Compressed 3D Gaussian Splatting for Accelerated Novel View Synthesis

Simon Niedermayr
simon.niedermayr@tum.de

Josef Stumpfegger
ga87tux@mytum.de

Rüdiger Westermann
westermann@tum.de

Technical University of Munich

Abstract

Recently, high-fidelity scene reconstruction with an optimized 3D Gaussian splat representation has been introduced for novel view synthesis from sparse image sets. Making such representations suitable for applications like network streaming and rendering on low-power devices requires significantly reduced memory consumption as well as improved rendering efficiency. We propose a compressed 3D Gaussian splat representation that utilizes sensitivity-aware vector clustering with quantization-aware training to compress directional colors and Gaussian parameters. The learned codebooks have low bitrates and achieve a compression rate of up to $31\times$ *on real-world scenes with only minimal degradation of visual quality. We demonstrate that the compressed splat representation can be efficiently rendered with hardware rasterization on lightweight GPUs at up to* $4\times$ *higher framerates than reported via an optimized GPU compute pipeline. Extensive experiments across multiple datasets demonstrate the robustness and rendering speed of the proposed approach.*

1. Introduction

Novel view synthesis aims to generate new views of a 3D scene or object by interpolating from a sparse set of images with known camera parameters. NeRF [16] and its variants have proposed the use of direct volume rendering to learn a volumetric radiance field from which novel views can be rendered. However, expensive neural network evaluations prohibit efficient training and rendering. Recent research utilizes explicit scene representations such as voxel-based [24] or point-based structures [29] to enhance rendering efficiency. The use of 3D voxel grids on the GPU in combination with a multiresolution hash encoding of the input [17] significantly reduces the operations needed and permits real-time performance.

While achieving excellent reconstruction quality and speed, many NeRF-style approaches require exhaustive memory resources. This affects both the training and rendering times and often prohibits the use of such representations in applications like network streaming and mobile rendering. To overcome these limitations, dedicated compression schemes for the learned parametrizations on regular grids have been proposed, including vector quantized feature encoding [15], learned tensor decomposition [3] or frequency domain transformations[20, 32].

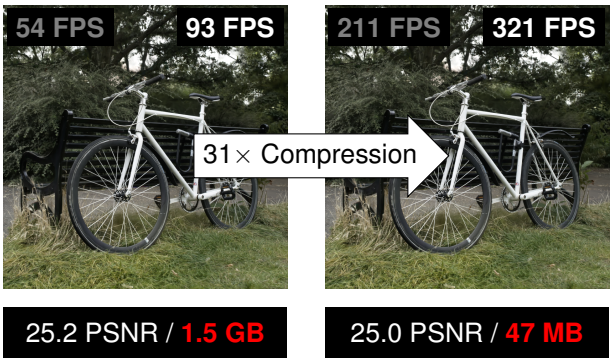

Figure 1. Our method achieves a $31\times$ compression at indiscernible loss in image quality and greatly improves rendering speed compared to [13]. Framerates in grey and white, respectively, are taken on NVIDIA's RTX 3070M and RTX A5000 at 1080p resolution.

Recently, differentiable 3D Gaussian splatting [13] has been introduced to generate a sparse adaptive scene representation that can be rendered at high speed on the GPU.The scene is modeled as a set of 3D Gaussians with shape and appearance parameters, which are optimized via differentiable rendering to match a set of recorded images. The optimized scenes usually consist of millions of Gaussians and require up to several gigabytes of storage and memory. This makes rendering difficult or even impossible on low-end devices with limited video memory, such as handhelds or head-mounted displays. Gaussians are rendered using a specialized compute pipeline, which shows real-time performance on high-end GPUs. This pipeline, however, cannot be seamlessly integrated into VR/AR environments

or games to work in tandem with hardware rasterization of polygon models.

We address the storage and rendering issue of 3D Gaussian splatting by compressing the reconstructed Scene parameters and rendering the compressed representation via GPU rasterization. To compress the scenes, we first analyze its components and observe that the SH coefficients and the multivariate Gaussian parameters take up the majority of storage space and are highly redundant. Inspired by previous work on volumetric radiance field compression[15, 25] and deep network weight quantization, we derive a compression scheme that reduces the storage requirements of typical scenes by up to a factor of $31\times$. Our compression scheme consists of three main steps:

- Sensitivity-aware clustering: We derive a sensitivity measure for each scene parameter by calculating its contribution to the training images. Color information and Gaussian parameters are encoded into compact codebooks via sensitivity-aware vector quantization.
- Quantization-aware fine-tuning: To regain information that is lost during clustering we fine-tune the scene parameters at reduced bit-rates using quantization-aware training.
- Entropy encoding: 3D Gaussians are linearized along a space-filling curve to exploit the spatial coherence of scene parameters with entropy and run-length encoding.

Further, we propose a renderer for the compressed scenes using GPU sorting and rasterization. It enables novel view synthesis in real-time, even on low-end devices, and can be easily integrated into applications rendering polygonal scene representations. Due to the reduced memory bandwidth requirements of the compressed representation and the use of hardware rasterization, a significant speed-up is achieved over the compute pipeline by Kerbl et al. [13].

We show the state-of-the-art quality of novel view rendering on benchmark datasets at significantly reduced memory consumption and greatly improved rendering performance (Fig. 1). The compressed scenes can be used in applications requiring network streaming, and they can be rendered on low-end devices with limited video memory and bandwidth capacities. We perform a number of experiments on benchmark datasets to empirically validate our method across different scenarios. The contribution of each individual step is demonstrated with an ablation study.

2. Related Work

Our work builds upon previous works in novel view synthesis via differentiable rendering and scene compression.

Novel View Synthesis Neural Radiance Fields (NeRF) [16] use neural networks to model a 3D scene. They represent the scene as a density field with direction-dependent colors that are rendered with volume rendering. The field is reconstructed from a set of images with known camera parameters using gradient-based optimization of the volume rendering process.

To speed up training and rendering efficiency, a number of different scene models have been proposed. Most often, structured space discretizations like voxel grids [10, 24, 28], octrees [6] or hash grids [17] are used to represent the scene. To avoid encoding empty space, point-based representations have been proposed. Xu *et al*. [29] perform nearest neighbor search in a point cloud to aggregate local features, and Rückert *et al*. [21] render a point cloud with deep features and use deferred neural rendering to generate the final image.

More recently, differentiable splatting [7, 13] has been positioned as a powerful alternative to NeRF-like approaches for novel view synthesis. In particular, 3D Gaussian Splatting [13] offers state-of-the-art scene reconstruction, by using a scene model consisting of an optimized set of 3D Gaussian kernels that can be rendered efficiently. Differentiable rendering on a set of training images is used to adaptively refine an initial set of Gaussian kernels and optimize their parameters.

NeRF Compression While grid-based NeRF variants achieve high rendering performance due to GPU ray-marching, in particular, the use of full spatial grids introduces considerable storage costs. Tensor decomposition [3, 26], frequency domain transformation [20, 32] and voxel pruning [4] have been proposed to reduce the memory consumption of grid-based NeRFs. Takikawa *et al*. [25] perform vector quantization during training with a learnable index operation. Li *et al*. [15] compress grid-based radiance fields by up to a factor of $100\times$ using post-training vector quantization. The use of a hash encoding on the GPU in combination with vector quantization of latent features reduces the required memory and permits high rendering performance [17]

A number of works have especially addressed memory reduction during inference, to make grid-based scene representations more suitable for low-end devices with limited video memory [19, 27]. To our knowledge, our approach is the first that aims at the compression of point-based radiance fields to enable high-quality novel view synthesis at interactive frame rates on such devices.

Quantization-Aware Training Rastegari et al. [18] simulate weight quantization during training to reduce quantization errors when using low-precision weights for inference. The use of quantization-aware training has been explored for neural scene representations [8] and voxel-based NeRFs [12], demonstrating effective weight quantization with negligible loss in rendering quality.

To reduce the size and latency of neural networks, various approaches for weight quantization have been explored [8, 11, 12, 18]. These methods rely on the observation that

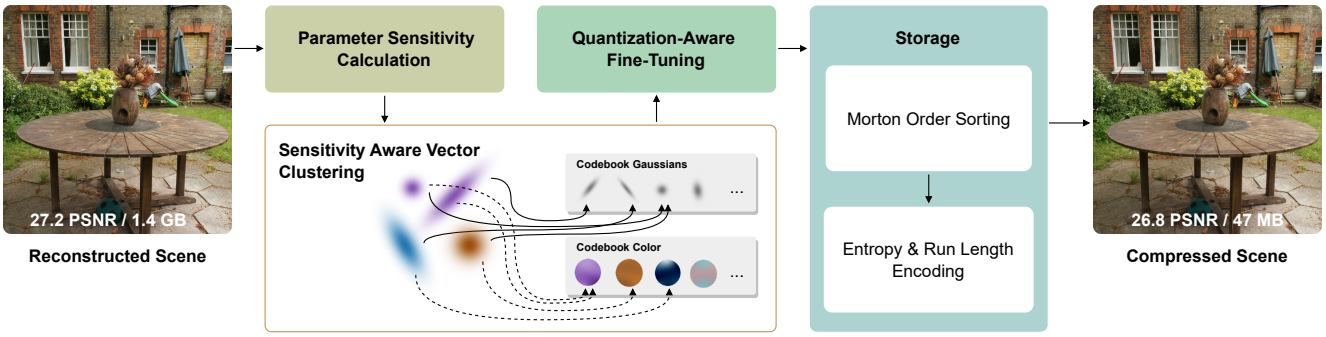

Figure 2. Proposed compression pipeline. Input is an optimized 3D Gaussian scene representation. First, a sensitivity measure is computed for the Gaussian parameters, and color and shape information is compressed into separate codebooks using sensitivity-aware and scale-invariant vector clustering. Next, the compressed scene is fine-tuned on the training images to recover lost information. Finally, the Gaussians are sorted in Morton order and further compressed using entropy and run-length encoding. The shown scene is from [2].

in most cases a lower weight precision is required for model inference than for training (e.g., 8-bit instead of 32-bit). In post-training quantization, the model weights are reduced to a lower bit representation after training. In quantization-aware training, the quantization is simulated during training while operations are performed at full precision to obtain numerically stable gradients. For storage and inference, the low precision weights can then be used with minor effects on the output.

3. Differentiable Gaussian Splatting

Differentiable Gaussian splatting [13] builds upon EWA volume splatting [34] to efficiently compute the projections of 3D Gaussian kernels onto the 2D image plane. On top of that, differentiable rendering is used to optimize the number and parameters of the Gaussian kernels that are used to model the scene.

The final scene representation comprises a set of 3D Gaussians, each described by a covariance matrix $\Sigma \in \mathbb{R}^{3\times3}$ centered at location $x \in \mathbb{R}^3$. The covariance matrix can be parameterized by a rotation matrix R and a scaling matrix S. For independent optimization of R and S, Kerbl *et al.* [13] represent the rotation with a quaternion q and scaling with a vector s, both of which can be converted into their respective matrices. In addition, each Gaussian has its own opacity $\alpha \in [0, 1]$ and a set of spherical harmonics (SH) coefficients to reconstruct a view-dependent color.

The 2D projection of a 3D Gaussian is again a Gaussian with covariance

$$\Sigma' = JW\Sigma W^T J^T, \tag{1}$$

where W is the view transformation matrix and J is the Jacobian of the affine approximation of the projective transformation. This allows to evaluate the 2D color and opacity footprint of each projected Gaussian. A pixel's color C is then computed by blending all N 2D Gaussians contributing to this pixel in sorted order:

$$C = \sum_{i\in N} c_i \alpha_i \prod_{j=1}^{i-1}(1-\alpha_j). \tag{2}$$

Here, c_i and α_i, respectively, are the view-dependent color of a Gaussian and its opacity, modulated by the exponential falloff from the projected Gaussian's center point.

The position x, rotation q, scaling s, opacity α, and SH coefficients of each 3D Gaussian are optimized so that the rendered 2D Gaussians match the training images. For more details on the reconstruction process, we refer to the original paper by Kerbl *et al.* [13].

4. Sensitivity-Aware Scene Compression

We compress a set of optimized 3D Gaussian kernels as follows: First, sensitivity-aware vector clustering is used to cluster the Gaussian appearance and shape parameters into compact codebooks (Sec. 4.1). Second, the clustered and other scene parameters are fine-tuned on the training images to recover information lost due to clustering. We use quantization-aware training in this step to reduce the scene parameters to a lower bit-rate representation (Sec. 4.2). By linearizing the set of 3D Gaussians along a space-filling curve, entropy and run-length encoding can exploit the spatial coherence of Gaussian parameters to further compress the scene (Sec. 4.3). An overview of the proposed compression pipeline is shown in Fig. 2.

4.1. Sensitivity-Aware Vector Clustering

Inspired by volumetric NeRF compression [15, 25], we utilize vector clustering for compressing 3D Gaussian kernels. We use clustering to encode SH coefficients and Gaussian shape features (scale and rotation) into two separate codebooks. As a result, each Gaussian can be compactly encoded via two indices into the codebooks stored alongside.

Parameter Sensitivity: The sensitivity of the reconstruction quality to changes of the Gaussian parameters is not consistent. While a slight change in one parameter of a Gaussian can cause a significant difference in the rendered image, a similar change in another parameter or the same parameter of another Gaussian can have low or no effect.

We define the sensitivity S of image quality to changes in parameter p with respect to the training images as

$$S(p) = \frac{1}{\sum_{i=1}^{N} P_i} \sum_{i=1}^{N} \left| \frac{\partial E_i}{\partial p} \right|. \tag{3}$$

N is the number of images in the training set used for scene reconstruction, and P_i is the number of pixels in image i. E is the total image energy, i.e., the sum of the RGB components over all pixels. The sensitivity of E to changes in p is considered via the gradient of E with respect to p, i.e., a large gradient magnitude indicates high sensitivity to changes in the respective parameter. With this formulation, the sensitivity to every parameter can be computed with a single backward pass over each of the training images.

Sensitivity-aware k-Means: Given a vector $\boldsymbol{x} \in \mathbb{R}^D$, we define its sensitivity as the maximum over its component's sensitivity:

$$S(\boldsymbol{x}) = \max_{d \in [1..D]} S(x_d). \tag{4}$$

The sensitivity measure is then used for sensitivity-aware clustering, i.e., to compute codebooks $\mathbf{C} \in \mathbb{R}^{K \times D}$ with K representatives $\boldsymbol{c_k} \in \mathbb{R}^D$ (so-called centroids).

We define the weighted distance between a vector $\boldsymbol{x}$ and a centroid $\boldsymbol{c_k}$ as

$$\mathcal{D}(\boldsymbol{x}, \boldsymbol{c}_k) = S(\boldsymbol{x}) \|\boldsymbol{x} - \boldsymbol{c}_k\|_2^2. \tag{5}$$

A codebook is then obtained by using k-Means clustering with $\mathcal{D}$ as a similarity measure. The codebooks are initialized randomly with a uniform distribution within the minimum and maximum values of each parameter. The centroids are computed with an iterative update strategy: In each step, the pairwise weighted distances between the vectors $\boldsymbol{x}$ and the codebook vectors $\boldsymbol{c}_k$ are calculated, and each vector is assigned to the centroid to which it has the minimum distance. Each centroid is then updated as

$$\boldsymbol{c}_k = \frac{1}{\sum\limits_{\boldsymbol{x}_i \in A(k)} S(\boldsymbol{x}_i)} \sum_{\boldsymbol{x}_i \in A(k)} S(\boldsymbol{x}_i) \boldsymbol{x}_i \tag{6}$$

Where $A(k)$ is the set of vectors assigned to centroid c_k.

For performance reasons, a batched clustering strategy is used [23]. In each update step, a random subset of vectors is picked and used to compute the update step. Then, the centroids are updated using the moving average with a decay factor λ_d.

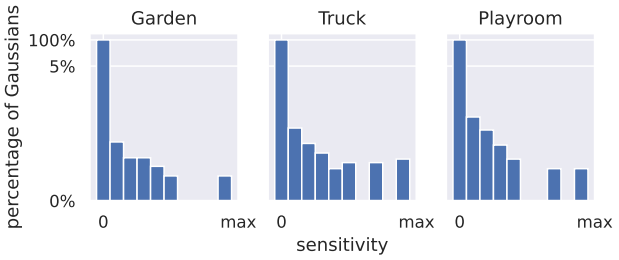

Figure 3. Histograms of maximum sensitivity to changes of SH coefficients for different scenes. Only SH coefficients of a tiny fraction of all Gaussians strongly affect image quality.

Color Compression: Each Gaussian stores SH coefficients to represent the direction-dependent RGB color (e.g., 48 coefficients in [13]). We treat SH coefficients as vectors and compress them into a codebook using sensitivity-aware vector clustering.

For volumetric NeRF models, Li *et al.* [15] have shown that only a small number of voxels contribute significantly to the training images. Thus, they propose to keep the color features that contribute the most and only compress the remaining features with vector clustering. We observe a similar behavior for 3D Gaussians, as shown for some benchmark scenes in Fig. 3. For a small percentage of all SH coefficients ($< 5\%$), the sensitivity measure indicates a high sensitivity towards the image quality. Thus, to keep the introduced rendering error low, we do not consider the SH vectors of Gaussians with a sensitivity higher than a threshold β_c in the clustering process. These vectors are added to the codebook after clustering.

Gaussian Shape Compression: A 3D Gaussian kernel can be parameterized with a rotation matrix R and a scaling vector $\mathbf{s}$. We observe that for typical scenes, the shapes of the Gaussians are highly redundant up to a scaling factor. Thus, we re-parameterize the scaling vector $\mathbf{s} = \eta\hat{\mathbf{s}}$, where $\eta = \|\mathbf{s}\|_2$ is the scalar scaling factor and $\hat{\mathbf{s}} = \eta^{-1}\mathbf{s}$ is the normalized scaling vector. With $\hat{S} = \mathrm{diag}(\hat{\mathbf{s}})$, the normalized covariance matrix is

$$\hat{\Sigma} = (R\hat{S})(R\hat{S})^T = \frac{1}{\eta^2}\Sigma. \tag{7}$$

Clustering is then performed using the normalized covariance matrices, and each Gaussian stores, in addition to a codebook index, the scalar scaling factor η. We compute the sensitivity to each of the matrix entries and perform sensitivity-aware vector quantization to compress them into a codebook. The sensitivity plots for Gaussian shape parameters look mostly similar to the SH plots shown in Fig. 3. As for SH coefficients, Gaussians with a maximum sensitivity over a threshold β_g are not considered for clustering and are added to the codebook.

Note that k-Means clustering of normalized covariance matrices results in covariance matrices that are again normalized. However, due to floating point errors, clustering can lead to non-unit scaling vectors. To counteract this problem, we re-normalize each codebook vector after each update step by dividing it through the trace of the covariance metric. In the appendix, we prove both the normalization preserving properties of k-Means and re-normalization.

After clustering, each codebook entry is decomposed into a rotation and scale parameter using an eigenvalue decomposition. This is required for quantization-aware training since direct optimization of the matrix is not possible[13]. In the final codebook, each matrix's rotation and scaling parameters are encoded via 4 (quaternion) plus 3 (scaling) scalar values.

4.2. Quantization-Aware Fine-Tuning

To regain information that is lost due to parameter quantization, the parameters can be fine-tuned on the training images after compression [15, 30]. To do so, we use the training setup described by Kerbl *et al*. [13]. We optimize for the position, opacity, and scaling factor of each Gaussian as well as the color and Gaussian codebook entries. For the two codebooks, the incoming gradients for each entry are accumulated and then used to update the codebook parameters.

For fine-tuning, we utilize quantization-aware training with Min-Max quantization (k-bit Quantization [18]) to represent the scene parameters with fewer bits. In the forward pass, the quantization of a parameter p is simulated using a rounding operation considering the number of bits and the moving average of each parameter's minimum and maximum values. The backward pass ignores the simulated quantization and calculates the gradient w.r.t. p as without quantization. After training, the parameters can be stored with only b-bit precision (e.g., 8-bit), while the minimum and maximum values required for re-scaling are stored at full precision (e.g., 32-bit float).

Quantization of opacity is applied after the sigmoid activation function. Quantization of the scaling and rotation vector is applied before the respective normalization step. For the scale factor parameter, the quantization is applied before the activation (exponential function) to allow for a fine-grained representation of small Gaussians without losing the ability to model large ones. We quantize all Gaussian parameters despite position to an 8-bit representation with the Min-Max scheme. 16-bit float quantization is used for position, as a further reduction decreases the reconstruction quality considerably.

4.3. Entropy Encoding

After quantization-aware fine-tuning, the compressed scene representation consists of a set of Gaussians and the codebooks storing SH coefficients and shape parameters. Indices into the codebooks are stored as 32-bit unsigned integers.

The data is then compressed using DEFLATE [5], which utilizes a combination of the LZ77 [33] algorithm and Huffman coding. In the reconstructed scenes, many features, such as color, scaling factor, and position, are spatially coherent. By ordering the Gaussians according to their positions along a Z-order curve in Morton order, the coherence can be exploited and the effectivity of run-length encoding (LZ77) can be improved. The effect on the compressed file size is analyzed in the ablation study in Sec. 6.4. Note that entropy encoding reduces the two codebook indices to their required bit-length according to the codebook sizes.

5. Novel View Rendering

Kerbl *et al*. [13] propose a software rasterizer for differentiable rendering and novel view synthesis. To render 3D Gaussian scenes fast especially on low-power GPUs, our novel view renderer utilizes hardware rasterization.

Preprocess: In a compute pre-pass, Gaussians whose 99% confidence interval does not intersect the view frustum after projection are discarded. For the remaining Gaussians, the direction-dependent color is computed with the SH coefficients. The color, the Gaussian's opacity, projected screen-space position, and covariance values are stored in an atomic linear-append buffer. The covariance values indicate the orientation and size of the 2D Gaussian into which a 3D Gaussian projects under the current viewing transformation [34]. As in [13], Gaussians are then depth-sorted to enable order-dependent blending. We use the Onesweep sorting algorithm by Adinets and Merrill [1] to sort the Gaussians directly on the GPU. Due to its consistent performance, the implementation is well suited for embedding into real-time applications.

Rendering: Gaussians are finally rendered in sorted order via GPU rasterization. For each Gaussian, one planar quad (a so-called splat) consisting of two triangles is rendered. A vertex shader computes the screen space vertex positions of each splat from the 2D covariance information. The size of a splat is set such that it covers the 99% confidence interval of the projected Gaussian. The vertex shader simply outputs the color computed in the pre-pass and the 2D splat center as input to the pixel shader. The pixel shader then discards fragments outside the 99% confidence interval. All remaining fragments use their distance to the splat center to compute the exponential color and opacity falloff and blend their final colors into the framebuffer.

6. Experiments

6.1. Datasets

We evaluate our compression and rendering method on the **Mip-Nerf360[2]** indoor and outdoor scenes, two scenes

Method	3D Gaussian Splatting				Ours				
Dataset	PSNR ↑	SSIM ↑	LPIPS ↓	SIZE ↓	PSNR ↑	SSIM ↑	LPIPS ↓	SIZE ↓	Compression Ratio ↑
Synthetic-NeRF [16]	33.21	0.969	0.031	69.89	32.936	0.967	0.033	3.68	19.17
Mip-NeRF360 [2]	27.21	0.815	0.214	795.26	26.981	0.801	0.238	28.80	26.23
Tanks&Temples [14]	23.36	0.841	0.183	421.90	23.324	0.832	0.194	17.28	23.26
Deep Blending [9]	29.41	0.903	0.243	703.77	29.381	0.898	0.253	25.30	27.81
average*	26.58	0.853	0.213	640.31	26.560	0.844	0.238	23.73	25.77

Table 1. Quantitative comparison to 3D Gaussian Splatting. Size is measured in Megabytes. *Synthetic scenes are excluded.

from the **Tanks&Temples[14]** and **Deep Blending [9]** dataset, and **NeRF-Synthetic[16]**. For Mip-Nerf360, Tanks&Temples and Deep Blending the reconstructions from Kerbl *et al*. [13] were used. We generated the 3D Gaussian representation for NeRF-Synthetic ourselves.

6.2. Implementation Details

We use a decay factor $\lambda_d = 0.8$ for batched clustering with 800 update steps for the Gaussians and 100 for the SH coefficients. A batch size of 2^{18} is used for the color features, and 2^{20} for the Gaussian parameters. We use 4096 as the default codebook size in all our experiments and set $\beta_c = 6 \cdot 10^{-7}$ and $\beta_g = 3 \cdot 10^{-6}$. We perform 5000 optimization steps of quantization-aware fine-tuning.

The renderer is implemented with the WebGPU graphics API in the Rust programming language. Thus, it can run in a modern web browser on a large variety of devices. More details about the implementation can be found in the supplementary material. The source code will be released upon acceptance.

6.3. Results

We use the scenes reconstructed by 3D Gaussian Splatting [13] and compress them using the proposed method. For all scenes, we evaluate the PSNR, SSIM, and LPIPS [31] before and after compression. Tab. 1 shows the results for different datasets.

Our compression method achieves a compression ratio of up to 31× with an average of 26× at the indiscernible loss of quality (0.23 PSNR on average) for real-world scenes. Here, it should be noted that a difference of 0.5 PSNR is considered indistinguishable for the human eye [22]. For some of the scenes, Fig. 5 compares training images to the renderings of the uncompressed and compressed scenes. Fig. 4 shows close-up views of uncompressed and compressed synthetic scenes. More comparisons and results are given in the supplementary material.

Image Quality Loss Fig. 5 shows that it is almost impossible to spot the difference between the uncompressed and the compressed scenes. We also analyze the images from all test sets with the largest drop in PSRN. The image which could be reconstructed least accurately is shown in Fig. 6. We observe that the loss is mainly due to very subtle color shifts below what can be perceived by the human eye.

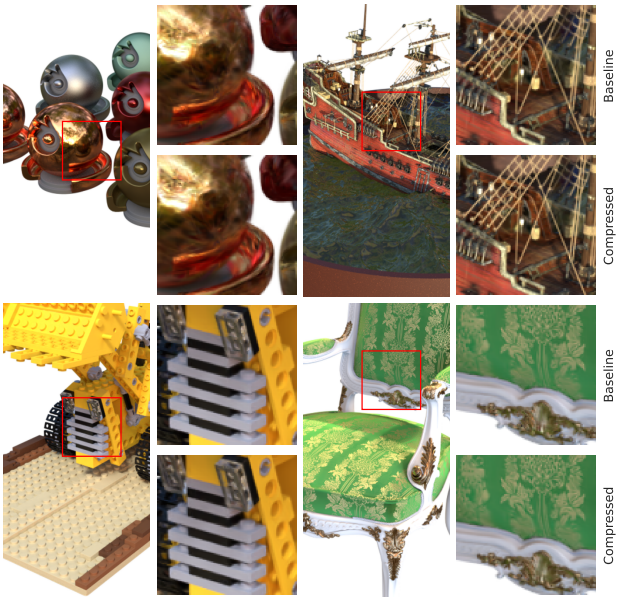

Figure 4. 3D Gaussian splatting of synthetic scenes [16]. Uncompressed (Baseline) vs. compressed scene.

Compression Runtime The compression process takes about 5-6 minutes and increases the reconstruction time by roughly 10%. The timings of each individual steps are given in the supplementary material.

Rendering Times We see a significant increase of up to a factor of 4× in rendering speed (see Tab. 2). Roughly a 2x increase can be attributed to the compressed data's reduced bandwidth requirements, hinting at the software rasterizer's memory-bound performance by Kerbl et al. [13]. The additional speedup is achieved by the hardware rasterization-based renderer, which pays off on low- and high-end GPUs. Timings of the different rendering stages are given in the supplementary material.

6.4. Ablation Study

In a number of experiments we evaluate the components of our compression pipeline. This includes a detailed analysis of the influence of the hyper-parameters.

Loss Contribution Tab. 3 indicates that the most sig-

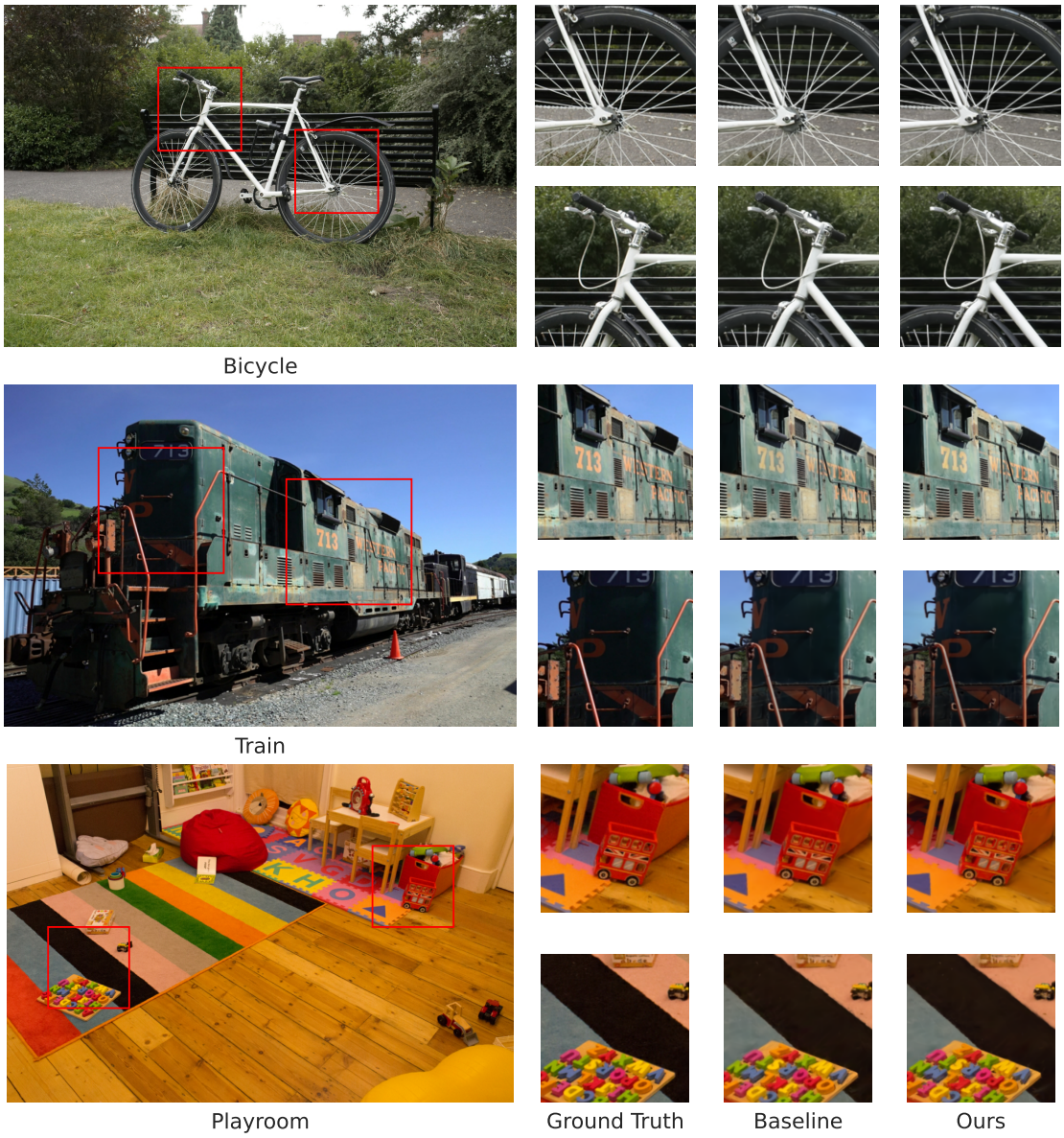

Figure 5. Ground truth images from the test set, results of Kerbl et al. [13] (Baseline), results using the compressed representation (Ours).

Figure 6. Test image with the highest drop in PSNR in all scenes used in this work. d) Per pixel mean absolute error between Kerbl et al. [13] b) and our approach c).

		NVIDIA RTX A5000	NVIDIA RTX 3070M	Intel UHD Graphics 11	AMD Radeon R9 380
Bicycle	Kerbl et al. [13]	93	54	-	-
	Ours	215	134	9	41
	Compressed	321	211	16	83
Bonsai	Kerbl et al. [13]	184	122	-	-
	Ours	414	296	23	76
	Compressed	502	380	28	128

Table 2. Rendering performance at 1080p resolution in frames per second, averaged over all training images. Bicycle consists of 6.1 million 3D Gaussians, Bonsai of 1.2 million 3D Gaussians.

	PSNR ↑	SSIM ↑	LPIPS ↓	SIZE ↓
baseline	27.179	0.861	0.115	1379.99
+ Pruning	27.083	0.856	0.118	1217.25
+ Color Clustering	25.941	0.818	0.178	278.41
+ Gaussian Clustering	25.781	0.811	0.186	164.15
+ QA Finetune	26.746	0.844	0.144	86.69
+ Encode	26.746	0.844	0.144	58.40
+ Morton Order	26.746	0.844	0.144	46.57

Table 3. Losses introduced and regained by individual stages of the compression pipeline. Experiments were performed with the garden scene from Mip-Nerf360[2]

nificant loss increase comes from the compression of the SH coefficients, which, on the other hand, gives the highest memory reduction. Quantization of shape parameters can additionally reduce the memory by about 60%, only introducing a slight loss in image quality. Quantization-aware fine-tuning can regain much of the information that is lost due to quantization and further reduces the memory by about 50%. Entropy and run length encoding in combination with Morton order layout saves an additional 50% of the memory.

Codebook Sizes SH coefficients and Gaussian shape parameters are compressed into codebooks of predefined sizes. Tab. 3 shows the effects of different codebook sizes on image quality. Errors were averaged over all test images, with the difference to the maximum error given in brackets. It can be seen that the codebook size has little effect on the average reconstruction error, independent of the scene. Nevertheless, larger codebooks reduce the maximum error with only minimal memory overhead.

Sensitivity Thresholds The sensitivity thresholds β^c and β^g are used to decide whether to consider SH coefficients and shape parameters for clustering. They offer a trade-off between quality and compression rate. The influence of these values is analyzed in Tab. 5, showing in particular the sensitivity of image quality to the quantization of SH coefficients.

6.5. Limitations

As the main limitation for making the proposed compression and rendering pipeline even more powerful, we see the current inability to aggressively compress the Gaussians' positions in 3D space. We performed experiments where positions were quantized to a lattice structure, and we even embedded these positional constraints into the Gaussian splatting training process. Unfortunately, we were not able to further compress the positions without introducing a significant error in the rendering process.

		PSNR ↑	SSIM ↑	LPIPS ↓	SIZE ↓
Color	1024	26.95(-0.67)	0.80(-0.02)	0.24(+0.03)	28.47
	2048	26.95(-0.62)	0.80(-0.02)	0.24(+0.03)	28.65
	4096	26.98(-0.63)	0.80(-0.02)	0.24(+0.03)	28.80
	8192	27.00(-0.58)	0.80(-0.02)	0.24(-0.03)	28.92
Gaussian	1024	26.95(-0.79)	0.80(-0.02)	0.24(+0.03)	28.14
	2048	26.97(-0.80)	0.80(-0.02)	0.24(+0.03)	28.45
	4096	26.98(-0.63)	0.80(-0.02)	0.24(+0.03)	28.80
	8192	26.97(-0.60)	0.80(-0.02)	0.24(+0.03)	29.06

Table 4. Average reconstruction error over the test images for different codebook sizes, including the maximum deviation from the baseline ($+/-$). Rows marked grey indicate the default configurations. Experiments were performed on the Mip-Nerf360[2] dataset.

		PSNR ↑	SSIM ↑	LPIPS ↓	SIZE ↓
baseline		26.976	0.801	0.238	28.80
β_c	$6.0 \cdot 10^{-8}$	27.22(−0.25)	0.81(−0.00)	0.22(+0.00)	56.50
	$3.0 \cdot 10^{-7}$	27.09(−0.41)	0.80(−0.01)	0.23(+0.02)	33.00
	$6.0 \cdot 10^{-7}$	26.98(−0.63)	0.80(−0.02)	0.24(+0.03)	28.80
	$1.2 \cdot 10^{-6}$	26.87(−0.75)	0.80(−0.02)	0.24(+0.04)	27.02
	$6.0 \cdot 10^{-6}$	26.74(−0.94)	0.80(−0.02)	0.25(+0.04)	25.97
	-	26.55(−1.47)	0.79(−0.03)	0.25(+0.05)	25.65
β_g	$3.0 \cdot 10^{-7}$	27.05(−0.41)	0.80(−0.01)	0.23(+0.03)	33.90
	$1.5 \cdot 10^{-6}$	27.00(−0.62)	0.80(−0.02)	0.24(+0.03)	29.95
	$3.0 \cdot 10^{-6}$	26.98(−0.63)	0.80(−0.02)	0.24(+0.03)	28.80
	$6.0 \cdot 10^{-6}$	26.91(−0.77)	0.80(−0.02)	0.24(+0.03)	28.08
	$3.0 \cdot 10^{-5}$	26.86(−0.72)	0.80(−0.02)	0.24(+0.04)	27.30
	-	26.80(−0.83)	0.80(−0.02)	0.25(+0.04)	27.10

Table 5. Sensitivity threshold ablation study. β^c and β^g are the sensitivity thresholds for controlling which SH vectors and shape parameters are clustered. The average error and in brackets the maximum deviation from the baseline are reported. The last row shows the results when no threshold is considered. The rows marked grey are the default configurations. Experiments were performed with Mip-Nerf360 [2] dataset.

7. Conclusion

We have introduced a novel compression and rendering pipeline for 3D Gaussians with color and shape parameters, achieving compression rates of up to 31× and up to a 4× increase in rendering speed. Our experiments with different datasets have shown that the compression introduces an indiscernible loss in image quality. The compressed data can be streamed over networks and rendered on low-power devices, making it suitable for mobile VR/AR applications and games. In the future, we aim to explore new approaches for reducing the memory footprint during the training phase, and additionally compressing positional information end-to-end. We also believe that 3D Gaussian splatting has the

potential for reconstructing volumetric scenes, and we will investigate advanced options for compressing and rendering the optimized representations.

References

[1] Andy Adinets and Duane Merrill. Onesweep: A Faster Least Significant Digit Radix Sort for GPUs. *arXiv preprint arXiv:2206.01784*, 2022. 5

[2] Jonathan T. Barron, Ben Mildenhall, Dor Verbin, Pratul P. Srinivasan, and Peter Hedman. Mip-NeRF 360: Unbounded Anti-Aliased Neural Radiance Fields. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5460–5469, New Orleans, LA, USA, 2022. IEEE. 3, 5, 6, 8

[3] Anpei Chen, Zexiang Xu, Andreas Geiger, Jingyi Yu, and Hao Su. TensoRF: Tensorial Radiance Fields. In *Computer Vision – ECCV 2022*, pages 333–350, Cham, 2022. Springer Nature Switzerland. 1, 2

[4] Chenxi Lola Deng and Enzo Tartaglione. Compressing Explicit Voxel Grid Representations: fast NeRFs become also small. In *2023 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pages 1236–1245, Waikoloa, HI, USA, 2023. IEEE. 2

[5] L. Peter Deutsch. DEFLATE Compressed Data Format Specification version 1.3, 1996. Issue: 1951 Num Pages: 17 Series: Request for Comments Published: RFC 1951. 5

[6] Sara Fridovich-Keil, Alex Yu, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. Plenoxels: Radiance Fields without Neural Networks. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5491–5500, New Orleans, LA, USA, 2022. IEEE. 2

[7] Yiming Gao, Yan-Pei Cao, and Ying Shan. SurfelNeRF: Neural Surfel Radiance Fields for Online Photorealistic Reconstruction of Indoor Scenes. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 108–118, Vancouver, BC, Canada, 2023. IEEE. 2

[8] Cameron Gordon, Shin-Fang Chng, Lachlan MacDonald, and Simon Lucey. On Quantizing Implicit Neural Representations. In *2023 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pages 341–350, Waikoloa, HI, USA, 2023. IEEE. 2

[9] 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 (Proc. SIGGRAPH Asia)*, 37(6):257:1–257:15, 2018. Publisher: ACM. 6

[10] 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

[11] 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

[12] Seungyeop Kang and Sungjoo Yoo. TernaryNeRF: Quantizing Voxel Grid-based NeRF Models. In *2022 IEEE International Workshop on Rapid System Prototyping (RSP)*, pages 8–14, Shanghai, China, 2022. IEEE. 2

[13] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkuehler, and George Drettakis. 3D Gaussian Splatting for Real-Time Radiance Field Rendering. *ACM Trans. Graph.*, 42(4), 2023. Place: New York, NY, USA Publisher: Association for Computing Machinery. 1, 2, 3, 4, 5, 6, 7, 8

[14] 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. Publisher: ACM New York, NY, USA. 6

[15] Lingzhi Li, Zhen Shen, Zhongshu Wang, Li Shen, and Liefeng Bo. Compressing Volumetric Radiance Fields to 1 MB. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4222–4231, Vancouver, BC, Canada, 2023. IEEE. 1, 2, 3, 4, 5

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

[17] 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. Publisher: ACM New York, NY, USA. 1, 2

[18] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks. In *Computer Vision – ECCV 2016*, pages 525–542, Cham, 2016. Springer International Publishing. 2, 5

[19] Christian Reiser, Rick Szeliski, Dor Verbin, Pratul Srinivasan, Ben Mildenhall, Andreas Geiger, Jon Barron, and Peter Hedman. Merf: Memory-efficient radiance fields for real-time view synthesis in unbounded scenes. *ACM Transactions on Graphics (TOG)*, 42(4):1–12, 2023. Publisher: ACM New York, NY, USA. 2

[20] Daniel Rho, Byeonghyeon Lee, Seungtae Nam, Joo Chan Lee, Jong Hwan Ko, and Eunbyung Park. Masked Wavelet Representation for Compact Neural Radiance Fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 20680–20690, 2023. 1, 2

[21] Darius Rückert, Linus Franke, and Marc Stamminger. ADOP: approximate differentiable one-pixel point rendering. *ACM Trans. Graph.*, 41(4):1–14, 2022. 2

[22] David Salomon and Giovanni Motta. *Handbook of data compression*. Springer Science & Business Media, 2010. 6

[23] D. Sculley. Web-scale k-means clustering. In *Proceedings of the 19th international conference on World wide web*, pages 1177–1178, Raleigh North Carolina USA, 2010. ACM. 4

[24] Cheng Sun, Min Sun, and Hwann-Tzong Chen. Direct Voxel Grid Optimization: Super-fast Convergence for Radiance Fields Reconstruction. In *2022 IEEE/CVF Conference on*

Computer Vision and Pattern Recognition (CVPR), pages 5449–5459, New Orleans, LA, USA, 2022. IEEE. 1, 2

[25] 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*, New York, NY, USA, 2022. Association for Computing Machinery. event-place: Vancouver, BC, Canada. 2, 3

[26] Jiaxiang Tang, Xiaokang Chen, Jingbo Wang, and Gang Zeng. Compressible-composable NeRF via Rank-residual Decomposition. In *Advances in Neural Information Processing Systems*, pages 14798–14809. Curran Associates, Inc., 2022. 2

[27] Krishna Wadhwani and Tamaki Kojima. SqueezeNeRF: Further factorized FastNeRF for memory-efficient inference. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 2716–2724, New Orleans, LA, USA, 2022. IEEE. 2

[28] Sebastian Weiss and Rüdiger Westermann. Differentiable Direct Volume Rendering. In *IEEE Transactions on Visualization and Computer Graphics*, pages 562–572, 2022. Issue: 1. 2

[29] 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. 1, 2

[30] Alex Yu, Ruilong Li, Matthew Tancik, Hao Li, Ren Ng, and Angjoo Kanazawa. PlenOctrees for Real-time Rendering of Neural Radiance Fields. In *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 5732–5741, Montreal, QC, Canada, 2021. IEEE. 5

[31] Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. The Unreasonable Effectiveness of Deep Features as a Perceptual Metric. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 586–595, Salt Lake City, UT, 2018. IEEE. 6

[32] Tianli Zhao, Jiayuan Chen, Cong Leng, and Jian Cheng. TinyNeRF: Towards 100 x Compression of Voxel Radiance Fields. *Proceedings of the AAAI Conference on Artificial Intelligence*, 37(3):3588–3596, 2023. Number: 3. 1, 2

[33] J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Trans. Inform. Theory*, 23(3):337–343, 1977. 5

[34] M. Zwicker, H. Pfister, J. Van Baar, and M. Gross. EWA volume splatting. In *Proceedings Visualization, 2001. VIS '01.*, pages 29–538, San Diego, CA, USA, 2001. IEEE. 3, 5