Compression in 3D Gaussian splatting

Nicolas Bongartz
Universiteit Hasselt
Hasselt, Belgium
nicolas.bongartz@student.uhasselt.be

***Abstract*—The advent of 3D Gaussian splatting has heralded a new domain of SOTA novel view synthesis, characterized by fast training and real-time rendering without significant loss of quality. In contrast, this introduces a memory drawback directly proportional to the scene's complexity. In this paper, an overview of some of the existing compression solutions is provided. Additionally a K-means alternative, K-medoids is proposed and compared.**

I. View synthesis

In graphics, inverse rendering is concerned with the reconstruction of a scene given a set of input observations. Unlike other 3D reconstruction techniques which may require various (sensory) data (e.g. LiDAR, CT), *image-based rendering* (IBR) relies purely on input images to reconstruct (novel) 3D views. Next to the application of triangle mesh [1] and material (e.g. photometric stereo) reconstruction that fits well into the classic rasterization pipeline, the image interpolated scene can have a differing representation together with its own image rendering model. Taking inspiration from physics, the generated scene can be generally defined as a field where $F : \mathbb{R}^n \to \mathbb{R}^n$ with as input a spatial and/or temporal component [2].

Initially, light fields were used to develop novel views as such in [3] where input images are sliced into a 2D part of the 4D light field; here resampling those slices results in new images. With the arrival of *Structure-from-Motion* (SfM) the accessibility of view synthesis technology enlarged, blossoming into communities like those of photogrammetry. Although key-point matching and triangulation are enough to have an accurate estimation of the original 3D environment, the visuals lack complexity (i.e. a sparse point cloud). *Multi-view stereo* (MVS), a grouping name of a large set of IBR techniques utilizing stereoscopy, has shown quality improvements. Since MVS makes use of camera parameters, possibly computed by SfM, it can be considered as a post-processing extension of SfM [4].

Recently neural solutions have made major advances in novel shape representations. Namely making use of neural networks as an estimation of the scene's surface or volume representation facilitates backpropagation to calculate the loss between rendered and ground truth images. Note that it is essential for stochastic gradient descent (SGD) that both the representation and rendering model are differentiable. Survey [5] goes into depth about how machine learning can be inserted into the rendering pipeline to create *neural scene representations and rendering*. Further, they provide logically structured definitions used in the current literature.

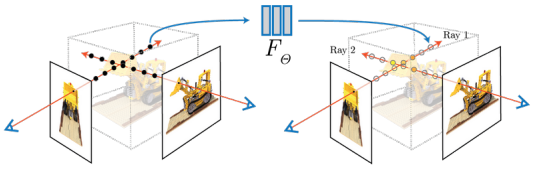

Fig. 1: Visual example of the NeRF rendering process: values of the samples along the ray are queried of the deep neural network (DNN). Image is taken from [7].

A. NeRFs

Because of computation limitations, physics definitions are approximately applied in graphics. So is radiance, the amount of light radiating from a surface in a given direction [8], modelled as a ray (i.e. geometric optics). Such radiance field, describing the light distribution in a scene, can be defined as

$$L : \mathbb{R}^3 \times \mathbb{S}^2 \to \mathbb{R}^n$$

with $(x, y, z) \in \mathbb{R}^3$ a point in space and $(\theta, \phi) \in \mathbb{S}^2$ spherical coordinates stating a direction. The difference between a radiance and a light field is not mathematically defined in graphics. Though [9] distinguishes between both by having the radiance value along a ray not change for a light field.

Neural Radiance Field (NeRF) is the first neural solution to achieve a high-resolution continuous scene from natural images, normally only seen in discrete representations (i.e. triangle meshes, voxel grids) [7]. It does so by encoding an implicit radiance field into a single multilayer perceptron Θ (MLP), hence $F_\Theta : (x, d) \to (c, \sigma)$ with $(x, d) = (x, y, z, \theta, \phi)$ and $(c, \sigma) = (r, g, b, \sigma)$ with σ volume density. Then to render out a scene, 3D sample points are found by ray-marching. These are then fed into Θ to query (c, σ) along the ray which are accumulated using volume rendering techniques. To overcome bias in high-frequency scenes the input data is first mapped to a higher dimension according to [10] ([7] refers to this as positional encoding). The performance is also improved by using hierarchical volume sampling to avoid querying points in non-dense environments.

After the release of NeRF, countless follow-ups have been published. Mip-NeRF 360 [11], an extension, deals with anti-aliasing problems. Plenoxels [12] and InstantNGP [13] on the other hand tackle the large training and rendering times problems regarding vanilla NeRF. Plenoxels replaces the neural network with a voxel grid containing spherical harmonics while InstantNGP uses feature hash maps.

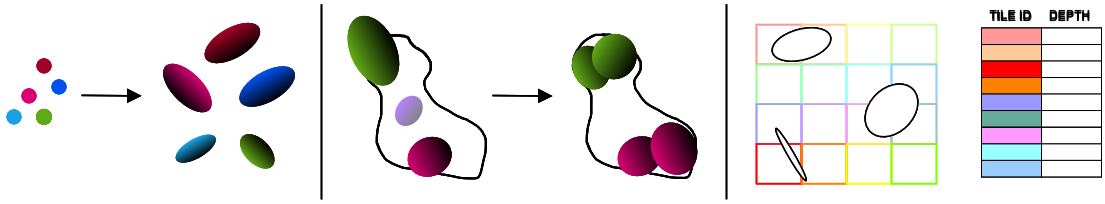

Fig. 2: The three main steps of 3DGS: convert SfM points to 3DG; split, prune and clone 3DGs to fit geometry; render them using a fast tile-based rasterizer.

B. 3D Gaussian splatting

After the build-up of neural networks in IBR, [14] took an alternative route by revisiting point-based rendering methods. To prevent holes and gaps in the geometry, point-based scenes are typically built with primitives larger than a pixel (e.g. ellipsoids) and rely on splatting to produce an image. Moreover, in the case of *3D Gaussians* (3DG) which are differentiable, further optimizations through backpropagation can be made to reduce inconsistencies.

- **Representation** Initially, 3D Gaussians are constructed from a sparse point cloud generated through SfM. These multivariate Gaussian distributions are characterized by a mean μ and covariance matrix $\Sigma \in \mathbb{R}^{k\times k}$ (for 3D $k = 3$). To avoid noisy normal estimation due to SfM, opacity (α) and spherical harmonics (SH) are added for view-dependent colours to each Gaussian to represent an explicit radiance field. Here

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

  represent a single Gaussian probability density function (PDF) (term before e omitted) with Σ^{-1} the inverse covariance matrix. Originally the Gaussian is isotropic where its size is equal to the mean of the distance to the closest three other Gaussian's.
- **Optimization** The process of optimization spans several iterations where SGD is used to compare the rendered images with their corresponding ground truth. The loss function $\mathcal{L}$ is a weighted sum between $\mathcal{L}_1$ and $\mathcal{L}_{D-SSIM}$ (D stands for data)

$$\mathcal{L} = (1-\lambda)\cdot\mathcal{L}_1 + \lambda\cdot\mathcal{L}_{D-SSIM} \tag{2}$$

  where arbitrarily $\lambda = 0.2$. Every 100 iterations the Gaussians are densified and pruned. The latter is to remove either uninfluential points in case $\alpha < \epsilon_\alpha$ or those with large prints in model or view space. By setting α close to zero after $K = 3000$ iterations and letting optimization increase the relevant opacities, camera floaters can be circumvented. Since large view space positional gradients ($< \tau_{pos}$) are the cause of under or over-constructed geometric areas, densification is applied to those Gaussians. In under-construction, the Gaussians are cloned to create new geometry, while in over-construction they are split and scaled with a certain factor $\phi = 1.6$.
  It is important to note that Σ is not directly optimized: gradient descent can lead to so-called non positive semi-definite covariance matrices which have no meaning (i.e. negative variances are undefined). Therefore the underlying scale S and rotation R matrix, represented by $s \in \mathbb{R}^3$ and $q \in \mathbb{R}^4$ being a quaternion, are trained instead. The covariance matrix can be then reconstructed through its eigendecomposition

$$\Sigma = RSS^TR^T \tag{3}$$

  with forms of B^TB ensuring positive semi-definiteness [15]. See [16] for more insight into the geometric properties of the covariance matrix.
- **Differentiable renderer** The renderer consists of a tile-based differentiable rasterizer using CUDA kernels. The screen is divided into 16x16 tiles, launching one thread block per tile. Given the camera parameters, the Gaussians are first culled against the frustum and tiles. The remaining are projected to 2D distributions (i.e. ellipses) in view space. Since this projection is not affine, it has to be approximated using the first-order Taylor expansion [17]:

$$\Sigma' = JW\Sigma W^TJ^T \tag{4}$$

  with J the Jacobian of the approximation.
  For all Gaussian tile intersections a 32-bit number, subdivided into view space depth and tile ID, is added to a list. A fast radixsort [18] then sorts this list. The performance gained by globally sorting instead of per pixel outweighs the barely noticeable introduced artefacts due to overlapping Gaussians. StopThePop [19] follow-up work solves this per-pixel issue efficiently by creating a sorting hierarchy. Each tile is then parallel rendered with its shared memory from front to back applying classic point-based α blending as its image formation model. For every pixel with $\mathcal{N}$ overlapping points, its colour is defined by

$$C = \sum_{i\in\mathcal{N}} c_i\alpha_i\prod_{j=1}^{i-1}(1-\alpha_j) \tag{5}$$

  . Here α_i is the learned opacity multiplied by the evaluation of the Gaussian in (Eq. 1). This results in an exponential increase in transparency as the distance from the mean increases.

The point-based rasterization method for radiance fields proves to be fast (real-time) and flexible, achieving quality similar to and in some cases even better than NeRF. Unfortunately,

this explicit radiance field comes with a significant memory complexity. Where memory in NeRF was primarily confined to the network weights, sometimes as low as 5MB [7], 3DGS requires every Gaussian to personally store all its information. This lowers the expense of querying an MLP but accompanies it with a rise in VRAM utilization. Although the memory consumption is smaller than Plenoxels, there is a lot of room for improvement.

II. Compression in 3DGS

In the following section, an overview of some existing solutions to the memory problem is given. The focus is laid on work building on top of vanilla 3DGS.

A. Model pruning

In neural networks, pruning of a model can be achieved by removing redundant weights, connections, and neurons. Although 3DGS's training step is not built using a neural network, it can be seen as a single layer of Gaussian parameters updated using SGD. Subsequently, pruning neurons involves removing entire Gaussian points, resulting in compression. As outlined in (Sec. I-B), opacity is the key ingredient in determining uninfluential Gaussians in vanilla 3DGS. Nonetheless, this method is not perfect: unnecessary Gaussians remain. Simply making this decision technique more aggressive, can lead to a significant drop in quality [20], [21]. Other pruning possibilities on top of the baseline thus need to be explored. Notice beforehand that opacity and volume are the main visual properties upon which a decision can be made. Generally, Gaussians with less transparency and large volumes are the biggest contributors to the construction of the scene.

One proposed way of pruning is by calculating a *significance score* for each Gaussian and removing those with small scores [20]. The significance, taken over all training views (N) pixels, of a Gaussian g is formulated by LightGaussian as

$$S = \sum_{j}^{N} \sum_{i=0}^{WH} \mathbb{1}(g, r_i) \cdot \alpha \cdot \gamma(\Sigma) \tag{6}$$

with W and H the width and height of the rendered image respectively, r_i the shot ray to the pixel, $\mathbb{1}$ the indicator function with subset all g hit with r_i. The covariance matrix is normalized by γ using the volume of the 90% largest Gaussian, estimating it as a sphere (i.e. $\frac{4}{3}s_0s_1s_2$). The intuition behind this is that the foreground Gaussians are many and densely populated, while the opposite is true for the background Gaussians. This spread of the Gaussians follows directly from the information found in the training images. Normalization thus levels the playing field, avoiding the otherwise biased significance of background Gaussians.

A complementary method involves calculating a *redundancy score* instead [21]. Inspired by level of detail (LOD) [21] identifies resolution-aware spherical regions of interest centred around each Gaussian's mean. The size for each region is chosen based on the closest training viewpoint to its corresponding visible Gaussian. Because the scene around each Gaussian is the most intricate for this closest viewpoint, the redundancy spheres are therefore tactfully placed to ensure maximum diversity and representation. The radius is determined by the number of pixels that observe the Gaussian a_{min} such that $r = \frac{a_{min}\sqrt{3}}{2}$ (with $a_{min}\sqrt{3}$ the diagonal of a cube with face area a_{min}). Finally, a score, equal to the amount of Gaussian intersections, can be assigned for every one of these spheres, creating a redundancy field. K-nn search here can simplify finding possible intersections. Also by scaling these Gaussians to the sphere radius, the intersection between volumes is reduced to a point.
To attribute a score to a Gaussian g, the smallest intersection of g with its surrounding spheres is chosen: the redundancy of g is underestimated. Namely, the Gaussian may be redundant in one region but may be of great importance in another. The scores of the Gaussians are listed: 50% of the Gaussians filtered by $max(\mu + \sigma, 3) < R$, with mean and standard deviation taken of all scores, are pruned.

Another typical technique in model pruning is *learnable (binary) masks* assigned to each neuron, telling something about its connections. In [22] such learnable binary mask strategy is proposed where each Gaussian has a supplemental mask parameter $m \in \mathbb{R}$ updated using SGD. From this parameter a binary mask $M \in \{0, 1\}$ for g can be calculated:

$$M = sg(\mathbb{1}(\sigma(m) > \epsilon) - \sigma(m)) + \sigma(m) \tag{7}$$

with sg the stop gradient function (identity function in the forward pass but blocks the gradient in the backward pass) and σ the sigmoid activation function for m. Notice that gradients from M are zero, therefore straight-through estimator (STE) is applied to bypass these. By multiplying M with scale and opacity, the Gaussian is potentially dismissed during rendering. The loss is then calculated by adding a mean term of all m to the vanilla loss function. During the densification stages, the Gaussians are permanently removed according to M.

Method	Compression proportion (%)
LightGaussian	71
Footprint3D	57
Compact 3D	63

TABLE I: The ablated Gaussian pruning compression proportions of the total memory reduction is taken from [20]–[22] with scenes respectively Room, Playroom and Mip-NeRF360 dataset. Because of a mismatch between scenes, the table is purely for a general estimated overview of how much pruning can contribute to the entire compression pipeline.

B. Colour compression

The majority of memory per Gaussian is attributed to SH, consisting of 81.3% of the total memory. Reduction of the

SH coefficients can significantly improve compression rates. Three existing different methods of partially dealing with this bottleneck are surveyed.

In neural networks *knowledge distillation* is the transferring of expertise from a large teacher model to a smaller student model by calculating the loss between both and applying this $\mathcal{L}_{distil}$ in the total loss function of the student. The student's parameters are therefore updated through imitation of the teacher's model. As noticed by [20] this mechanism can be employed to successfully distil information from the third to the second degree of SH:

$$\mathcal{L}_{distil} = \frac{1}{WH} \sum_{j=0}^{WH} \|C_{teacher}(r_j) - C_{student}(r_j)\|_2^2 \quad (8)$$

with C a scalar representing the pixel's intensity (i.e. 0 - 255) of the rendered image. Due to the limited amount of training views the distillation fails to shift the view-dependent effects to a lower coefficient. Therefore it is proposed to generate synthetic training views as an augmentation through variation of camera position by $p_{synt} = p_{train} + \mathcal{N}(0, \sigma^2)$ with $\mathcal{N}$ a normal distribution.

Redundancy does not only manifest itself in the scene as a whole but also on an individual Gaussian level. Footprint3D [21] has shown how certain Gaussians can effectively be represented using fewer bands of spherical harmonics, thus paving the way for a scene where Gaussians *have variable amount of bands*. They have explored two parallel ways of lowering the degree.

1) The distribution of colour of a Gaussian g over all training views N can be "modelled" using the weighted average colour μ and standard deviation σ:

$$\mu = \frac{\sum_j^N c_j \bar{T}_j}{\sum_j^N \bar{T}_j}, \quad \sigma = \frac{\sum_j^N (c_j - \mu)^2 \bar{T}_j}{\sum_j^N \bar{T}_j} \quad (9)$$

with $\bar{T}_j = \frac{\sum_k^P T_{jk}}{P}$ the average transmittance of all pixels P overlapping g. View dependency becomes evident when the variance of colour (μ) over the N training views high is. In case not, reduce g's SH to band zero consisting solely of the base-RGB colour (replaced by μ).

2) The distance d between colour c_k with $k \in [0, 2]$ (degree of SH) and c_3 is a measure of how good lower degrees can approximate degree three. The weighted average distance to c_3 overall N representing the average approximation accuracy,

$$d_k = \frac{\sum_j^N \|c_{3j} - c_{kj}\|_2 \bar{T}_j}{\sum_j^N \bar{T}_j}, \quad (10)$$

can be compared with a threshold to decide which band could represent the Gaussian. The smallest filtered d_k is chosen.

Compact 3D [22] uses an entirely different representation for colour based on [11] to contract the positions of the Gaussians to a bounded range needed for the usage of hash grids [13]. The view direction d to the Gaussian together with the resulting feature from the contracted position in the hash grid and parameters θ are fed into a tiny *neural field* to get a band 0 SH.

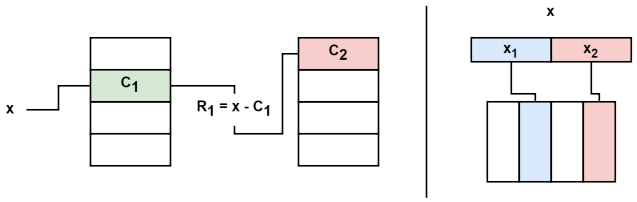

Fig. 3: On the left is a stage two RVQ and on the right PQ of vector x with two components using a shared codebook.

Method	Compression proportion (%)
LightGaussian	12
Footprint3D	33

TABLE II: The ablated SH pruning compression proportion of the total memory reduction is taken from [20], [21] with scenes respectively Room and Mip-NeRF360 dataset. Because of a mismatch between scenes, the table is purely for a general estimated overview of how much colour pruning can contribute to the entire compression pipeline.

C. Vector quantization

Exploiting the fact that 3D Gaussians have shared configurations of parameters (e.g. colour and geometry), a more compact representation of the data can drastically decrease size. A standard way of doing this is through vector quantization (VQ) where the set of all data points $X \in \mathbb{R}^{n \times m}$ is grouped into clusters C such that $X = C_0 \cup ... \cup C_N$; visually this corresponds to a Voronoi-diagram. Every cluster is denoted by a centroid/code, the collection of these is referred to as a codebook. Instead of storing the full data, only the codebook and the position of all $x \in X$ in this codebook (i.e. the indices) are stored. If $|C| \ll |X|$, lossy compression is achieved.
The most common clustering algorithm, also in 3DGS, is K-means (Lloyd–Forgy algorithm) [26]. Considering its simplicity, it offers a lot of room for flexibility. To update a centroid $c \in C$ the mean is taken over all the points x belonging to the cluster of c, after which points are reassigned to the closest centroid. This process repeats for a certain amount of iterations (or till convergence) to minimize

$$\sum_k^{|C|} \sum_i^j ||x_{ki} - c_k||_2^2, \quad (11)$$

the variance within each cluster.

One of the first studies for VQ in 3DGS was done by [23] investigating the different parameters to cluster, codebook sizes and the effect of shared parameter codebooks. CompGS

focused on clustering covariance parameters and colour since clustering positions result in overlapping Gaussians. They noticed that the DC component of colour was the least sensitive with low codebook sizes already contributing to high quality. Rotation and SH remain stagnant after a certain size, while scale is almost proportional to its size. These results hint towards the level of entropy associated with each parameter. Furthermore, they noticed that the quality severely degrades when the clustering happens post-training. Therefore to allow the optimization to compensate for the quantization loss, CompGS renders using quantized parameters every K or so iterations. Since VQ is not differentiable, STE is used to pass the gradients to the non-quantized parameters to update them. This moving of otherwise fixed data points in the clusters enables the ability to update centroids without needing to reassign points to them, reducing thus extra training overhead. The conventional way of dealing with costly reassignments required to update centroids, is mini-batching K-means, which leads to a decrease in accuracy.
SGD thus helps to recover from scene quality loss due to cluster configurations. Alternatively, different clustering techniques can also improve cluster quality. A few of these techniques applied in 3DGS are described.

Residual vector quantization (RVQ) for the generation of a geometry codebook, is opted by 3D [24] in the last $1K$ training iterations. RVQ tries to increase precision by adopting sequential stages of VQ (Fig. 3). Each stage apart from the first, takes as input the difference between the non-quantized and quantized parameters. The original vector can be then approximated through $x'_i = \sum_{j=0}^{k} c_{ij}$ with j the stage number and k the number of stages.
Another way of enhancing the precision of plain K-means is through *product vector quantization* (PVQ) (Fig 3) that clusters the individual components of a vector separately. Through this process, PVQ exponentially increases the number of potential data representations for a vector $x \in \mathbb{R}^n$, scaling with the dimensionality of the data (e.g. for k clusters, k^n choices applies). Footprint3D [21] implements codebook sizes of 256 entries for each scalar, hence 1-byte indices can be used. This allows for the unusual quantization of opacity and brings the indices' memory footprint, the bottleneck of PVQ, to a minimum.

K-means simplicity can also be exploited. Compressed 3D [25] introduces a *sensitivity term* derived from the training views. For a parameter x with components x_k this term equals to

$$S(x) = \max_{k \in [0..K]} \frac{1}{\sum_{j=0}^{N} P_j} \sum_{j=0}^{N} |\frac{\partial E_i}{\partial x_k}|, \tag{12}$$

with P the pixels in view j and E the sum of RGB colours of all pixels of an optimized image. Centroid updates (Eq. 11) are modified to a weighted average using $S(x)$: they are thus likely to stir more in the direction of highly sensitive

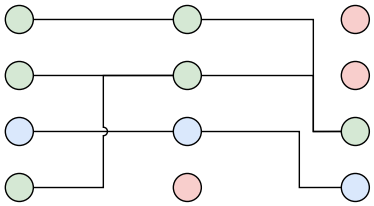

Fig. 4: Visualisation of $m = 3$ with $r = 4$ and $k = 2$; red points are unmapped vertices. Notice that a vertice can have multiple incoming but only one outgoing connection.

Gaussians instead of the centre of the cluster. To accelerate this effect certain significant Gaussians, proceeding a sensitivity threshold, are immediately placed in the codebook. As noticed by [23] the scene has a lot of entropy due to the scale parameter, complicating the clustering of geometry. Compressed3D, therefore, stores the scale individually and clusters instead the normalized covariance matrix $\hat{\Sigma} = \frac{1}{\|s\|_2^2}\Sigma$ containing purely the rotation factor (done for practical reasons regarding duality memory and time complexity of Σ calculation on the GPU).

D. Scalar quantization

A lot of models can be represented using fewer bits as a post-processing effect. As proven by [20], [21] half-precision 16-bit has hardly any impact on scene quality. Strongly asymmetric min-max 8-bit quantization on every parameter except for position has been demonstrated to be effective by [22], [23], [25]. Here a $f32$ can be quantized to a $ui8$ in range $[0, 255]$ by making use of

$$Q(x) = \lfloor x \cdot S - Z \rfloor \tag{13}$$

with $S = \frac{2^8-1}{\max_x - \min_x}$ and $Z = \min_x \cdot S$.

E. Entropy encoding

Famous mathematician Claude Shannon defined in 1948 the concept of entropy in information theory as

$$H(S) = \sum_{s \in S} p_s \log_2 \frac{1}{p_s} \tag{14}$$

with S the set of all states and p_s the probability of that state [27]. Following from (Eq. 14) states with high probability have less entropy, minimizing the amount of bits needed for that state. This idea of statistical entropy is at the basis of every prefix code in lossless compression, successfully diminishing storage size on disk.
For 3DGS encodings like *Huffman coding* and *run-length coding* can be employed for the resulting (non-)quantized parameters and clustering indices.

F. Future work

While remaining excellent in evaluation, work in 3DGS compression has shown impressive compression rates of x25 and more, adopting various techniques. Nevertheless one of the big bottlenecks remains the Gaussian positions where half-precision continues to be the only available compression with acceptable results.

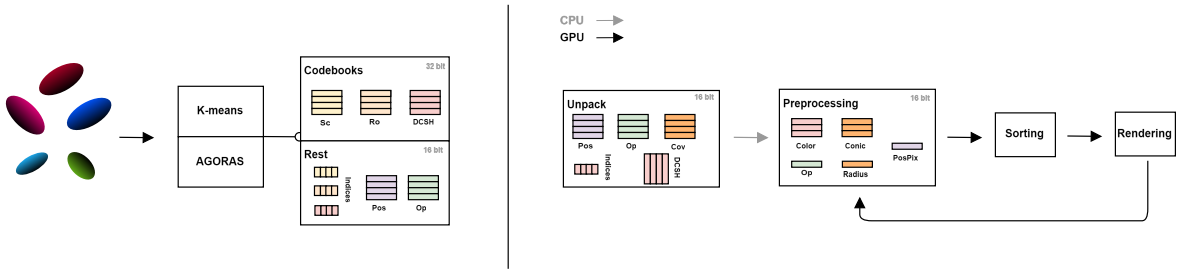

Fig. 5: An optimized scene as input is clustered in scale, rotation and diffuse/SH codebooks. Here position (mean), opacity and indices are stored as 32 bits. During the unpacking phase, the covariance matrix is calculated as a one-time operation on the CPU, unfortunately increasing VRAM usage. Every frame the necessary information for rendering is pre-calculated for the quad-based rendering step, reducing the need to repeat this for every quad vertice.

III. K-MEDOIDS

As already mentioned K-means is the most widely used clustering algorithm because of its simplicity and efficiency (taking the NP-hardness of VQ into account). However, the different techniques in clustering are numerous, going from different initializations (e.g. K-means++), assignments of centroids (e.g. EMA) and even variations in the same algorithm (e.g. Hartigan-Wong and MacQueen method). For huge datasets, like 3DGS, the lack of efficiency tho plagues the choice of algorithms.

One promising approach would be K-medoids that mitigates outliers and noise through partitioning around medoids (PAM [28]) (i.e. real data points). Augmented graph of ordered random samples (AGORAS [6]) overcomes the high complexity of PAM by isolating clusters taken from subsequently drawn random sample sets, hence independent of the data set size. In this way AGORAS outperforms PAM and even CLARANS [29] in speed, though this estimation of clusters comes with a quality drop.

AGORAS identifies k clusters given m random sample sets of equal cardinality r, by way of calculating the pairwise distance between all points of consecutive sets S. For a point $s_i \in S$ the closest from $s_{i+1} \in S$ is chosen. This can be seen as constructing k trees (with roots in s_m) via the described process repeating itself for subsequent sample sets, ignoring unmapped vertices (Fig. 4). To let this process converge quickly they initialise r with $k \ln k + \gamma k$ based on the similarity between a sufficient sample set (i.e. the set containing a minimum of one sample for every cluster) and the coupon collector's problem in probability theory [35]. Upon failing to establish k trees, r is then resized via test-and-change formulas. Following this the process then restarts. For a full pseudo code overview see [6].

Since **K-medoids** (e.g. AGORAS) is not influenced by outliers within clusters as it does not involve averaging, it has the potential to **achieve higher quality cluster configurations** than K-means.

IV. IMPLEMENTATION

In this paper an AGORAS implementation of 3DGS is compared with a vanilla mini-batched K-means, taking both an optimized scene as input. Also given the role compression plays in extending features to less powerful devices, the **easiness of integration in a web environment is tested**.

A. Clustering

The programming setup is done in PyTorch [30] having access to GPU-accelerated functionality through CUDA. This is reinforced with custom CUDA kernels, handy for specific combinations of distance calculations otherwise requiring multiple asynchronous GPU calls.

For the sake of comparison, both the K-means and AGORAS implementation have the same codebook sizes, namely 4096 [23]. As well proposed by [23] the DH and SH components are combined into one shared codebook; rotation and scale are clustered separately. K-means is initialized with random centroids with 100 and 3000 iterations respectively for colour and geometry. For performance issues, the K-means implementation is mini-batched with a batch size of 100000. A similar idea is suggested for AGORAS: based on the observation that the convergence to k clusters takes a longer time with large k, an empirically determined margin hyperparameter $\lambda = \frac{k}{40}$ is introduced. In this way if the number of roots in s_m falls within this positive and small (otherwise endangering cluster quality) margin, the algorithm ends with the first k clusters chosen instead of restarting. Regarding the number of sample sets m, for geometry 50 while for the high dimensional colour 5 is chosen.

Mean, opacity and the indices are stored at 16-bit. Since the codebooks are only a small percentage of the total size, they can be stored at full precision (32-bit). The data is finally stored using Numpy's [31] ZIP-DEFLATE compression [25].

B. Rendering

The rendering pipeline is implemented in the yet experimental new rendering API WebGPU [32]. Being a successor to WebGL it differs by being more stateless and robust through the concepts of immutable pipelines and command buffers. Additionally, it comes with dedicated built-in compute shaders permitting operations like GPU sorting on every frame thus reducing image errors.

The Gaussian data (codebooks, indices, means and opacities)

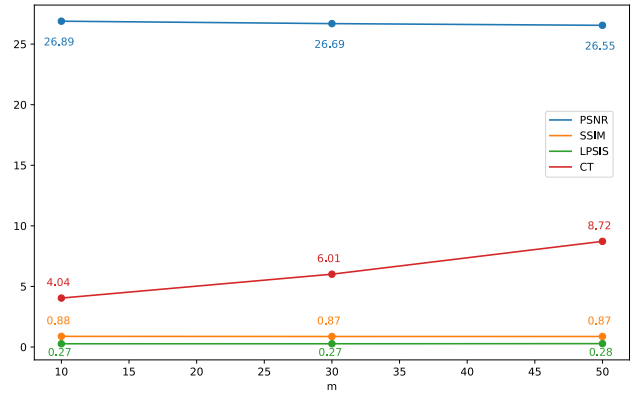

Fig. 6: The graph that shows the non-existent correlation between small increasing steps of m and improvement in quality, collected from the *Bonsai* scene, is taken over an average of three measurements per value of m. For the sake of further investigations, all clustering is done with $m = 50$.

are first unpacked on the CPU. Because half-precision is not native in Javascript as well in WebGPU (a lot GPU's do not yet support it), workarounds in the form of using other supported data types are needed (for Javascript see [33] and for WebGPU see [34]). The covariance matrix is here calculated as a one-time operation. During the pre-processing step, the Gaussians are splatted, culled and colour-evaluated only once for every frame, after which the data is globally sorted based on depth. Again the results are stored as much as possible at half-precision. Contrary to the base implementation, a quad renderer is used to benefit from the vertex and fragment shader capabilities. For each Gaussian, a bounding box is defined by the largest of the ellipse axes where

$$\alpha = \sqrt{\lambda_1} \quad \beta = \sqrt{\lambda_2}, \tag{15}$$

with λ the eigenwaarden of the splatted covariance matrix. The Gaussian is then drawn onto this box by evaluating (Eq. 1), discarding invalid fragments.

V. Results

Although AGORAS is profiled as being independent of the data size n, both the initial sample size r and the recommended minimum number of sample sets m are subjected to the number of clusters. And because the codebook size normally enlarges with bigger data sets as a means of representing the data more accurately[1], the complexity of the algorithm is implicitly influenced by n. For quality findings of clusters the minimum value of m is recommended to be equal to the initial sample size, underpinned by a meta version of the coupon collectors problem [6]. But it goes without saying that larger numbers of m are able to cover more ground and therefore increase quality.

[1]Especially since overfitting the number of clusters is not a concern for 3DGS due to the explicit radiance field.

However m is also proportional to the clustering time. Theoretical the probability of m sufficient sample sets reduces with greater m (luckily this has a minor effect because of large $\mathbb{P}\{S\}$ [36]), but mainly the costly practical computations have a severe impact. Therefore the minimum value for $m = 4096 \ln 4096 + \gamma 4096 \approx 36433$ for 3DGS, is far from feasible. As displayed in Fig. 6, even small values of m already require extensive clustering times (CT).

Method	SSIM	PSNR	LPIPS
3DGS	0.92	30.41	0.19
K-means	0.87	26.74	0.24
AGORAS	0.85	25.49	0.25

TABLE III: Quality measures of *Mip-NeRF360 indoor* scenes trained on 30K iterations. All further experiments used this set and were run on an NVIDIA GeForce GTX 1050.

As expected both the clustering algorithms perform way worse than plain 3DGS because of the lack of any fine-tuning on the training views using SGD. But more importantly, the AGORAS implementation has more quality errors than K-means contradicting the hypothesis. This is most likely due to the small but necessary value of m not permitting the algorithm to perform at its full extent.

Dataset	K. CT	A. CT	Comp. ratio
Room	16.33	12.96	19.36
Counter	14.55	9.25	19.04
Kitchen	17.98	10.59	19.44
Bonsai	14.63	8.56	19.10
Average	15.87	10.34	19.24

TABLE IV: The K-means and AGORAS clustering times are in minutes and the compression ratio is the fraction between the original and compressed size.

Seeing as both algorithm's time complexity is independent of the data set size (i.e. mini-batched, uniform samples) the clustering times closely resemble each other. The fluctuations can be attributed to computations or in the case of AGORAS the amount of restarts. Entropy encoding is responsible for the small variety of compression ratios.

Clustering	Quantization	Entropy
0.93	0.06	0.01

TABLE V: The proportions of the total memory reductions are in %. The differences in percentages are partially attributed to the applying order of the compression methods.

Conclusions

Despite the fact that AGORAS underachieves K-means because of the necessary small value of m, other K-medoids alternatives could potentially deliver results confirming the hypothesis. Nevertheless, it shows that with simple compression schemes already x19 ratios can be accomplished and fairly easily integrated into a web environment. That said the

overview in this paper has expressed a bright (already) present and future for compression, quality and FPS in 3DGS.

ACKNOWLEDGMENTS

I'd like to thank my promoter dr. Jeroen Put for proofreading of the paper and helpful tips throughout the process.

REFERENCES

[1] Chen, Z. A Review of Deep Learning-Powered Mesh Reconstruction Methods. (2023)

[2] Xie, Y., Takikawa, T., Saito, S., Litany, O., Yan, S., Khan, N., Tombari, F., Tompkin, J., Sitzmann, V. & Sridhar, S. Neural Fields in Visual Computing and Beyond. (2022)

[3] Levoy, M. & Hanrahan, P. Light field rendering. *Proceedings Of The 23rd Annual Conference On Computer Graphics And Interactive Techniques*. pp. 31-42 (1996), https://doi.org/10.1145/237170.237199

[4] Furukawa, Y. & Hernández, C. Multi-View Stereo: A Tutorial. *Foundations And Trends® In Computer Graphics And Vision*. **9**, 1-148 (2015), http://dx.doi.org/10.1561/0600000052

[5] Tewari, A., Thies, J., Mildenhall, B., Srinivasan, P., Tretschk, E., Wang, Y., Lassner, C., Sitzmann, V., Martin-Brualla, R., Lombardi, S., Simon, T., Theobalt, C., Niessner, M., Barron, J., Wetzstein, G., Zollhoefer, M. & Golyanik, V. Advances in Neural Rendering. (2022)

[6] Rangel, E., Hendrix, W., Agrawal, A., Wei-Liao & Choudhary, A. AGORAS: A Fast Algorithm for Estimating Medoids in Large Datasets. *Procedia Computer Science*. **80** pp. 1159-1169 (2016), https://www.sciencedirect.com/science/article/pii/S1877050916309309, International Conference on Computational Science 2016, ICCS 2016, 6-8 June 2016, San Diego, California, USA

[7] Mildenhall, B., Srinivasan, P., Tancik, M., Barron, J., Ramamoorthi, R. & Ng, R. NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis. *ECCV*. (2020)

[8] Mittal, A. Neural Radiance Fields: Past, Present, and Future. (2024)

[9] Wang, H., Ren, J., Huang, Z., Olszewski, K., Chai, M., Fu, Y. & Tulyakov, S. R2L: Distilling Neural Radiance Field to Neural Light Field for Efficient Novel View Synthesis. *European Conference On Computer Vision*. (2022)

[10] Rahaman, N., Baratin, A., Arpit, D., Draxler, F., Lin, M., Hamprecht, F., Bengio, Y. & Courville, A. On the Spectral Bias of Neural Networks. (2019)

[11] Barron, J., Mildenhall, B., Verbin, D., Srinivasan, P. & Hedman, P. Mip-NeRF 360: Unbounded Anti-Aliased Neural Radiance Fields. *CVPR*. (2022)

[12] Fridovich-Keil and Yu, Tancik, M., Chen, Q., Recht, B. & Kanazawa, A. Plenoxels: Radiance Fields without Neural Networks. *CVPR*. (2022)

[13] Müller, T., Evans, A., Schied, C. & Keller, A. Instant Neural Graphics Primitives with a Multiresolution Hash Encoding. *ACM Trans. Graph.*. **41**, 102:1-102:15 (2022,7), https://doi.org/10.1145/3528223.3530127

[14] Kerbl, B., Kopanas, G., Leimkühler, T. & Drettakis, G. 3D Gaussian Splatting for Real-Time Radiance Field Rendering. *ACM Transactions On Graphics*. **42** (2023,7), https://repo-sam.inria.fr/fungraph/3d-gaussian-splatting/

[15] Mittal, R. (n.d.). Lecture 11: Positive semidefinite matrix. https://www.cse.iitk.ac.in/users/rmittal/prev_course/s14/notes/lec11.pdf (visited on 11/05/2024)

[16] Henderson, T. C. (n.d.). A geometric interpretation of the covariance matrix. https://users.cs.utah.edu/ tch/CS4640/resources/A geometric interpretation of the covariance matrix.pdf (visited on 11/05/2024)

[17] M. Zwicker, H. Pfister, J. van Baar and M. Gross, "EWA volume splatting," Proceedings Visualization, 2001. VIS '01., San Diego, CA, USA, 2001, pp. 29-538, doi: 10.1109/VISUAL.2001.964490. keywords: Rendering (computer graphics);Surface fitting,

[18] Adinets, A. & Merrill, D. Onesweep: A Faster Least Significant Digit Radix Sort for GPUs. (2022)

[19] Radl, L., Steiner, M., Parger, M., Weinrauch, A., Kerbl, B. & Steinberger, M. StopThePop: Sorted Gaussian Splatting for View-Consistent Real-time Rendering. (2024)

[20] Fan, Z., Wang, K., Wen, K., Zhu, Z., Xu, D. & Wang, Z. LightGaussian: Unbounded 3D Gaussian Compression with 15x Reduction and 200+ FPS. (2024)

[21] Papantonakis, P., Kopanas, G., Kerbl, B., Lanvin, A. & Drettakis, G. Reducing the Memory Footprint of 3D Gaussian Splatting. *Proceedings Of The ACM On Computer Graphics And Interactive Techniques*. **7** (2024,5), https://repo-sam.inria.fr/fungraph/reduced-3dgs/

[22] Lee, J., Rho, D., Sun, X., Ko, J. & Park, E. Compact 3D Gaussian Representation for Radiance Field. (2024)

[23] Navaneet, K., Meibodi, K., Koohpayegani, S. & Pirsiavash, H. Compact3D: Compressing Gaussian Splat Radiance Field Models with Vector Quantization. (2023)

[24] Lee, J., Rho, D., Sun, X., Ko, J. & Park, E. Compact 3D Gaussian Representation for Radiance Field. *ArXiv Preprint ArXiv:2311.13681*. (2023)

[25] Niedermayr, S., Stumpfegger, J. & Westermann, R. Compressed 3D Gaussian Splatting for Accelerated Novel View Synthesis. (2024)

[26] Jin, X. & Han, J. K-Means Clustering. *Encyclopedia Of Machine Learning*. pp. 563-564 (2010), https://doi.org/10.1007/978-0-387-30164-8_425

[27] Shannon, C. A Mathematical Theory of Communication. *Bell System Technical Journal*. **27**, 379-423 (1948), https://onlinelibrary.wiley.com/doi/abs/10.1002/j.1538-7305.1948.tb01338.x

[28] Kaufman, L. and Rousseeuw, P.J. (1990) Partitioning around Medoids (Program PAM). In: Kaufman, L. and Rousseeuw, P.J., Eds., Finding Groups in Data: An Introduction to Cluster Analysis, John Wiley & Sons, Inc., Hoboken, 68-125.

[29] Ng, R. & Han, J. CLARANS: a method for clustering objects for spatial data mining. *IEEE Transactions On Knowledge And Data Engineering*. **14**, 1003-1016 (2002)

[30] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J. & Chintala, S. PyTorch: An Imperative Style, High-Performance Deep Learning Library. (2019)

[31] Harris, C., Millman, K., Walt, S., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N., Kern, R., Picus, M., Hoyer, S., Kerkwijk, M., Brett, M., Haldane, A., Río, J., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C. & Oliphant, T. Array programming with NumPy. *Nature*. **585**, 357-362 (2020,9), https://doi.org/10.1038/s41586-020-2649-2

[32] W3C WebGPU. (https://www.w3.org/TR/webgpu/,2023) (visited on 18/05/2024)

[33] Petamoriken Float16: JavaScript implementation of 16-bit floating point numbers. https://github.com/petamoriken/float16,2023) (visited on 18/05/2024)

[34] Rocks, W. WGSL Functions: Packing. (https://webgpu.rocks/wgsl/functions/packing/,2023) (visited on 18/05/2024)

[35] Erdős, P. & Rényi, A. On a classical problem of probability theory. *Publicationes Mathematicae*. **6** pp. 215-220 (1961)

[36] Brian Dawkins. Siobhan's problem: The coupon collector revisited. 45(1):76, February 1991.

Dataset	3DGS			K-means			AGORAS		
	SSIM	PSNR	LPSIS	SSIM	PSNR	LPSIS	SSIM	PSNR	LPSIS
Room	0.91	30.63	0.22	0.85	26.32	0.24	0.84	25.17	0.26
Counter	0.91	28.70	0.20	0.84	25.54	0.26	0.83	24.37	0.27
Kitchen	0.92	30.32	0.13	0.88	27.20	0.19	0.86	25.79	0.19
Bonsai	0.94	31.98	0.21	0.89	27.88	0.26	0.87	26.61	0.28

TABLE VI: Results from individual evaluated scenes of *Mip-NeRF360 indoor*