrow}$	PSNR$^{\uparrow}$	Mem
3DGS	0.811	21.99	0.878	25.38	20.0
3DGS-No-SH	0.798	21.40	0.871	24.92	4.8
Variants of CompGS					
Pos	0.673	19.81	0.730	21.65	19.0
SH	0.809	21.88	0.876	25.27	4.8
SH, DC	0.806	21.68	0.875	25.24	3.8
Rot(R)	0.808	21.83	0.876	25.32	18.7
Scale(Sc)	0.809	21.79	0.877	25.30	19.0
SH,R	0.805	21.67	0.874	25.20	3.5
SH,Sc	0.806	21.63	0.875	25.18	3.8
SH,Sc,R	0.801	21.64	0.872	25.02	2.6
SH+DC,Sc,R	0.797	21.41	0.868	24.89	1.6
SH,DC,Sc,R	0.801	21.64	0.871	24.97	1.7
SH,DC,Sc,R Int16	0.790	21.49	0.869	24.93	1.0

Table 6. **Effectiveness of quantization on different Gaussian parameters.** Each Gaussian in 3DGS is parameterized using position (pos), scale, rotation (rot) and color (DC and harmonics SH). We analyze the effect of quantizing each of these parameters and their combinations on the view synthesis performance. SH+DC denotes that a single codebook is used for both SH and DC. Position values cannot be quantized without greatly affecting model performance. The rest of the parameters can be simultaneously combined to obtain a high degree of compression without much loss in quality of the generated views.

similar to the non-quantized version with as few as 128 cluster centers. The harmonics (SH) components of color lead to a much bigger drop at lower number of clusters and improve as more clusters are added. Note that CompGS with only SH components is nearly the same size as 3DGS-No-SH but has a much better performance (23.43 for ours v/s 23.14 for 3DGS-No-SH). The covariance parameters (rotation and scale) have a drop in performance even at a codebook size of 1024 but improve as the codebook size is increased. Scale parameter especially benefits with more codes, showing a large improvement with 8192 codes. Based on trade-off between reconstruction performance and model size and training time, We choose 512 clusters

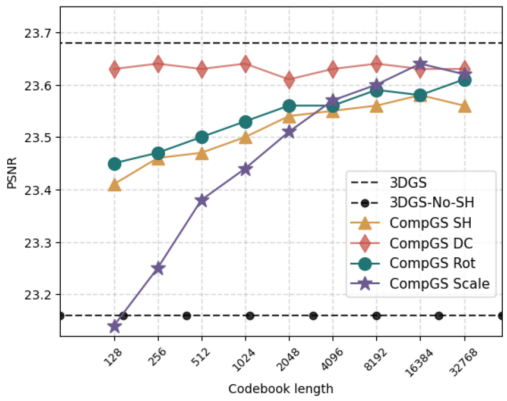

Figure 6. **Effect of codebook length.** We evaluate the performance of CompGS for varying values of codebook length on the Tanks&Temples dataset. We ablate the method independently for the four (SH, DC, Rotation, Scale) parameters by quantized just one of them in each experiment. Scale requires a large number of codes for effective modeling while the performance of DC component of color saturates with just 128 clusters. CompGS SH is significantly better than 3DGS-No-SH for all codebook lengths.

for the color parameters and 4096 clusters for covariance parameters in CompGS 4k variant and 4096 codes for color and 32768 for covariance in CompGS 32k.

Conclusion: 3D Gaussian Splatting efficiently models 3D radiance fields, outperforming NeRF in learning and rendering efficiency at the cost of increased storage. To reduce storage demands, we apply k-means-based vector quantization, compressing indices and employing a compact codebook. Our method cuts the storage cost of 3D Gaussian Splatting by almost 20×, maintaining image quality across benchmarks.

Acknowledgments: This work is partially funded by NSF grant 1845216 and DARPA Contract No. HR00112190135.

References

[1] Official code repository of 3d gaussian splatting for real-time radiance field rendering. https://github.com/graphdeco-inria/gaussian-splatting. 4

[2] Soroush Abbasi Koohpayegani, Ajinkya Tejankar, and Hamed Pirsiavash. Compress: Self-supervised learning by compressing representations. *Advances in Neural Information Processing Systems*, 33:12980–12992, 2020. 3

[3] Lei Jimmy Ba and Rich Caruana. Do deep nets really need to be deep? *arXiv preprint arXiv:1312.6184*, 2013. 3

[4] 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. 2, 4, 5

[5] Gilad Baruch, Zhuoyuan Chen, Afshin Dehghan, Tal Dimry, Yuri Feigin, Peter Fu, Thomas Gebauer, Brandon Joffe, Daniel Kurz, Arik Schwartz, and Elad Shulman. Arkitscenes - a diverse real-world dataset for 3d indoor scene understanding using mobile rgb-d data. In *NeurIPS*, 2021. 7

[6] Gilad Baruch, Zhuoyuan Chen, Afshin Dehghan, Yuri Feigin, Peter Fu, Thomas Gebauer, Daniel Kurz, Tal Dimry, Brandon Joffe, Arik Schwartz, and Elad Shulman. ARKitscenes: A diverse real-world dataset for 3d indoor scene understanding using mobile RGB-d data. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*, 2021. 2, 4

[7] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013. 4

[8] Anpei Chen, Zexiang Xu, Andreas Geiger, Jingyi Yu, and Hao Su. Tensorf: Tensorial radiance fields. In *European Conference on Computer Vision*, pages 333–350. Springer, 2022. 2, 3

[9] Guobin Chen, Wongun Choi, Xiang Yu, Tony Han, and Manmohan Chandraker. Learning efficient object detection models with knowledge distillation. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 742–751, 2017. 3

[10] Zhiqin Chen, Thomas Funkhouser, Peter Hedman, and Andrea Tagliasacchi. Mobilenerf: Exploiting the polygon rasterization pipeline for efficient neural field rendering on mobile architectures. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16569–16578, 2023. 2

[11] Minsik Cho, Keivan A Vahid, Qichen Fu, Saurabh Adya, Carlo C Del Mundo, Mohammad Rastegari, Devang Naik, and Peter Zatloukal. edkm: An efficient and accurate train-time weight clustering for large language models. *arXiv preprint arXiv:2309.00964*, 2023. 2

[12] Pamela C Cosman, Karen L Oehler, Eve A Riskin, and Robert M Gray. Using vector quantization for image processing. *Proceedings of the IEEE*, 81(9):1326–1341, 1993. 2

[13] Chenxi Lola Deng and Enzo Tartaglione. Compressing explicit voxel grid representations: fast nerfs become also small. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 1236–1245, 2023. 3

[14] Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. Llm. int8 (): 8-bit matrix multiplication for transformers at scale. *arXiv preprint arXiv:2208.07339*, 2022. 2, 5

[15] William H Equitz. A new vector quantization clustering algorithm. *IEEE transactions on acoustics, speech, and signal processing*, 37(10):1568–1575, 1989. 2

[16] John Flynn, Ivan Neulander, James Philbin, and Noah Snavely. Deepstereo: Learning to predict new views from the world's imagery. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5515–5524, 2016. 2

[17] Sara Fridovich-Keil, Alex Yu, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. Plenoxels:

Radiance fields without neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5501–5510, 2022. 2, 5

[18] Stephan J Garbin, Marek Kowalski, Matthew Johnson, Jamie Shotton, and Julien Valentin. Fastnerf: High-fidelity neural rendering at 200fps. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 14346–14355, 2021. 2

[19] Allen Gersho and Robert M Gray. *Vector quantization and signal compression*. Springer Science & Business Media, 2012. 2

[20] Yunchao Gong, Liu Liu, Ming Yang, and Lubomir Bourdev. Compressing deep convolutional networks using vector quantization. *arXiv preprint arXiv:1412.6115*, 2014. 2

[21] Robert Gray. Vector quantization. *IEEE Assp Magazine*, 1 (2):4–29, 1984. 2

[22] Shuyang Gu, Dong Chen, Jianmin Bao, Fang Wen, Bo Zhang, Dongdong Chen, Lu Yuan, and Baining Guo. Vector quantized diffusion model for text-to-image synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10696–10706, 2022. 2

[23] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. Deep learning with limited numerical precision. In *International conference on machine learning*, pages 1737–1746. PMLR, 2015. 3

[24] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015. 2, 3

[25] 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. 2, 4

[26] Peter Hedman, Pratul P Srinivasan, Ben Mildenhall, Jonathan T Barron, and Paul Debevec. Baking neural radiance fields for real-time view synthesis. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5875–5884, 2021. 2

[27] Philipp Henzler, Niloy J Mitra, and Tobias Ritschel. Escaping plato's cave: 3d shape from adversarial rendering. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9984–9993, 2019. 2

[28] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015. 3

[29] Binbin Huang, Xinhao Yan, Anpei Chen, Shenghua Gao, and Jingyi Yu. Pref: Phasorial embedding fields for compact neural representations. *arXiv preprint arXiv:2205.13524*, 2022. 3

[30] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2704–2713, 2018. 2, 3

[31] 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, 4, 5, 13

[32] 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. 4

[33] Raghuraman Krishnamoorthi. Quantizing deep convolutional networks for efficient inference: A whitepaper. *arXiv preprint arXiv:1806.08342*, 2018. 2, 3

[34] Yoon Yung Lee and John W Woods. Motion vector quantization for video coding. *IEEE Transactions on Image Processing*, 4(3):378–382, 1995. 2

[35] Lingzhi Li, Zhen Shen, Zhongshu Wang, Li Shen, and Ping Tan. Streaming radiance fields for 3d video synthesis. *Advances in Neural Information Processing Systems*, 35: 13485–13498, 2022. 3

[36] 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. 3

[37] John Makhoul, Salim Roucos, and Herbert Gish. Vector quantization in speech coding. *Proceedings of the IEEE*, 73 (11):1551–1588, 1985. 2

[38] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2020. 1, 2, 13

[39] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Transactions on Graphics (ToG)*, 41(4):1–15, 2022. 2, 3, 5

[40] Parsa Nooralinejad, Ali Abbasi, Soroush Abbasi Koohpayegani, Kossar Pourahmadi Meibodi, Rana Muhammad Shahroz Khan, Soheil Kolouri, and Hamed Pirsiavash. Pranc: Pseudo random networks for compacting deep models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 17021–17031, 2023. 3

[41] Songyou Peng, Chiyu Jiang, Yiyi Liao, Michael Niemeyer, Marc Pollefeys, and Andreas Geiger. Shape as points: A differentiable poisson solver. *Advances in Neural Information Processing Systems*, 34:13032–13044, 2021. 2

[42] Eric Penner and Li Zhang. Soft 3d reconstruction for view synthesis. *ACM Transactions on Graphics (TOG)*, 36(6):1–11, 2017. 2

[43] Antonio Polino, Razvan Pascanu, and Dan Alistarh. Model compression via distillation and quantization. *arXiv preprint arXiv:1802.05668*, 2018. 3

[44] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part IV*, pages 525–542. Springer, 2016. 2

[45] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks, 2016. 2

[46] Ali Razavi, Aaron Van den Oord, and Oriol Vinyals. Generating diverse high-fidelity images with vq-vae-2. *Advances in neural information processing systems*, 32, 2019. 2

[47] Christian Reiser, Songyou Peng, Yiyi Liao, and Andreas Geiger. Kilonerf: Speeding up neural radiance fields with thousands of tiny mlps. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 14335–14345, 2021. 2

[48] Gernot Riegler and Vladlen Koltun. Free view synthesis. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XIX 16*, pages 623–640. Springer, 2020. 2

[49] Katja Schwarz, Axel Sauer, Michael Niemeyer, Yiyi Liao, and Andreas Geiger. Voxgraf: Fast 3d-aware image synthesis with sparse voxel grids. *Advances in Neural Information Processing Systems*, 35:33999–34011, 2022. 2

[50] Vincent Sitzmann, Justus Thies, Felix Heide, Matthias Nießner, Gordon Wetzstein, and Michael Zollhofer. Deepvoxels: Learning persistent 3d feature embeddings. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2437–2446, 2019. 2

[51] 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. 2

[52] Towaki Takikawa, Alex Evans, Jonathan Tremblay, Thomas Müller, Morgan McGuire, Alec Jacobson, and Sanja Fidler. Variable bitrate neural fields. In *ACM SIGGRAPH 2022 Conference Proceedings*, pages 1–9, 2022. 2, 3

[53] Jiaxiang Tang, Xiaokang Chen, Jingbo Wang, and Gang Zeng. Compressible-composable nerf via rank-residual decomposition. In *Advances in Neural Information Processing Systems*, 2022. 3

[54] Justus Thies, Michael Zollhöfer, and Matthias Nießner. Deferred neural rendering: Image synthesis using neural textures. *Acm Transactions on Graphics (TOG)*, 38(4):1–12, 2019. 2

[55] Aaron Van Den Oord, Oriol Vinyals, et al. Neural discrete representation learning. *Advances in neural information processing systems*, 30, 2017. 2

[56] Vincent Vanhoucke, Andrew Senior, and Mark Z Mao. Improving the speed of neural networks on cpus. 2011. 3

[57] Liao Wang, Jiakai Zhang, Xinhang Liu, Fuqiang Zhao, Yanshun Zhang, Yingliang Zhang, Minye Wu, Jingyi Yu, and Lan Xu. Fourier plenoctrees for dynamic radiance field rendering in real-time. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13524–13534, 2022. 2

[58] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. *arXiv preprint arXiv:1608.03665*, 2016. 3

[59] Xiuchao Wu, Jiamin Xu, Zihan Zhu, Hujun Bao, Qixing Huang, James Tompkin, and Weiwei Xu. Scalable neural indoor scene rendering. *ACM Transactions on Graphics (TOG)*, 2022. 2

[60] Qiangeng Xu, Zexiang Xu, Julien Philip, Sai Bi, Zhixin Shu, Kalyan Sunkavalli, and Ulrich Neumann. Point-nerf: Point-based neural radiance fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5438–5448, 2022. 2

[61] Alex Yu, Ruilong Li, Matthew Tancik, Hao Li, Ren Ng, and Angjoo Kanazawa. Plenoctrees for real-time rendering of neural radiance fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5752–5761, 2021. 2, 3

[62] Tinghui Zhou, Shubham Tulsiani, Weilun Sun, Jitendra Malik, and Alexei A Efros. View synthesis by appearance flow. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part IV 14*, pages 286–301. Springer, 2016. 2

Appendices

In this supplementary pdf, we compare the performance of our CompGS with state-of-the-art approaches on the NeRF-Synthetic dataset (Section A). Section B provides insights on the learnt codebook assignments. We also provide additional visualizations and qualitative comparisons on the ARKit dataset in Section C.

A. Results on NeRF-Synthetic dataset

The results (PSNR) for the NeRF-Synthetic dataset [38] are presented in Table 7. Our CompGS approach achieves an impressive average improvement of 1.13 points in PSNR compared to the 3DGS-No-SH baseline while using less than half its memory. As reported in the main submission, we report metrics for 3DGS both from the original paper and using our own runs. We observe an improvement for 3DGS [31] over their official reported numbers by 0.5 points.

B. Analysis of learnt code assignments

In Fig. 7, we plot the sorted histogram of the code assignments (cluster to which each Gaussian belongs to) for each parameter on the 'Train' scene of Tanks&Temples dataset. We observe that just a single code out of the 512 in total is assigned to nearly 5% of the Gaussians for both the SH and DC parameters. Similarly, a few clusters dominate even in the case of rotation and scale parameters, albeit to a lower extent. Such a non-uniform distribution of cluster sizes suggest that further compression can be achieved by using Huffman coding to store the assignment indices.

Table 7. **Results on NeRF-Synthetic dataset.** Here, we present the PSNR values for the synthesized novel views on the NeRF-Synthetic dataset [38]. Our CompGS approach achieves an impressive average improvement of 1.13 points in PSNR compared to the 3DGS-No-SH baseline while using less than half its memory. As reported in the main submission, we report metrics for 3DGS both from the original paper and using our own runs. We observe an improvement of 3DGS over the reported numbers by 0.5points. * indicates our own run.

	Mic	Chair	Ship	Materials	Lego	Drums	Ficus	Hotdog	Avg.
Plenoxels	33.26	33.98	29.62	29.14	34.10	25.35	31.83	36.81	31.76
INGP-Base	36.22	35.00	31.10	29.78	36.39	26.02	33.51	37.40	33.18
Mip-Nerf	36.51	35.14	30.41	30.71	35.70	25.48	33.29	37.48	33.09
Point-NeRF	35.95	35.40	30.97	29.61	35.04	26.06	36.13	37.30	33.30
3DGS	35.36	35.83	30.80	30.00	35.78	26.15	34.87	37.72	33.32
3DGS*	36.80	35.51	31.69	30.48	36.06	26.28	35.49	38.06	33.80
3DGS-No-SH	34.37	34.09	29.86	28.42	34.84	25.48	32.30	36.43	31.97
CompGS 4k	35.99	34.92	31.05	29.74	35.09	25.93	35.04	37.04	33.10

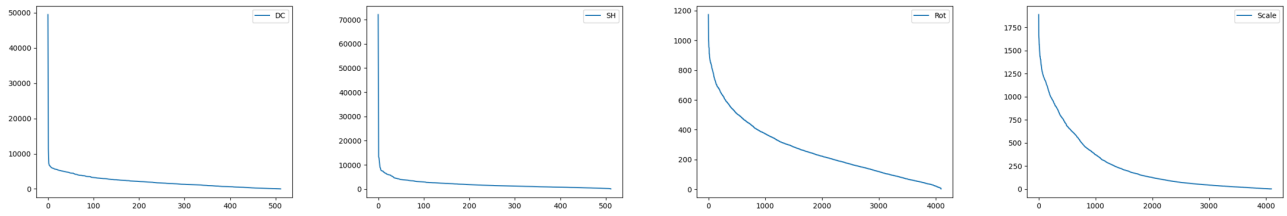

Figure 7. **Histogram of code assignments.** We plot the sorted histogram of the code assignments (cluster to which each Gaussian belongs to) for each parameter on the 'Train' scene of Tanks&Temples dataset. We observe that just a single code out of the 512 in total is assigned to nearly 5% of the Gaussians for both the SH and DC parameters. Similarly, a few clusters dominate even in the case of rotation and scale parameters, albeit to a lower extent. Such a non-uniform distribution of cluster sizes suggest that further compression can be achieved by using Huffman coding to store the assignment indices.

Figure 8. **Visualization of results on Sythetic-NeRF dataset.** We compare the performance of our compressed CompGS with the original 3DGS and 3DGS-No-SH approaches on different scenes of the NeRF-Synthetic dataset. The difference between CompGS and 3DGS-No-SH is apparent in some of these scenes. E.g., 3DGS-No-SH fails to effectively model the brown color of branches and shadows and bright light on the leaves of the 'Ficus' scene. All approaches including 3DGS have imperfect reconstruction in some of the scenes like 'Drums' and 'Lego'. The scenes and views used for visualization were chosen at random.

C. Qualitative comparison on ARKit dataset.

Figures 9 and 10 provide qualitative results on the ARKit dataset.

Figure 9. **Visualization of ARKit dataset.** ARKit is a 3D indoor scene dataset captured using a iPads/iPhones. The dataset consists of videos of indoor environments like houses and office space from multiple view-points. We uniform sample images from each video to form our benchmark dataset for novel view synthesis. Some sample images from different scenes are shown in this figure. The dataset presents unique challenges such as the presence of motion blur due to the use of videos.

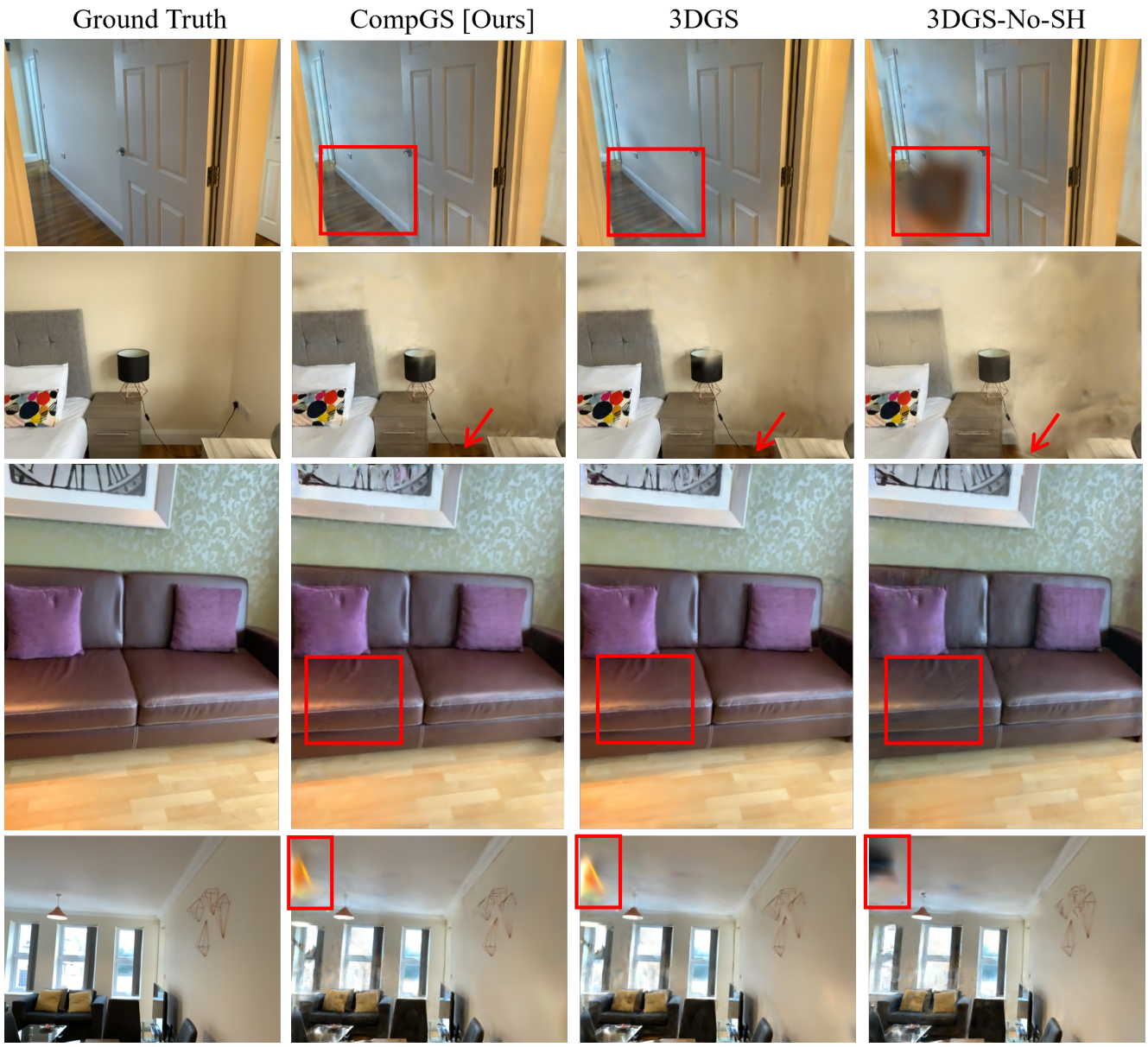

Figure 10. **Qualitative analysis on ARKit dataset.** We visualize the results of CompGS along with the uncomressed 3DGS and its variant 3DGS-No-SH . Presence of large noisy blobs is a common error mode for 3DGS-No-SH on this dataset. It also fails to faithfully reproduce the colors and lighting in several scenes. The visual quality of the synthesized images for all methods is lower on this dataset compared to the scenes present in standard benchmarks like Mip-NeRF360 , indicating its utility as a novel benchmark. Further comparison with various NeRF based approaches and more analysis can help improve the results on this dataset.