FACULTY OF ENGINEERING AND ARCHITECTURE

# Compression techniques for 3D Gaussian Splatting

Bert Ramlot
Student number: 01908133

Supervisors: Prof. dr. Peter Lambert, Prof. dr. ir. Glenn Van Wallendael
Counsellor: Ir. Martijn Courteaux

Master's dissertation submitted in order to obtain the academic degree of
Master of Science in Computer Science Engineering

Academic year 2023-2024

GHENT UNIVERSITY

FACULTY OF ENGINEERING AND ARCHITECTURE

# Compression techniques for 3D Gaussian Splatting

Bert Ramlot
Student number: 01908133

Supervisors: Prof. dr. Peter Lambert, Prof. dr. ir. Glenn Van Wallendael
Counsellor: Ir. Martijn Courteaux

Master's dissertation submitted in order to obtain the academic degree of
Master of Science in Computer Science Engineering

Academic year 2023-2024

GHENT UNIVERSITY

# Permission of use on loan

# Acknowledgements

I want to thank ir. Martijn Courteaux, without whom this thesis would not have been possible. Martijn Courteaux proposed the thesis' subject and provided invaluable guidance throughout this thesis. Additionally, I thank prof. dr. ir. Glenn Van Wallendael for his help structuring this thesis, and encouraging me to shape this thesis according to my vision and interests. Lastly, I thank my family who helped structure and proofread this dissertation.

Bert Ramlot
June 2024

# Explanatory note

This master's dissertation is part of an exam. Any comments formulated by the assessment committee during the oral presentation of the master's dissertation are not included in this text.

# Abstract

Creating 3D scenes and viewing them in real-time are important technologies in computer graphics and virtual reality. Recently, 3D Gaussian Splatting (3DGS) has emerged as the most promising contender to Neural Radiance Fields (NeRF) in 3D scene reconstruction and real-time novel view synthesis. 3DGS outperforms NeRF in training and inference speed, but falls short in storage requirements, with typical unbounded scenes requiring anywhere from 250 MB to 1.5 GB of space. To remedy this downside, this master's dissertation proposes two novel, post-training, compression techniques for 3DGS. First, a modified 3DGS rasterizer is proposed to accurately and efficiently calculate a splat's contribution and the change in PSNR upon its removal. Both metrics are then used to remove between 54% and 84% of splats, depending on the scene, while introducing only minor visual artifacts and without altering any other attributes. Splat removal also significantly boosts inference speeds, with some scenes more than doubling their framerate. Second, a novel spherical harmonics energy compaction method is proposed which substantially lowers the AC lighting coefficients' entropy. Using a heavily modified version of ridge regression, up to 95% of AC lighting coefficients are set to zero while simultaneously lowering their L2 norm. Additionally, the proposed method can be used for a wide range of post-processing spherical harmonics operations, for example, increasing/decreasing the degree of spherical harmonics after training or removing hallucinated colors. Finally, the two novel compression techniques are combined to form an encoder that achieves compression ratios of 35-96x, compressing 255-844 MB scenes to 7-10 MB, while minorly affecting visual acuity. Contrary to all other 3DGS compression literature, no gradient descent-based methods are used.

**Keywords:** 3DGS, Compression, Culling, Spherical harmonics

# Compression techniques for 3D Gaussian Splatting

Bert Ramlot
Supervisors: Prof. dr. ir. Glenn Van Wallendael, Prof. dr. Peter Lambert
Counsellors: Ir. Martijn Courteaux

***Abstract*—Creating 3D scenes and viewing them in real-time are important technologies in computer graphics and virtual reality. Recently, 3D Gaussian Splatting (3DGS) has emerged as the most promising contender to Neural Radiance Fields (NeRF) in 3D scene reconstruction and real-time novel view synthesis. 3DGS outperforms NeRF in training and inference speed, but has substantially higher storage requirements. To remedy this downside, this work proposes two novel, post-training, compression techniques for 3DGS. First, a modified 3DGS rasterizer is proposed to accurately and efficiently calculate a splat's contribution and the change in PSNR upon its removal. Both metrics are then used to remove between 54% and 84% of splats, depending on the scene, while introducing only minor visual artifacts and without altering any other attributes. Splat removal also significantly boosts inference speeds, with some scenes more than doubling their framerate. Second, a novel spherical harmonics energy compaction method is proposed which substantially lowers the AC lighting coefficients' entropy. Using a heavily modified version of ridge regression, up to 95% of AC lighting coefficients are set to zero while simultaneously lowering their L2 norm. Finally, the two novel compression techniques are combined to form an encoder that achieves compression ratios of 35-96x, compressing 255-844 MB scenes to 7-10 MB, while minorly affecting visual acuity. Contrary to all other 3DGS compression literature, no gradient descent-based methods are used.**

***Index Terms*—3DGS, Compression, Culling, Spherical harmonics**

## I. Introduction

Creating realistic pictures from new angles using only a few real photos is a long-standing challenge in computer graphics. Lately, Neural Radiance Fields (NeRF) [1] have become increasingly popular with substantial improvements year over year [2, 3, 4]. More recently, 3D Gaussian Splatting (3DGS) [5] emerged as the most promising contender to the now dominant NeRF with better training and inference speeds. 3DGS achieves this feat through a custom rasterizer and scene representation based on feature-rich volumetric points called 'splats'. This alternate representation is also the culprit behind 3DGS's biggest comparative downside to NeRF, namely, substantially higher storage requirements. For many practical applications of 3DGS, the typical unbounded scene size of 250 MB to 1.5 GB is prohibitively large. As a result, applications that download scenes on demand, e.g. web viewers, already apply crude compression methods. Furthermore, compression will become more important as Gaussian splatting is expanded towards larger, non-static scenes for most practical use cases, e.g. immersive video.

Therefore, to address this downside, this work explores and proposes compression methods for 3DGS. Contrary to other compression works which alter the training loop [6, 7, 8] or train to remove introduced artifacts [9, 10], this work operates purely *after* the scene has already been trained. Furthermore, no gradient-descent-based methods of any kind are used.

First, the number of splats in a scene is identified as a major contributor to the size and consequently reduced by 54% to 84%, depending on the scene. This reduction introduces only minor visual artifacts and leaves all attributes untouched. Additionally, the inference framerates of the four studied scenes increased by 39% to 132%. Second, the AC lighting coefficients' entropy is lowered substantially, while only introducing a minor loss in quality through a novel energy compaction method. This method aggressively increases sparsity and lowers the L2 norm of the AC lighting coefficients using a heavily modified version of ridge regression. Moreover, this method forms the baseline for numerous spherical harmonics-based operations, e.g. increasing or decreasing the degree of spherical harmonics used and fixing color aliasing problems.

## II. Gaussian splatting

### A. 3D Gaussians

3DGS scenes consist of a collection of splats. Each splat affects the transparency of the space around it, emitting the splat's color to the camera proportional to the transparency it absorbs. The degree to which a splat absorbs the transparency and emits light at a point in space $\mathbf{x} \in \mathbb{R}^3$ is proportional to a 3D Gaussian:

$$\mathcal{G}(\mathbf{x}) = \exp(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})) \tag{1}$$

where $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ denote the splat's center and covariance matrix respectively. For a given camera, this 3D Gaussian is projected into screen space as a 2D Gaussian $\mathcal{G}^{\text{2D}}$, exploiting multivariate Gaussians' affine invariance.

### B. Alpha compositing

A pixel's color is the result of combining multiple splats' colors via alpha compositing. Each splat corresponds with a layer (and vice-versa) in the alpha compositing process, where the layers are ordered according to the depth of their associated splats. A layer's pixel, located at the screen space position $\mathbf{x}$, has an associated opacity $\alpha$:

$$\alpha = o \cdot \mathcal{G}^{2D}(\mathbf{x}) \tag{2}$$

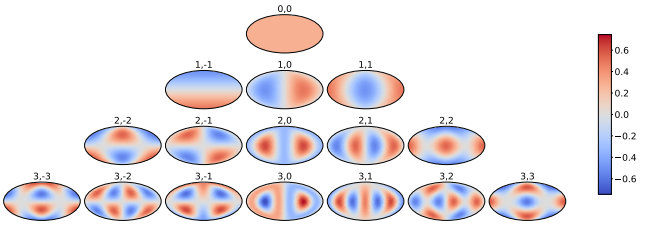



Fig. 1: The Mollweide projection of all real spherical harmonics with $l \leq 3$. Labels are of form $l, m$.

where $o$ is the location-independent base opacity of a splat. The yet undetermined fraction of a pixel's color is called the *transmittance* and after the $k$ first layers, it is defined as:

$$T_k = \prod_{j=0}^{k-1}(1 - \alpha_j) \tag{3}$$

Putting everything together, a pixel's final color, after alpha-composited $K$ layers, is defined as:

$$\mathbf{c}(\mathbf{x}) = \sum_{k=0}^{K-1} T_k \alpha_k \mathbf{c}_k \tag{4}$$

### C. Spherical Harmonics

The alpha compositing process uses a splat's color $\mathbf{c}_k$. This color changes in function of the camera's position using spherical harmonics. A real spherical harmonic is denoted as $Y_{l,m}$, where $l \in \mathbb{N}_0$ represents the degree and $m = -l, -l+1, ..., l$ the order. Figure 1 shows all real spherical harmonics with a degree $\leq 3$ using the Mollweide projection, an equal-area projection method.

The decomposition of a spherical function $L(\theta, \phi)$ using spherical harmonics is defined as follows:

$$L(\theta, \phi) = \sum_{l,m} L_{l,m} Y_{l,m}(\theta, \phi) \tag{5}$$

where $L_{l,m}$ is $Y_{l,m}$'s coefficient, and often simply referred to as its *lighting coefficient*. As it is impractical to iterate over spherical harmonics using the degree and order, each $l, m$ pair is often mapped to a unique scalar. The mapping chosen for this work iterates over the spherical harmonics row-by-row, left-to-right as seen in Figure 1.

Within 3DGS, three color channels (RGB) are defined using spherical harmonics, resulting in a splat's color being fully specified by three sets of lighting coefficients. To convert these lighting coefficients to an actual color, only the position of the splat and camera is required, meaning that a splat is monochromatic within a single frame so that only the opacity changes between pixels.

## III. Splat Removal

### A. Analysis

Let the contribution of a splat to a camera $\mathbf{s}$ be the fraction of the rendered image's color originating from said splat. A

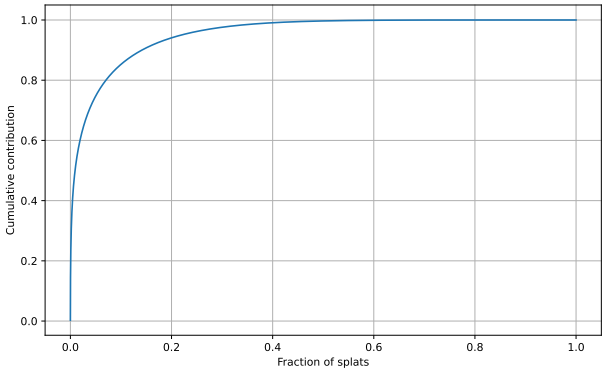



Fig. 2: Cumulative contribution of the fraction of the highest contributing splats for the scene Truck.

splat's contribution for a render of width $W$ and height $H$ can be calculated as follows:

$$I_k(\mathbf{s}) = \frac{1}{WH} \sum_{\mathbf{x}} T_{i_k} \alpha_{i_k} \tag{6}$$

The scene-wide equivalent, called the contribution of a splat, is defined as follows, given $N$ cameras:

$$I_k = \frac{1}{N} \sum_{\mathbf{s}} I_k(\mathbf{s}) \tag{7}$$

Figure 2 shows the cumulative contribution for different fractions of the highest contributing splats. Using this figure, two observations are made:

1) There is a steep initial climb in the cumulative contribution, indicating that a small number of splats make up most of the final image.
2) A cumulative contribution of nearly 100% is achieved after only considering 45% of splats. The majority of the splats therefore have little to no impact on the scene.

These two observations show the immense potential of splat removal as a compression technique.

### B. Effect of a splat's removal

A splat with a negligible contribution is a sufficient condition for its removal to have an insignificant impact on the visual quality, however, it is not a necessary one. For example, removing a red splat might reveal another red splat behind it, making its removal effect very minor, regardless of the contribution of the splat. Thus, the actual value we are interested in is how the pixel values of the rendered images change due to a splat's removal. Let therefore the cull difference $\mathbf{CD}_j(\mathbf{x})$ be the change in a pixel's color due to the $j$-th splat's removal. $\mathbf{CD}_j(\mathbf{x})$ can be calculated as follows:

$$\begin{aligned} \mathbf{CD}_j(\mathbf{x}) &= \tilde{\mathbf{c}}_j(\mathbf{x}) - \mathbf{c}(\mathbf{x}) \\ &= -T_j \alpha_j \mathbf{c}_j + \frac{\alpha_j}{1 - \alpha_j} \sum_{k=j+1}^{K} T_k \alpha_k \mathbf{c}_k \end{aligned} \tag{8}$$

where $\mathbf{c}(\mathbf{x})$ and $\tilde{\mathbf{c}}_j(\mathbf{x})$ are the color of the pixel, prior and post the removal of the $j$-th splat respectively. The cull difference

is used to calculate the difference in the squared error as a result of the removal of the $j$-th splat for a single pixel:

$$\triangle\mathbf{SE}_j(\mathbf{x}) = (\mathbf{CD}_j(\mathbf{x}))^2 + 2\ \mathbf{CD}_j(\mathbf{x})\ (\mathbf{c}(\mathbf{x}) - \mathbf{c}_{\text{gt}}(\mathbf{x})) \quad (9)$$

We are interested in the effect a splat's removal has on the scene as a whole, not solely on a single pixel:

$$\triangle\text{SE}_j = \sum_{\mathbf{x}} \triangle\mathbf{SE}_j(\mathbf{x}) \cdot \mathbf{1} \quad (10)$$

When all $\triangle\mathbf{SE}_j(\mathbf{x})$ values are averaged instead of summed, the mean squared error $\triangle\text{MSE}_j$ is calculated instead. Efficiently calculating $\triangle\text{SE}_j$ is done using a quantity already available during the render pass: the partial color $\mathbf{P}_j$. This is the color of a pixel after the $j$-th splat has been considered in the alpha compositing process and is defined as follows:

$$\mathbf{P}_j = \sum_{k=0}^{j} T_k \alpha_k \mathbf{c}_k \quad (11)$$

The cull difference can be rewritten in terms of partial colors and $\alpha_j$:

$$\mathbf{CD}_j(\mathbf{x}) = \mathbf{P}_{j-1} - \frac{1}{1-\alpha_j}\mathbf{P}_j + \frac{\alpha_j}{1-\alpha_j}\mathbf{P}_K \quad (12)$$

Using this equation, the proposed modified rasterizer efficiently calculates $\mathbf{CD}_j(\mathbf{x})$. This is in turn used to calculate all $\triangle\mathbf{SE}_j(\mathbf{x})$ values using Equation 9, all of whom are accumulated per splat to arrive at all $\triangle\text{SE}_j$ values.

### C. Culling controller

Knowing all $\triangle\text{SE}_j$ values does not yet provide a methodology on *how* and *which* splats to remove. This is what the culling controller is responsible for and is non-trivial due to a splat's removal possibly changing other splat's $\triangle\text{SE}_j$. The culling controller should prioritize removing as many splats as quickly as possible since this speeds up all future render passes. This leads to the counter-intuitive result that a good culling controller does not start by trying to remove the splats with the lowest $\triangle\text{SE}_j$.

The proposed culling controller starts from an empty $S$ and adds splats one by one. To define the quality-compression trade-off, the maximal error that can be introduced by removing a splat $\triangle\text{MSE}_{\text{MAX}}$ is given. The culling controller tries to end up with little to no splats where $\triangle\text{MSE}_j < \triangle\text{MSE}_{\text{MAX}}$. To determine when to stop adding splats to $S$, a splat's contribution is used as a proxy for its removal effect. Once the 'culling budget' is reached, $S$ is finalized and all splats in $S$ are removed. The culling budget is calculated as follows:

$$\text{cull\_budget} = \frac{1}{\#\text{ iters left}} \sum_{k \in \{\, j \mid \triangle\text{MSE}_j < \triangle\text{MSE}_{\text{MAX}} \,\}} I_k \quad (13)$$

The culling budget is reached when:

$$\text{cull\_budget} \leq \sum_{k \in S} I_k \quad (14)$$

Splats are added to $S$ is in ascending order of their $m(\frac{\triangle\text{MSE}_j}{\triangle\text{MSE}_{\text{MAX}}})$ value. The mapping function $m(x)$ is chosen so

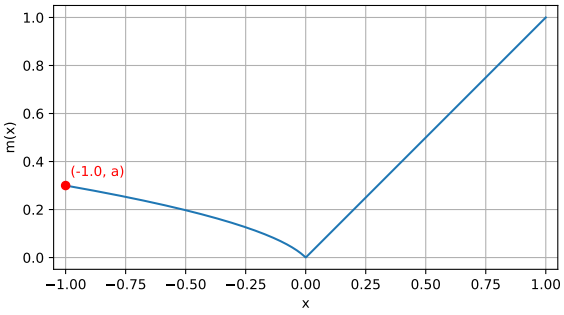



Fig. 3: The mapping function $m(x)$ with $a = 0.3$.

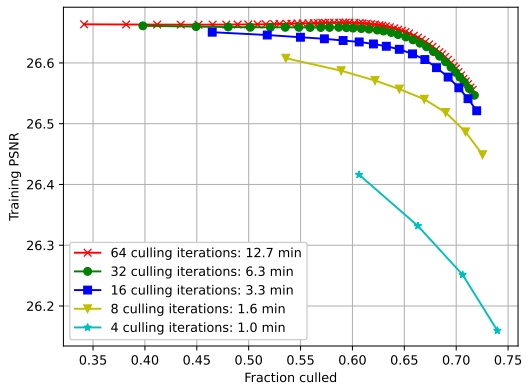



Fig. 4: Effect of the number of culling iterations on the training PSNR of the scene Truck ($\triangle\text{MSE}_{\text{MAX}} = 5 \cdot 10^{-10}$).

splats with a small $|x|$ are removed first, with the exact sign of $x$ not being important so long that $x$ stays sufficiently small. The preference shifts towards negative $x$ values as $|x|$ grows. Practically, the following function was chosen:

$$m(x) = \begin{cases} x & x \geq 0 \\ \frac{a^2}{2(1-a)}\left(\sqrt{1 - \frac{4(1-a)}{a^2}x} - 1\right) & x < 0 \end{cases} \quad (15)$$

The meaning of $a$ is conveyed in Figure 3 which also visualizes $m(x)$ in the relevant range of values.

### D. Results

Figure 4 shows the results for differing numbers of culling iterations. For this configuration, using more than 32 culling iterations has negligible benefits.

Culling will be the first step in the proposed encoder, and thus, the starting point for further compression steps. To simplify all upcoming discussions, a default starting point, namely 32 culling iterations with $\triangle\text{MSE}_{\text{MAX}} = 5 \cdot 10^{-10}$, is chosen. An overview of this configuration's results on four scenes is shown in Table I.

| Scene | PSNR$^{\uparrow}$ (+/-$^{\uparrow}$) | SSIM $^{\uparrow}$ (+/- $^{\uparrow}$) | LPIPS $^{\downarrow}$ (+/- $^{\downarrow}$) | # Splats $^{\downarrow}$ (% removed $^{\uparrow}$) | FPS $^{\uparrow}$ (+/- $^{\uparrow}$) | Runtime $^{\downarrow}$ |
|---|---|---|---|---|---|---|
| Drjohnson [11] | 29.53 (-0.59) | 0.889 (-0.007) | 0.271 (+0.023) | 600,180 (82%) | 220 (+117) | 10.5 min |
| Playroom [11] | 29.64 (-0.29) | 0.888 (-0.013) | 0.271 (+0.026) | 419,181 (84%) | 220 (+125) | 7.2 min |
| Train [12] | 22.12 (+0.35) | 0.793 (-0.011) | 0.238 (+0.020) | 467,167 (54%) | 125 (+35) | 6.7 min |
| Truck [12] | 24.87 (-0.08) | 0.858 (-0.013) | 0.171 (+0.017) | 717,686 (72%) | 121 (+48) | 6.3 min |

TABLE I: The proposed method's post-culling results for various scenes. The FPS and runtime were measured using a GTX 1660 Ti.

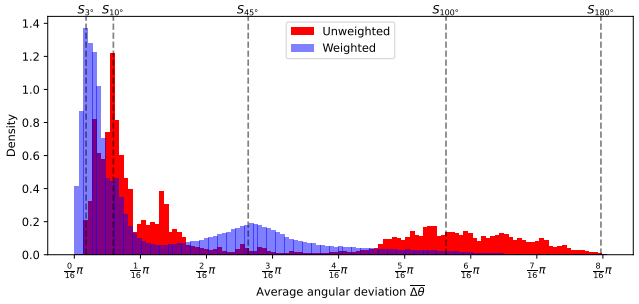



Fig. 5: Histogram of the average angular deviation $\overline{\Delta\theta}$ for all splats in the scene Truck. The unweighted and weighted versions ($\mathbf{w}_k = (I_k(\mathbf{s}_1), ..., I_k(\mathbf{s}_N))^T$) are shown. Some reference uniformly sampled spherical caps $S_\alpha$, with cone angle $\alpha$, are marked.

## IV. Spherical Harmonics Energy Compaction

### A. Analysis

Splats' colors are defined for every viewing angle while only a small fraction is typically relevant. To help quantify the severity of this phenomenon, the average angular deviation $\overline{\Delta\theta}$ is defined as the average angle between a splat's observation direction for a given camera and the splat's mean observation direction. A splat's observation direction for a given camera corresponds to the vector going from the splat center to the camera. We talk about the weighted average angular deviation if the mean observation direction and average deviation are weighed by some vector $\mathbf{w}$. Figure 5 shows a histogram of $\overline{\Delta\theta}$ for all splats.

A small $\overline{\Delta\theta}$ indicates that a splat is only looked at from a narrow range of directions. Changing a splat's color for observation directions outside this meaningful region will not have any visual impact as the splat is never looked at from said region. Therefore, the smaller $\overline{\Delta\theta}$ is, the more combinations of lighting coefficients will produce near identical results. The key idea is to find an alternate set of lighting coefficients that is easier to compress. Such a set typically has low energy and high sparsity.

### B. Proposed method

Consider a single channel of a splat's color for the $i$-th camera. A channel's value is defined using its $M$ lighting coefficients:

$$C(\mathbf{s}_i) = \sum_{l=1}^{M} L_l Y_l(\mathbf{s}_i) \tag{16}$$

Equation 16 holds for all $N$ cameras. All the corresponding equations for a single splat can be combined using a matrix multiplication as follows:

$$\underbrace{\begin{pmatrix} C(\mathbf{s}_1) \\ C(\mathbf{s}_1) \\ \vdots \\ C(\mathbf{s}_N) \end{pmatrix}}_{\mathbf{C}} = \underbrace{\begin{pmatrix} Y_1(\mathbf{s}_1) & Y_2(\mathbf{s}_1) & \cdots & Y_M(\mathbf{s}_1) \\ Y_1(\mathbf{s}_2) & Y_2(\mathbf{s}_2) & \cdots & Y_M(\mathbf{s}_2) \\ \vdots & \vdots & \ddots & \vdots \\ Y_1(\mathbf{s}_N) & Y_2(\mathbf{s}_N) & \cdots & Y_M(\mathbf{s}_N) \end{pmatrix}}_{\mathbf{Y}} \times \underbrace{\begin{pmatrix} L_1 \\ L_2 \\ \vdots \\ L_M \end{pmatrix}}_{\mathbf{L}} \tag{17}$$

Energy compaction aims to find a new $\mathbf{L}$, denoted as $\mathbf{L}'$, located on the Pareto front of 'quality' and 'compressibility'. This work uses ridge regression to define the optimal trade-off:

$$\mathbf{L}' = \underset{\mathbf{x}\in\mathbb{R}^{M\times 1}}{\arg\min}(\|\mathbf{Y}\mathbf{x} - \mathbf{C}\|_2^2 + \|\mathbf{\Gamma}\mathbf{x}\|_2^2) \tag{18}$$

where $\|\cdot\|_2$ is the Euclidian norm of a vector, and $\mathbf{\Gamma}$ denotes the Tikhonov matrix which defines how $\mathbf{x}$ is regularized. The Tikhonov matrix is chosen as follows:

$$\mathbf{\Gamma} = \sqrt{\lambda}\left(\begin{array}{c|c} 0 & \mathbf{0}_{1\times M-1} \\ \hline \mathbf{0}_{M-1\times 1} & \mathbf{I}_{M-1\times M-1} \end{array}\right) \tag{19}$$

where $\mathbf{I}$ denotes the identity matrix. The top-left element is zero as we do not wish to regularize the DC lighting coefficient.

Ridge regression was chosen as it has a closed-form solution, which is essential as $\mathbf{L}'$ has to be calculated for each splat. The proposed version of ridge regression is heavily modified to, in part, fix the weaknesses of ridge regression:

*1) Contribution:* A splat's contribution to a camera differs greatly from camera to camera. Equation 18 punishes each deviation, $\mathbf{C}'_i - \mathbf{C}_i$, equally for all cameras. A more fair loss function should punish deviations in accordance to the cameras' contributions. This can be achieved by doing the following substitutions in Equation 18:

$$\mathbf{C} \leftarrow \mathbf{C} \odot_{\text{col}} (I_k(\mathbf{s}_1), ..., I_k(\mathbf{s}_N))^T \tag{20}$$

$$\mathbf{Y} \leftarrow \mathbf{Y} \odot_{\text{col}} (I_k(\mathbf{s}_1), ..., I_k(\mathbf{s}_N))^T \tag{21}$$

where $\odot_{\text{col}}$ represents a column element-wise multiplication.

*2) Color model:* 3DGS uses RGB as its color model, unsurprisingly, lighting coefficients corresponding to the same basis function, but a different color channel, correlate highly. Moving to a luminance/chrominance-based color model, e.g. YUV, has two benefits: (1) the collective energy is reduced, and (2) the chrominance channels can be more harshly regularized as humans are comparatively less perceptive to changes in chrominance.

*3) Sparsity:* Ridge regression comes with the notable downside that the magnitude of lighting coefficients is punished super-linearly. As a result, energy is dispersed more than desirable across multiple lighting coefficients.

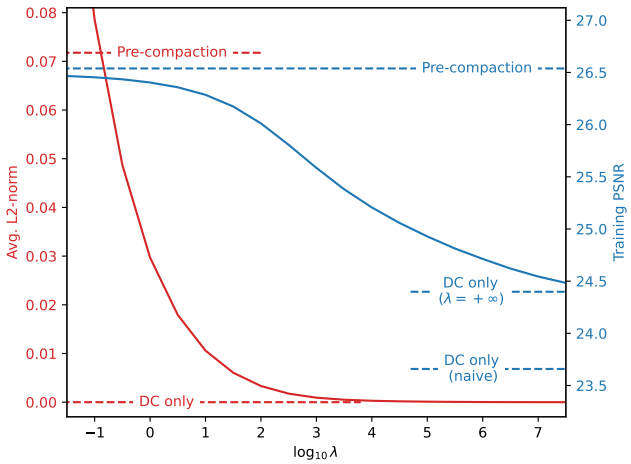
To promote sparsity and prevent under-regularization, the $i$-th column of $\mathbf{Y}$ is set to zero (which leads to $L'_i = 0$) if a $j$-th column vector exists which is 'sufficient parallel' and where $j < i$. Sufficiently parallel is defined as having a cosine similarity whose absolute value is larger than some constant $\alpha$.

*4) Two-pass regularization:* Removing parallel columns using $\alpha = 0.9$ for the scene Truck sets 60% of AC coefficients to zero. Ridge regression forces an additional 34% of AC coefficients to be *almost* zero such that these lighting coefficients will be quantized to zero in Section V. To reduce the quantization loss and to present ridge regression with a more true representation of its flexibility, a two-pass regularization system is used. The first pass identifies all almost zero AC coefficients and forces them to zero using the same technique presented in the sparsity section. The second pass finds the optimal values for the remaining non-zero lighting coefficients.

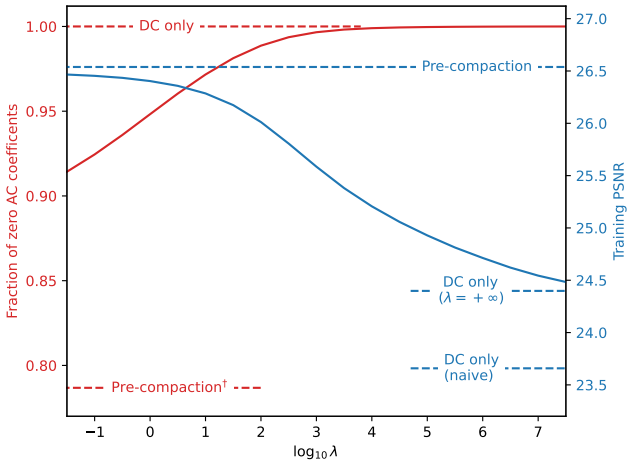
### C. Results

Figure 6 shows that the proposed method can lower the L2-norm and increase the sparsity while minimally affecting the PSNR. The default configuration, $\lambda = 1.0$ and $\alpha = 0.9$, matches the default configuration's splat removal compression-quality trade-off.

The proposed method can identify and exploit compression opportunities. For example, Figure 7 shows that while every type of splat gets more sparse after energy compaction, splats outside the camera circle get markedly sparser. Similar discrepancies exist when comparing different color channels, spherical harmonics basis, and splats with differing contributions. Finally, Figure 8 provides a practical example of how energy compaction can affect the splats of a scene.




(a) L2-norm and quality




(b) Sparsity and quality

Fig. 6: Effect of $\lambda$ on the L2-norm, sparsity, and quality with $\alpha = 0.9$. 'Naive' refers to setting all AC coefficients to zero without recomputing the DC coefficients. † denotes that an AC coefficient is regarded as zero if $|L_i| < \frac{1}{32}$.

## V. Encoder Design

This section constructs an encoder (and decoder) using the compression techniques presented in sections III and IV. Some tangental topics are briefly discussed including quantization and lossless compression. The proposed encoder gives the final compression, quality, and runtime results.

### A. Quantization

All attributes are represented as single precision floats, each taking up 4 bytes. This level of precision is unnecessary and detrimental to lossless compression. All attributes are therefore quantized to resolve this, tailoring the quantization on a per-attribute basis.

The quantized positions of the splats are represented by the centers of the leaves of an octree, where each leaf corresponds with exactly one splat. To increase the precision of the $j$-th splat's position, the leaf in which it resides can be repeatably split until the Euclidean distance between the leaf center and the $j$-th splat's position is smaller than some threshold $\gamma_j$. Using the hyperparameter $\texttt{PF}$ and the Euclidean distance between the $j$-th splat and its closest camera $l_j^{\min}$, the proposed encoder sets $\gamma_j$ to:

$$\gamma_j = \texttt{PF} \cdot l_j^{\min} \tag{22}$$

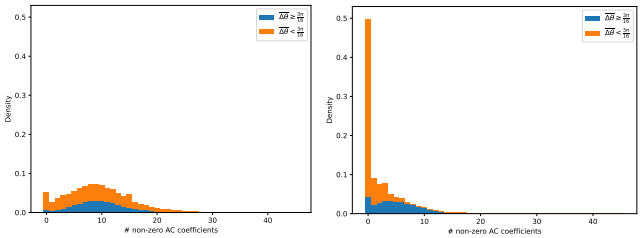
This is based on the fact that if a splat is a distance $l$ removed from a camera, then a small change $dx$ in a positional attribute will cause the projected splat in screen space to move approximately proportional to $\frac{dx}{l}$.

The non-positional attributes are uniformly quantized by converting them to integers using the following quantization function:

$$q(x) = \left\lfloor \frac{1}{2} + x \cdot \texttt{PF} \right\rfloor \tag{23}$$

| Scene | PSNR$^{\uparrow}$ (+/-$^{\uparrow}$) | SSIM$^{\uparrow}$ (+/-$^{\uparrow}$) | LPIPS$^{\downarrow}$ (+/-$^{\downarrow}$) | Size$^{\downarrow}$ (compression factor$^{\uparrow}$) | FPS$^{\uparrow}$ (+/-$^{\uparrow}$) | Runtime$^{\downarrow}$ |
|---|---|---|---|---|---|---|
| Drjohnson [11] | 29.48 (-0.54) | 0.887 (-0.009) | 0.275 (+0.027) | 8.8 MB (96x) | 220 (+117) | 14.7 min |
| Playroom [11] | 29.47 (-0.45) | 0.884 (-0.017) | 0.274 (+0.030) | 6.9 MB (91x) | 220 (+125) | 8.9 min |
| Train [12] | 22.06 (+0.29) | 0.786 (-0.019) | 0.246 (+0.028) | 8.3 MB (35x) | 125 (+35) | 9.2 min |
| Truck [12] | 24.76 (-0.18) | 0.854 (-0.017) | 0.177 (+0.022) | 10 MB (62x) | 121 (+48) | 9.7 min |

TABLE II: The proposed encoder/decoder's results for various scenes.




(a) Before energy compaction ($L_i$ is non-zero if $|L_i| < \frac{1}{32}$)

(b) After energy compaction ($L_i$ is non-zero if $L_i \neq 0$)

Fig. 7: Stacked density histogram of the number of non-zero AC lighting coefficients per splat. Splats outside the camera ring ($\overline{\Delta\theta} < \frac{3\pi}{16}$), which are more unequally sampled, have a significantly higher sparsity after energy compaction.

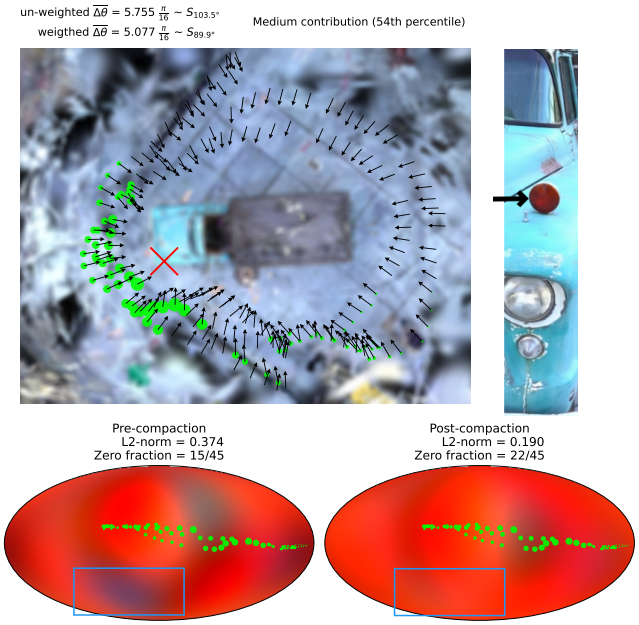



Fig. 8: A red cross marks a medium contributing splat, part of the truck's red reflector, on a blurred, top-down view, of the scene Truck. All training cameras are shown as black arrows with the size of the green dot indicating the cameras' relative contribution. Additionally, Mollweide projections visualize the splat's possible colors, with each green dot representing a camera and their relative contribution. After energy compaction, the L2-norm decreases, the sparsity increases, and a hallucinated blue color (highlighted using a blue rectangle) is removed from an unsampled viewing area.

De-quantizing such an attribute is as simple as dividing the quantized value by the hyperparameter `PF`. The precision factor `PF` differs from attribute to attribute.

### B. Order of steps

Due to culling speeding up other operations, the encoder starts by running most of its allocated culling iterations. After achieving the bulk of the performance benefits from culling, all attributes, except the lighting coefficients, are quantized and de-quantized. This provides the subsequent energy compaction step with a representation already very close to the final representation. The new lighting coefficients are quantized and de-quantized, just as the other attributes, such that the remaining culling iterations are given as close to the decoded representation as possible. After the last culling iteration, all lossless steps are finished and the quantized scene is serialized. This bitstream is then losslessly compressed using zstd [1]. The proposed decoder decompresses, deserializes, and de-quantizes the bitstream in that order. Figure 9 provides an overview of the encoder and decoder with achievable metrics for the scene Truck.

### C. Results

Table II lists the proposed method's result on various scenes. In conjunction, Figure 10 gives an idea of the achieved quality and introduced artifacts. The achieved compression factor differs substantially between scenes, primarily due to varying degrees of culling effectiveness (as seen in Table I). Depending on the scene, the non-culling compression steps achieve a compression factor of around 13x to 16x. Not all attributes are compressed equally, the AC lighting coefficients are compressed approximately 68x while all other attributes are compressed anywhere from 3x to 6x. Without the spherical harmonics energy compaction step, the final size would approximately double.

Due to AC lighting coefficients being compressed so much, they are no longer the primary contributor to size. The DC lighting coefficients are approximately as expensive to encode as all AC lighting coefficients.

## VI. Conclusion

This work proposes two novel compression techniques and an associated encoder/decoder for 3D Gaussian splatting scene compression. All presented techniques are post-processing

[1] https://github.com/facebook/zstd

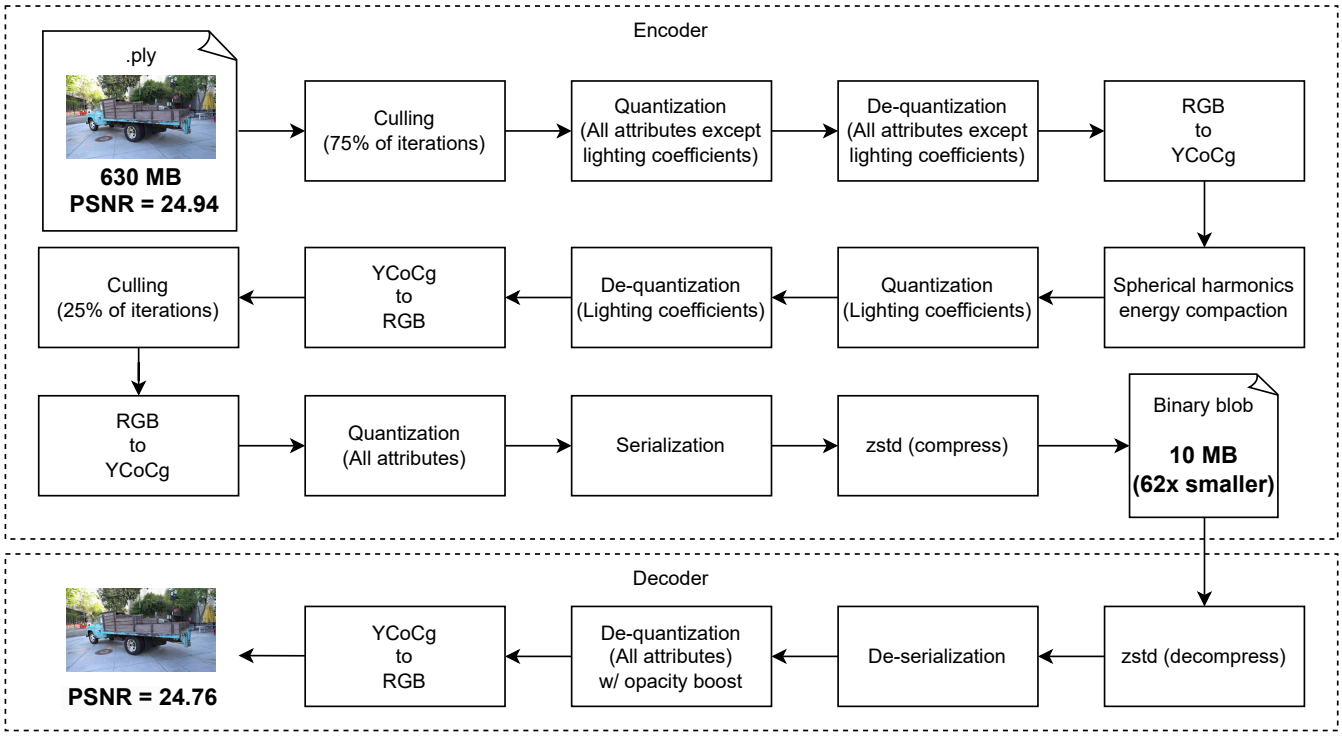



Fig. 9: Schematic diagram of the proposed encoder and decoder.

steps, operating independently of the training loop. Additionally, this is the first compression work not to utilize gradient descent-based steps. Despite this, experiments on 4 scenes show that compression factors from 35x to 96x are achievable with minor visual artifacts.

## REFERENCES


[1] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, "Nerf: Representing scenes as neural radiance fields for view synthesis," in *ECCV*, 2020.

[2] J. T. Barron, B. Mildenhall, M. Tancik, P. Hedman, R. Martin-Brualla, and P. P. Srinivasan, "Mip-nerf: A multiscale representation for anti-aliasing neural radiance fields," *ICCV*, 2021.

[3] J. T. Barron, B. Mildenhall, D. Verbin, P. P. Srinivasan, and P. Hedman, "Mip-nerf 360: Unbounded anti-aliased neural radiance fields," *CVPR*, 2022.

[4] ——, "Zip-nerf: Anti-aliased grid-based neural radiance fields," *ICCV*, 2023.

[5] B. Kerbl, G. Kopanas, T. Leimkühler, and G. Drettakis, "3d gaussian splatting for real-time radiance field rendering," *ACM Transactions on Graphics*, vol. 42, no. 4, July 2023. [Online]. Available: https://repo-sam.inria.fr/fungraph/3d-gaussian-splatting/

[6] J. C. Lee, D. Rho, X. Sun, J. H. Ko, and E. Park, "Compact 3d gaussian representation for radiance field," *arXiv preprint arXiv:2311.13681*, 2023.

[7] W. Morgenstern, F. Barthel, A. Hilsmann, and P. Eisert, "Compact 3d scene representation via self-organizing gaussian grids," 2023.

[8] K. Navaneet, K. P. Meibodi, S. A. Koohpayegani, and H. Pirsiavash, "Compact3d: Compressing gaussian splat radiance field models with vector quantization," 2023.

[9] S. Niedermayr, J. Stumpfegger, and R. Westermann, "Compressed 3d gaussian splatting for accelerated novel view synthesis," 2023.

[10] Z. Fan, K. Wang, K. Wen, Z. Zhu, D. Xu, and Z. Wang, "Lightgaussian: Unbounded 3d gaussian compression with 15x reduction and 200+ fps," 2023.

[11] P. Hedman, J. Philip, T. Price, J.-M. Frahm, G. Drettakis, and G. Brostow, "Deep blending for free-viewpoint image-based rendering," *ACM Transactions on Graphics (SIGGRAPH Asia Conference Proceedings)*, vol. 37, no. 6, November 2018. [Online]. Available: http://www-sop.inria.fr/reves/Basilic/2018/HPPFDB18

[12] A. Knapitsch, J. Park, Q.-Y. Zhou, and V. Koltun, "Tanks and temples: Benchmarking large-scale scene reconstruction," *ACM Transactions on Graphics*, vol. 36, no. 4, 2017.


Fig. 10: A comparison of test images before and after compression with corresponding ground truth images. The test images correspond with those presented by 3DGS [5].

# Contents

# List of Figures

# List of Tables

# 1

# Introduction

Creating realistic pictures from new angles using only a few real photos is a long-standing challenge in computer graphics. Lately, Neural Radiance Fields (NeRF) [1] have become increasingly popular with substantial improvements year over year [3, 4, 5]. More recently, 3D Gaussian Splatting (3DGS) [2] emerged as the most promising contender to the now dominant NeRF with better training and inference speeds. 3DGS achieves this feat through a custom rasterizer and scene representation based on feature-rich volumetric points called 'splats'. This alternate representation is also the culprit behind 3DGS's biggest comparative downside to NeRF, namely, substantially higher storage requirements. For many practical applications of 3DGS, the typical unbounded scene size of 250 MB to 1.5 GB is prohibitively large. As a result, applications that download scenes on demand, e.g. web viewers, already apply crude compression methods. Furthermore, compression will become more important as Gaussian splatting is expanded towards larger, non-static scenes for most practical use cases, e.g. immersive video.

Therefore, to address this downside, this dissertation explores and proposes compression methods for 3DGS. Contrary to other compression works which alter the training loop [6, 7, 8] or train to remove introduced artifacts [9, 10], this work operates purely *after* the scene has already been trained. Furthermore, no gradient-descent-based methods of any kind are used.

First, the number of splats in a scene is identified as a major contributor to the size and consequently reduced by 54% to 84%, depending on the scene. This reduction introduces only minor visual artifacts and leaves all attributes untouched. Additionally, the inference framerates of the four studied scenes increased by 39% to 132%. Second, the AC lighting coefficients' entropy is lowered substantially, while only introducing a minor loss in quality through a novel energy compaction method. This method aggressively increases sparsity and lowers the L2 norm of the AC lighting coefficients using a heavily modified version of ridge regression. Moreover, this method forms the baseline for numerous spherical harmonics-based operations, e.g. increasing or decreasing the degree of spherical harmonics used and fixing color aliasing problems.

Before delving into the compression of 3DGS scenes, Chapter 2 first introduces novel view synthesis and discusses recent advances, contextualizing the field in which 3DGS exists. Chapter 3 discusses 3DGS further to lay the foundation for the compression discussion in Chapter 4. The compression discussion covers the state-of-the-art techniques and identifies why 3DGS files are so large. Using this background, Chapter 5 presents a high-level framework on how I reason about compression within 3DGS. Using this framework, two novel compression techniques are discussed in Chapter 6 and 7 for culling and energy compaction respectively. Both techniques are then combined into an encoder/decoder in Chapter 8. Finally, future research directions are discussed in Chapter 9 before ending with a conclusion.

# 2

# Novel view synthesis

> *Can we ask photography to give us back all the variety offered by the direct view of objects? Is it possible to create a photographic print in such a way as to represent the outside world as framed, in appearance, between the edges of the print, as if these edges were those of an open window? The edges of the print, as if those edges were the edges of a window to reality? It seems so; we can ask infinitely more of Photography than of the human hand. — Gabriel Lippmann, 1906 [11]*

In 1860, shortly after the invention of the black-and-white photograph, François Willème invented the *photo sculpture*, a novel systematic technique to create sculptures by taking several pictures from different viewpoints. It works as follows: a person is placed in the center of a camera circle where all cameras simultaneously capture an image. Each photograph is traced onto a slice of wood using a pantograph, a tracing tool for transferring and rescaling a trace. These wood slices are then combined to form the rough outline of a sculpture, an example of which is shown in Figure 2.1.

This marked the first time a systematic technique was used to go from a set of images to a medium that allows for novel viewpoints to be synthesized. A physical and continuous representation (the subject photographed) was sampled (the pictures), and these samples were converted to a medium (the sculpture) that is again continuous and allows for novel views to be synthesized. Using a set of pictures to synthesize a novel viewpoint is called *novel view synthesis*.

While this is an impressive start, novel view synthesis only recently, circa 2020 [1], became ready for widespread adaption. In the meantime, a theoretical framework surrounding light was built that would prove essential for novel view synthesis.

## 2.1 Plenoptic function

In 1991, Adelson and Bergen asked: "What information about the world is contained in the light filling a region of space?" [12]. To answer this question, they introduced the *plenoptic function*, which is not a single function but an umbrella term for all its variants. A specific plenoptic function is typically referred to using the number of dimensions of its domain, for example, a very complete and detailed plenoptic function is the 9-dimensional plenoptic function:

$$I = L(\mathbf{x}, \lambda, t, \mathbf{s}, \theta, \phi) \tag{2.1}$$

Figure 2.1: An unfinished photo sculpture. — Source: George Eastman Museum

with $I \in \mathbb{R}_{\geq 0}$ the intensity of light, $\mathbf{x} \in \mathbb{R}^3$ a point in space, $\lambda \in \mathbb{R}_{\geq 0}$ a wavelength, $t \in \mathbb{R}_{\geq 0}$ a point in time, $\mathbf{s} \in [0, 2\pi) \times [0, \pi)$ a 3D-direction, and, $\theta \in [0, \pi)$ the polarization and $\phi \in [0, 2\pi)$ the phase of light. The plenoptic function thus describes at each point in space and time the properties of the light present there. The human eye cannot resolve the polarization or phase of light, additionally, only the activation of the 3 types of cones can be perceived, not the wavelengths directly. Therefore, for time-independent representations, the more commonly used plenoptic function is the 5D-plenoptic function:

$$I = L(\mathbf{x}, \mathbf{s}) \tag{2.2}$$

A single 5D-plenoptic function can represent a black-and-white scene, while three can do so for a colored version.

In literature, the plenoptic function is often not mentioned or used directly, instead an adjacent concept/term is used that is some form of a simplified plenoptic function. Which term is used depends on the context and the field, for example, when capturing the plenoptic function 'light field' is a popular term, and when talking about the machine learning aspect 'radiance field' is popular, with numerous other terms existing. For this work, the distinction between these terms is irrelevant.

## 2.2 Differentiable rendering

Early novel view synthesis techniques in computer graphics [13] were, in essence, elegant interpolation techniques that necessitate an extremely dense camera plane. Without such a dense camera plane, a novel view synthesis technique must

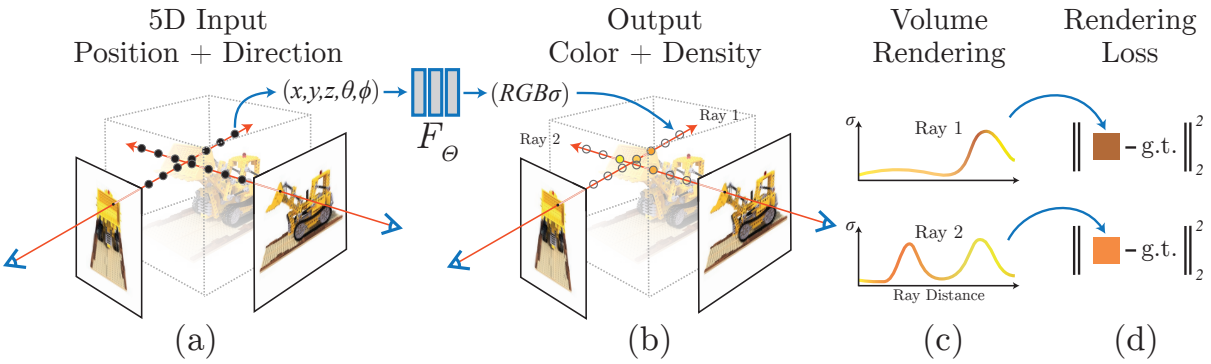



Figure 2.2: An overview of NeRF. — Source: Mildenhall et al. [1]

understand certain aspects of the captured scene. This type of novel view synthesis typically uses training images and their respective camera properties to learn about the scene. Since the advent of (modern) machine learning, this once elusive idea has come to fruition using *differentiable rendering*.

Rendering defines a rendering function $r$ that produces an image $I$ given a camera $c$, and a model $M$, consisting of model parameters $p_i$, i.e. $I = r(M, c)$. Each $I$ has a corresponding ground truth image $I_{\text{gt}}$ which can be used to define a loss using a loss function $L$. Minimizing this loss using a gradient-based optimization technique requires calculating the loss's partial derivative with respect to the model parameters. For example, when $L$ is the squared error, this amounts to:

$$\frac{\partial}{\partial p_i} L(I, I_{\text{gt}}) = \frac{\partial}{\partial p_i} (I - I_{\text{gt}})^2 = 2(I - I_{\text{gt}}) \frac{\partial}{\partial p_i} r(M, c) \tag{2.3}$$

In general, as is the case here, $r$ must be differentiable to achieve this. If this is the case, we speak of *differentiable rendering*. Differentiable rendering functions are carefully designed to maneuver around numerous pitfalls, e.g. discontinuities.

## 2.3 Recent advances

There are currently two competing, state-of-the-art, novel view synthesis techniques. They are based on two completely different rendering functions, however, both are differentiable.

### 2.3.1 NeRF

In 2020, a seminal paper introduced Neural Radiance Fields (NeRF) [1] which kick-started a new domain within novel view synthesis research. There have been multiple iterations of NeRF since, but the general idea is to use a neural network to learn a continuous scene representation and sample it during volumetric rendering. Each pixel casts a ray that samples multiple points, with a color and density, to calculate a pixel's color. Figure 2.2 provides an overview.

After some time, Mip-NeRF 360 [4] emerged as the go-to for state-of-the-art quality within NeRF, as it can cope with complex geometries and lighting effects. However, this high quality comes at the cost of extensive training times ranging from hours

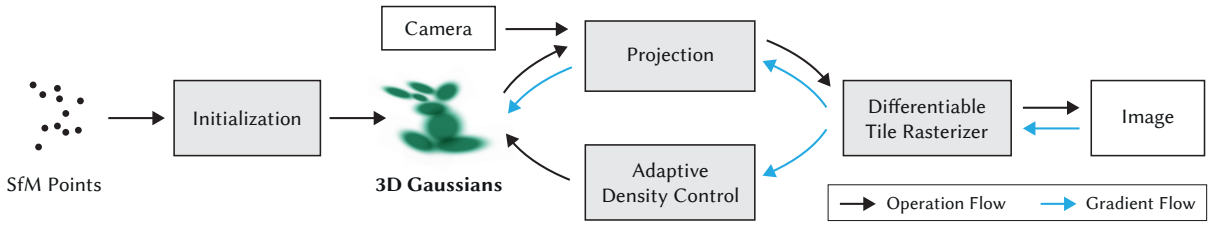



Figure 2.3: An overview of 3DGS. — Source: Kerbl et al. [2]

to days, and inference times measured in seconds for all but the simplest scenes. Recognizing that, for most practical use cases, inference speed and training times are more important than image quality, Instant NGP [14] presented a way to improve the training and inference times by two orders of magnitude at the cost of Mip-NeRF 360's exceptional quality.

Recently, during ICCV 2023, two best paper finalists, Zip-NeRF [5] and Tri-MipRF [15], presented new methods that achieve Mip NeRF-360 [3] quality with Instant NGP [14] speeds.

### 2.3.2 Gaussian kernels

In July of 2023, Kerbl et al. [2] released their seminal paper where they presented 3D Gaussian Splatting (3DGS). This marked the first time the novel view synthesis 'trifecta' was hit: real-time rendering (100+ fps for unbounded scenes), reasonable training times (minutes to at most an hour), and photorealism.

3DGS starts by extracting camera properties and a sparse point cloud from the training images using Structure-from-motion [16]. The points in the sparse point cloud are replaced by splats, which are feature-rich primitives that have a position, shape, rotation, opacity, and view-dependent color. This scene representation, i.e. the collection of splats, is trained to accurately reconstruct the training images using 3DGS's differentiable rasterizer. The amount of splats is increased during training in areas where necessary in a process called 'Adaptive Density Control'. Figure 2.3 provides an initial overview. A further, more complete, overview of 3DGS is presented in Chapter 3 as this is the technology this thesis resolves around.

Splats in 3DGS are localized in space using Gaussian kernels, building on a framework presented over 20 years ago by Zwicker et al. [17]. More recently, in 2018 and 2020, Ruben Verhack used Gaussian kernels for novel view synthesis to create Steered mixture-of-experts (SMoE) [18, 19]. Martijn Courteaux, who worked alongside Ruben Verhack, further developed SMoE into a real-time novel-view synthesis application [1] making it very reminiscent of 3DGS.

### 2.3.3 Comparison

NeRF's main advantage stems from its use of neural networks. Neural networks seem more capable of adapting to a scene, for example, NeRF has shown more prowess than 3DGS in: complex lighting situations [20, 21], under constructed in-the-wild

[1] https://www.youtube.com/watch?v=TKuu8Di76RQ&list=PLn3xmXYrC53Cdl5nI_LfCB6j6zVdc92Xi

photographs [21], and few-shot scene reconstruction [22]. The use of neural networks is also a downside as their black-box nature makes them inherently unlocalized, compare this to Gaussian kernels where each parameter has a clear meaning and each kernel is an expert in a region of space. This leads to works like kiloNeRF [23] and Mega-NeRF [24] (cf. Vast Gaussians [25]) to segment physical space into regions, each with its own MLP.

The main advantage of Gaussian kernel-based methods, and arguably the most important difference compared to NeRFs, is their use of a rasterization engine, compared to NeRFs ray marching approach. Rasterization is at the heart of why 3DGS can achieve such high inference speeds while NeRF, with over 4 years of research, still can not match the speed presented by 3DGS. Mobilenerf [26] recently showed some promise in consolidating NeRF with a rasterization engine at inference time, however, training still uses ray marching which leads to reported training times of around a day.

In terms of file size, a 1 GB uncompressed 3DGS scene can have a NeRF equivalent of only 35 MB [2]. However, although NeRFs have a leg up on their starting size compared to 3DGS, further compressing a NeRF is considerably more challenging due to its black-box nature. NeRFs' only real compression option is a change of architecture to a more 'file-size-friendly' one, e.g. BiRF [27].

## 2.4 Applications

In creative works, novel synthesis is used to create cinematic effects, for example, the famous 'bullet time' effect from The Matrix. More recently, both NeRF [3] and 3DGS [4] have appeared in popular music videos. Outside creative works, 3DGS is used to create flythroughs to show spaces [5], while NeRFs are used as a telepresence tool by Google with their Project Starline [6]. Google also uses novel view synthesis for Google Maps Immersive View [7], which allows you to fly around in bars, restaurants, or landmarks. Google's involvement with the applications of novel view synthesis is not surprising as they have been spearheading NeRF-related technology with involvement in (non-exhaustive): NeRF [1], NeRF in the Wild [21], Mip-NeRF360 [3], and Zip-NeRF [5].

Popular 3D creation tools, such as Unity and Unreal Engine, are able to use NeRF and Gaussian Splatting through third-party plugins [8] [9] [10], paving the way for the widespread adoption of novel view synthesis.

[2] https://radiancefields.com/3d-gaussian-splatting-but-smaller
[3] https://radiancefields.com/zayn-music-video-premieres-using-nerfs/
[4] https://radiancefields.com/3dgs-appears-in-usher-music-video/
[5] https://lumalabs.ai/flythroughs
[6] https://blog.google/technology/research/project-starline/
[7] https://blog.research.google/2023/06/reconstructing-indoor-spaces-with-nerf.html
[8] https://github.com/aras-p/UnityGaussianSplatting
[9] https://github.com/xverse-engine/XV3DGS-UEPlugin
[10] https://lumaai.notion.site/Luma-Unreal-Engine-Plugin-0-41-8005919d93444c008982346185e933a1

# 3

# Gaussian splatting

This work studies compression techniques within Gaussian splatting. To understand why and how the proposed techniques work, some background is required on the inner workings of 3DGS. This chapter solely provides information that will become relevant during later chapters and is not a full account of 3DGS.

## 3.1 3D Gaussians

3DGS scenes consist of a collection of splats. Each splat affects the transparency of the space around it, emitting the splat's color to the camera proportional to the transparency it absorbs. The degree to which a splat absorbs the transparency and emits light at a point in space $\mathbf{x} \in \mathbb{R}^3$ is proportional to a 3D Gaussian:

$$\mathcal{G}(\mathbf{x}) = \exp(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})) \tag{3.1}$$

where $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ denote the splat's center and covariance matrix respectively. Multivariate Gaussians exhibit affine invariance, i.e. the affine transformation of a multivariate Gaussian is again a multivariate Gaussian. The new center and covariance matrix of a multivariate Gaussian after an affine transformation is denoted as $\boldsymbol{\mu}'$ and $\boldsymbol{\Sigma}'$ respectively.

The first transformation of interest is the viewing transform, which, within 3DGS, is an affine transformation $\mathbf{W}$. The second transformation of interest is the projective transformation, which is not affine, as line parallelism is not preserved. The projective transformation is therefore approximated as an affine transformation using its Jacobian $\mathbf{J}$. Applying both the viewing and projective transformation results in the following covariance matrix:

$$\boldsymbol{\Sigma}' = \mathbf{J}\mathbf{W}\boldsymbol{\Sigma}\mathbf{W}^T\mathbf{J}^T \tag{3.2}$$

After removing the third row and column from $\boldsymbol{\Sigma}'$, the covariance matrix of the projected 2D Gaussian in screen space $\boldsymbol{\Sigma}^{2D}$ is found. A similar procedure can be done to compute the 2D center of the splat $\boldsymbol{\mu}^{2D}$. For a pixel corresponding with a screen position $\mathbf{x}^{2D} \in \mathbb{R}^2$, the 2D Gaussian corresponding to the projected 3D Gaussian is defined as follows:

$$\mathcal{G}^{2D}(\mathbf{x}^{2D}) = \exp(-\frac{1}{2}(\mathbf{x}^{2D}-\boldsymbol{\mu}^{2D})^T(\boldsymbol{\Sigma}^{2D})^{-1}(\mathbf{x}^{2D}-\boldsymbol{\mu}^{2D})) \tag{3.3}$$

Whether a Gaussian, position, mean or covariance matrix is 2D or 3D will not always be explicitly mentioned as this is typically evident from the context.

Gaussians are easily differentiable which is necessary as 3DGS is a differentiable rendering technique. This work only explains aspects of the forward pass of the rasterizer. The backward pass, where the change in a model parameter is related to a change in the rendered image, is not discussed.

## 3.2 Alpha compositing

A pixel's color is the result of combining multiple splats' colors via alpha compositing. Each splat corresponds with a layer (and vice-versa) in the alpha compositing process, where the layers are ordered according to the depth of their associated splats. A layer's pixel, located at the screen space position $\mathbf{x}$, has an associated opacity $\alpha$:

$$\alpha = o \cdot \mathcal{G}^{2D}(\mathbf{x}) \tag{3.4}$$

where $o$ is the location-independent base opacity of a splat. Additionally, the opacity's dependence on $\mathbf{x}$ is omitted for notational simplicity (in line with 3DGS). The $\alpha$ values of previous layers determine the extent to which the next layer is occluded. The yet undetermined fraction of a pixel's color is called the *transmittance* and after the $k$ first layers, it is defined as:

$$T_k = \prod_{j=0}^{k-1}(1 - \alpha_j) \tag{3.5}$$

where, again, the dependence on $\mathbf{x}$ is omitted for notational simplicity. Putting everything together, a pixel's final color, after alpha-composited $K$ layers, is defined as:

$$\mathbf{c}(\mathbf{x}) = \sum_{k=0}^{K-1} T_k \alpha_k \mathbf{c}_k + T_K \mathbf{c}_{\text{bg}} \tag{3.6}$$

where $\mathbf{c}_{\text{bg}}$ represents the background color. Typically, and assumed in this work, the background is black ($\mathbf{c}_{\text{bg}} = \mathbf{0}$), resulting in the omission of the term $T_K \mathbf{c}_{\text{bg}}$. More recently, support has been added for a random background color [1] which is ignored in this work.

Theoretically, $K$ is expected to be in the same order of magnitude as the number of splats, as Gaussians span all of space. However, practically, most splats are not considered as their contribution to the given pixel is negligible. Additionally, once $T_k$ becomes very small ($T_k \leq 0.0001$ is used by 3DGS) the alpha compositing process is stopped prematurely. As a result, $K$ is typically only around 50 (see Figure 6.3 later on), instead of the hundreds of thousands to millions it should theoretically be.

## 3.3 Spherical Harmonics

The alpha compositing process uses a splat's color $\mathbf{c}_k$. This color changes in function of the camera's position using spherical harmonics. Spherical harmonics are special functions defined on the surface of a sphere that, together, form an orthonormal

[1] https://github.com/graphdeco-inria/gaussian-splatting/commit/f11001b46c5c73a0a7d553353c898efd68412abe

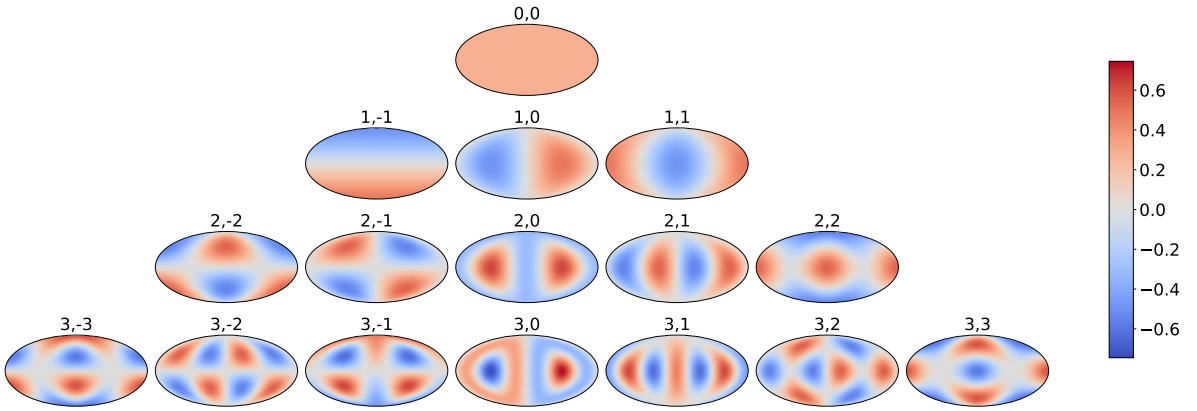



Figure 3.1: All real spherical harmonics with $l \leq 3$. Labels are of the form $(l, m)$. —Source: Image by the author.

basis. In the same way that a Fourier series decomposes a periodic function into the sum of complex exponentials, spherical harmonics decompose any (twice-differentiable) spherical function into a sum of spherical harmonics. A real spherical harmonic is denoted as $Y_{l,m}$, where $l \in \mathbb{N}_0$ represents the degree and $m = -l, -l+1, ..., l$ the order. Figure 3.1 shows all real spherical harmonics with a degree $\leq 3$ using the Mollweide projection, an equal-area projection method. There are minor conventional differences between different works using spherical harmonics (e.g. the Condon-Shortley Phase[2]) which can affect visualizations such as Figure 3.1, but has no further impact.

The decomposition of a spherical function $L(\theta, \phi)$ using spherical harmonics is defined as follows:

$$L(\theta, \phi) = \sum_{l,m} L_{l,m} Y_{l,m}(\theta, \phi) \tag{3.7}$$

where $L_{l,m}$ is $Y_{l,m}$'s coefficient, and often simply referred to as its *lighting coefficient*. As it is impractical to iterate over spherical harmonics using the degree and order, each $l, m$ pair is often mapped to a unique scalar. The mapping chosen for this work iterates over the spherical harmonics row-by-row, left-to-right as seen in Figure 3.1, and is defined as follows:

$$(l, m) \rightarrow l(l+1) + m + 1 \tag{3.8}$$

Within 3DGS, three color channels (RGB) are defined using spherical harmonics, resulting in a splat's color being fully specified by three sets of lighting coefficients. To convert these lighting coefficients to an actual color, only the position of the splat and camera is required, meaning that a splat is monochromatic within a single frame so that only the opacity changes between pixels. Visualizations, such as Figure 3.1, are not renders of splats themselves, but rather visualize the view-dependent properties of a splat, answering questions such as: "When looked at from this angle, what intensity does $Y_{1,0}$ have?".

It is often handy to specifically refer to, or single out, the first spherical harmonic $Y_{0,0}$. This spherical harmonic is unique as it is view-independent, and not zero-meaned. All spherical harmonics beside $Y_{0,0}$ are called the 'AC spherical harmonics'.

[2] https://mathworld.wolfram.com/Condon-ShortleyPhase.html

Conversely, 'DC' refers to the first spherical harmonic.

Finally, it should be noted that spherical harmonics are commonly used within computer graphics [28]. They are certainly not unique to Gaussian Splatting, with other novel view synthesis techniques [14, 29] also using them to introduce view-dependent properties.

# 4

# Compression

The scene Truck from the Tanks and Temples benchmark [30], is chosen as the reference scene for all compression techniques, visualizations, and discussions. 3DGS provides a pre-trained version (30,000 iterations) of the scene Truck [1]. This uncompressed scene is the starting point of this thesis and is 630 MB large.

This chapter explores the factors contributing to the scene's large size, the potential reduction in size achievable through compression, the current state-of-the-art, and the motivations driving the need for compression.

## 4.1 Expectations

An initial question might be: *How small can we expect this 3DGS scene to become after compression?* To answer this informally, consider only 3DGS's capabilities to re-render a training image. This makes the scene just another lossy representation of the input images that fulfills the same role as an image or video codec. An H.265 encoded video of the training images with a similar quality as the 3DGS scene, is only 5 MB large. While this comparison is informal and ignores numerous nuances, it sets the tone that the opportunity for compression is enormous. To a lesser extent, the existing 3DGS compression literature echoes the same message.

## 4.2 Format

Most attributes (position, opacity, and lighting coefficients) are saved directly. The covariance matrix $\mathbf{\Sigma}$ is an exception as it is decomposed, to ensure it has a physical meaning[2], into a scaling matrix $\mathbf{S} \in \mathbb{R}^{3\times3}$ and a rotation matrix $\mathbf{R} \in \mathbb{R}^{3\times3}$:

$$\mathbf{\Sigma} = \mathbf{RSS}^T\mathbf{R}^T \tag{4.1}$$

The scaling matrix is fully defined by a scaling vector $\mathbf{s} \in \mathbb{R}^3$, and the rotation matrix is fully defined by a quaternion $\mathbf{q} \in \mathbb{R}^4$. It is these two vectors, $\mathbf{s}$ and $\mathbf{q}$, that the model learns and stores.

[1] https://repo-sam.inria.fr/fungraph/3d-gaussian-splatting/datasets/input/tandt_db.zip

[2] Only covariance matrices that are positive semi-definite have an underlying physical meaning.

| Attribute class | Short-hand | Definition | # attributes | Uncompressed size (bytes) |
|---|---|---|---|---|
| Position | $\boldsymbol{\mu}$ | Splat center | 3 | 12 |
| Scale | $\mathbf{s}$ | Eigenvalues of $\boldsymbol{\Sigma}$ | 3 | 12 |
| Rotation | $\mathbf{q}$ | Quaternion form of eigenvectors of $\boldsymbol{\Sigma}$ | 4 | 16 |
| Opacity | $o$ | Base opacity | 1 | 4 |
| Color | $\{\mathbf{L}_{l,m}\}$ | 3 sets of lighting coefficients ($l \leq 3$) | 48 | 192 |
| Total | | | 59 | 236 |

Table 4.1: An overview of the attribute classes within 3DGS. Uncompressed attributes are stored as single precision floats.

Table 4.1 provides an overview of all attributes and their impact on the number of bytes per splat. The normal of a splat is ignored throughout this work as it is always saved as zero for the studied scenes. This table highlights that lighting coefficients are the main culprit behind the high uncompressed splat size, making it a primary focus of this thesis. The other primary focus, which will be addressed first, is the number of splats.

## 4.3 Number of splats

3DGS implements a simple heuristic to keep the number of splats under control during training by setting all splat opacities to 1% (or lower) every so often. The hope is that only the essential splats will recover their opacity through gradient descent, allowing the other splats to be removed after some time. However, even with this technique, scenes still have *millions* of splats. For example, the Truck scene has 2.5 million splats. 3DGS can handle such a high load fairly well as it was designed under the assumption of a high load of small splats. Still, the question arises if such a high number of splats is necessary. To that end, an unofficial reimplementation of 3DGS [3] claims to have achieved similar visual quality with only a quarter to a fifth of the splats.

### 4.3.1 Motivation

Reducing the number of splats has multiple upsides:

1. **Compression**
   An uncompressed 3DGS scene is simply an attribute-rich point-cloud. Reducing the number of splats linearly decreases the uncompressed file size. Additionally, reducing the number of splats synergizes remarkably well with attribute-based compression techniques, as their effects are combined multiplicatively. For example, if we can reduce the number of splats by a factor of 5, and the average splat size by a factor of 5, then a compression factor of 25 is achieved.

[3] https://github.com/wanmeihuali/taichi_3d_gaussian_splatting

2. **Memory footprint**
   The memory requirements (mainly VRAM) of 3DGS during training and inference can be quite high depending on the desired quality. Additionally, 3DGS's performance has shown evidence of being memory-bound on high-end GPUs [9], with the GitHub repository of 3DGS [4] stating that 24 GB of VRAM is needed during training to reach their presented results. The number of splats is a major contributor to the training and inference memory requirements. 3DGS's inference memory footprint has already caused the most popular Gaussian splatting Unity implementation [5] to implement compression methods.

3. **Training/Inference speed**
   The fewer splats, the faster the render pass is executed during training and inference. This is the second reason, besides 3DGS's memory footprint, why the compression literature [9, 7, 10, 6] has been able to boast such impressive frame rate improvements. As it turns out, the easiest way to increase the performance is to reduce the number of splats. The same goes for this work where frame rates are substantially improved.

4. **Faster processing**
   Operations on 3DGS scenes often boil down to operations on splats. Concerning compression, this might be quantization or energy compaction of spherical harmonics, both of which are done in this work. Consequently, removing splats is the logical first step in a 3DGS encoder as it reduces the file size and lessens the workload for future operations. The number of splats can even become prohibitive for some operations, typically for those with superlinear time complexity. A compression literature example of this is Morgenstern et al. [7] who, due to the high number of splats, had to invent a novel, highly parallelized sorting strategy, to be able to apply their desired compression technique.

### 4.3.2 State-of-the-art

Most existing compression literature uses some method to reduce the number of splats [9, 7, 10, 6]. Morgenstern et al. [7] observe that removing 30% of the lowest opacity splats has negligible impact on the peak signal-to-noise ratio (PSNR), and therefore did just that. Similarly, Niedermayr et al. [9] remove all splats with zero sensitivity in the color parameters, resulting in 15% of the splats being removed. Fan et al. [10] define and calculate the 'Global Significance' of a splat, which is based on its volume and opacity, afterward, splats with a low Global Significance are removed. Lee et al. [6] propose a learnable mask that affects both the transparency and volume of a splat, i.e. as a splat gets more 'masked' its opacity lowers and volume shrinks. The masking of transparency has merit, as shown by Morgenstern et al. [7], and the volume masking shares the same notion as presented by Fan et al. [10], where a splat has a higher Global Significance score the larger the volume it occupies. Lee et al. [6] state:

> *"The scale attribute of each Gaussian determines its 3D volume, which is then reflected in the rendering process. Small-sized Gaussians, due to their minimal volume, have a negligible contribution to the overall rendering quality, often to the point where their effect is essentially imperceptible. In such cases, it becomes highly beneficial to identify and remove such unessential Gaussians."*

[4] https://github.com/graphdeco-inria/gaussian-splatting
[5] https://github.com/aras-p/UnityGaussianSplatting

While this sounds plausible, one of the contributions of this thesis is providing contradictory evidence to the above claim (see Section 6.1.1). Morgenstern et al. [7] also, indirectly, cast some doubt, showing that removing splats based on their volume has a real, immediate impact on PSNR.

## 4.4 Spherical Harmonics

The Truck scene uses all real spherical harmonics with $l \leq 3$ (shown in Figure 3.1). This leads to 16 coefficients per color channel. With 3DGS employing three color channels, this amounts to 48 attributes. Contrary to the number of splats, 3DGS makes no effort to reduce the number of spherical harmonics.

### 4.4.1 Motivation

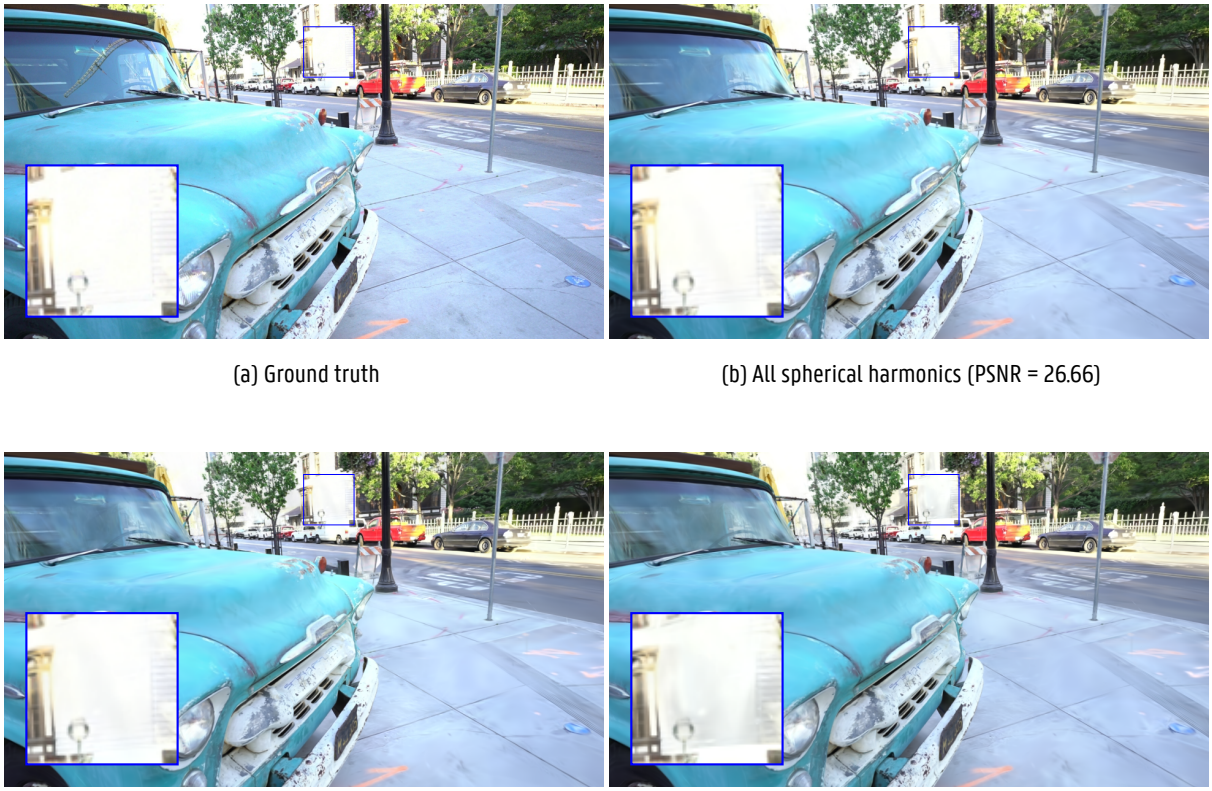
Simply not using any AC spherical harmonics, results in 45 fewer attributes. This reduces file sizes by over 76% as all other attributes (DC color, position, rotation, scale, and opacity) only use 14 attributes together. It should therefore come as no surprise that initial web-based viewers [6] [7] [8] [9] often do no use AC spherical harmonics at all. The flip side is that all splats become monochromatic, hampering the model's capabilities to model lighting phenomena such as reflections. This trade-off does not have to be made if the lighting coefficients are compressed, giving the primary motivation for spherical harmonics compression.

Models trained without spherical harmonics can still attain a very high quality, however, training a model first with spherical harmonics and then removing them afterward is a deceptively hard problem. The obvious solution, removing all AC lighting coefficients, seems optimal due to AC spherical harmonics being zero-meaned. However, this naive approach produces visual artifacts, as shown in Figure 4.1d. The proposed compression method provides a better way to remove spherical harmonics, as shown in Figure 4.1c. This highlights that developing techniques to compress lighting coefficients can result in more than just compression benefits, for example, here it provides a general method to change the spherical harmonics used post-training.

### 4.4.2 State-of-the-art

Two works in literature do not use spherical harmonics [7, 6]. Morgenstern et al. [7] states no reason, but I suspect this is due to AC lighting coefficients showing poor spatial correlation, which their proposed technique heavily relies on. Lee et al. [6] replaces spherical harmonics with hash grids and a small MLP. This technique is borrowed from the realm of NeRF [14] and is not discussed further as this work focuses on spherical harmonics. Additionally, it fundamentally alters the rendering pipeline, causing an approximate 17% drop in frame rate.

[6] https://gsplat.tech/
[7] https://github.com/antimatter15/splat
[8] https://github.com/dylanebert/gaussian-viewer
[9] https://github.com/laanlabs/metal-splats



(a) Ground truth

(b) All spherical harmonics (PSNR = 26.66)

(c) AC spherical harmonics removed: proposed (PSNR = 24.50)

(d) AC spherical harmonics removed: naive (PSNR = 23.73)

Figure 4.1: Comparison of the proposed and naive AC spherical harmonics removal method. The ground truth image and the uncompressed model's render are shown as references. — Source: Images by the author

Of the literature that compresses lighting coefficients, two works [9, 8] directly apply vector quantization on the given lighting coefficients. Only one work, LightGaussian [10], first tries to find an alternate set of lighting coefficients through 'distillation', a technique where the rendered images of the original model are used to train a model with fewer spherical harmonics.

# 5

# Compression framework

Gaussian splatting is a new field, with its compression literature still in its infancy. Currently, there is no established methodology or framework on how compression should be approached, how techniques should be evaluated, or how they should be presented. To provide a baseline, this chapter discusses the different classes of compression techniques, how the quality of a scene should be evaluated, and a splat's contribution, one of the most important metrics in talking about compression.

## 5.1 In-loop vs out-of-loop

I identify two, largely mutually exclusive, paradigms in which compression techniques fall:

1. **In-loop**
   The first paradigm alters the training loop and thus necessitates control over the training process. An example of an in-loop technique is changing the training loss function to promote compressibility.

2. **Out-of-loop**
   The second paradigm does not embed itself in the training loop. Instead, it can leverage the training process to achieve better results without relying on it entirely. For example, one can remove splats from an already trained model and then perform a few training iterations to correct any introduced errors.

While both approaches have their place, the out-of-loop technique is easier to iterate over, comprehend, and reason on. Hence, I focus solely on out-of-loop techniques for this initial work. Most out-of-loop techniques benefit from additional training steps as the scene representation is altered during compression. In literature, this is referred to under different names, e.g. fine-tuning [9] or co-adaptation [10]. Consequently, evaluating an out-of-loop technique's performance *after* the additional training iterations makes sense. However, I believe this should not be the only point at which an algorithm is evaluated as the extra training iterations act as a black-box that obscures the actual behavior of a technique. This is not done in literature, which I suspect is the primary reason I could find contradictory evidence to Lee et al. [6]'s claim that small splats have a negligible contribution to the overall rendering quality. An in-depth account of this contradictory evidence is provided in Section 6.1.1.


(a) Image from an in-distribution camera.

(b) Image from an out-of-distribution camera.

Figure 5.1: Two images from novel viewpoints of the scene Truck.

## 5.2 Query density function

A minor oversight in comparing H.265 and uncompressed 3DGS scenes (Section 4.1), is 3DGS's need for camera information to reconstruct the original images. While this oversight has a negligible impact on the size of the 3DGS scenes, it does raise an interesting observation that if the information about the cameras is not given, determining the quality of a 3DGS scene becomes markedly more difficult, subjective, and ill-defined. This might feel counterintuitive, as 3DGS is a *novel* view synthesis method, i.e. you do not know all the camera properties in advance. However, by capturing a scene we implicitly convey the intended usage, for example, compare the two images shown in Figure 5.1. Both these images are taken from a novel viewpoint, but it is clear that only one camera falls within the implicitly defined collection of 'expected cameras'.

Coupling back to compression, only the quality of in-distribution cameras is relevant. More formally, I define the concept of in- and out-distribution samples using a probability density function, namely, the query density function $Q_c$. This function takes the same arguments as the plenoptic function and defines the relative likelihood of a plenoptic sample. The pixels of in-distribution cameras are possible samples of $Q_c$, while out-distribution cameras' pixels are impossible samples of $Q_c$. Having a grasp on $Q_c$ is extremely valuable as it is instrumental for any selective introduction of loss during compression and is therefore used (implicitly) at nearly every compression step throughout this work.

As mentioned before, $Q_c$ is not explicitly defined. Therefore, for this initial work, two assumptions are made:

1. **The training cameras are available.** This is not a given as currently the most popular way to share 3DGS scenes is using a point cloud format containing no camera information. A manual remedy is to resample $Q_c$ manually by walking around the scene. A more automated approach is not presently available and is left for future work.

2. **The training cameras' pixels are representative samples of $Q_c$.** This coincides with the observation that there are 'right' ways to query a scene, as shown in Figure 5.1. While this is evidently true in a broad sense (uncaptured

regions are not expected to be queried), the more nuanced implication is that if something is captured often, it is also expected to be queried often. This might not hold if, for example, a region has more samples due to its level of detail requiring more samples. Besides this, the assumption is fairly forgiving due to the rectilinear propagation of light.

## 5.3 Splat contribution

The term 'a splat's contribution' is informally used in literature [6, 10, 7] to convey how much a splat 'draws' of the total scene for a given $Q_c$. For example, a large splat part of the truck's front tire, visible in Figure 5.1a, has a high contribution, while an occluded splat that is never visible has no contribution. To the best of my knowledge, previous works were only interested in a splat's contribution as a qualitative, not a quantitative, measure. This work breaks this precedence and defines the $k$-th splat's contribution:

$$I_k = \int_\Omega Q_c(\boldsymbol{\omega}) T_{i_k} \alpha_{i_k} \, d\boldsymbol{\omega} \tag{5.1}$$

where $\Omega$ denotes the domain of $Q_c$ which coincides with the domain of the underlying plenoptic function. $T_{i_k}$ and $\alpha_{i_k}$ respectively refer to the transmittance and opacity of the $k$-th splat during the alpha compositing process (see equation 3.6) for the ray implicitly defined by $\boldsymbol{\omega}$.

A splat's contribution thus depends on $Q_c$ and the non-color attributes of all splats (position, rotation, scale, and opacity) in a complicated non-linear fashion. Luckily, using the aforementioned assumptions on $Q_c$, a splat's contribution for a single frame is easily and accurately estimated by modifying the alpha compositing process of the 3DGS rasterizer. The rasterizer renders a frame for a given camera, therefore, first estimate the contribution of a splat to a camera $\mathbf{s}$ as follows:

$$I_k(\mathbf{s}) \approx \frac{1}{WH} \sum_{\mathbf{x}} T_{i_k} \alpha_{i_k} \tag{5.2}$$

where $W$ and $H$ respectively denote the width and height of the rendered image. Now consider $N$ cameras to arrive at an estimation for the contribution of a splat:

$$I_k \approx \frac{1}{N} \sum_{\mathbf{s}} I_k(\mathbf{s}) = \frac{1}{N} \sum_{\mathbf{s}} \frac{1}{WH} \sum_{\mathbf{x}} T_{i_k} \alpha_{i_k} \tag{5.3}$$

The fact that this is an estimation will be omitted from this point on as it is implicitly understood and only complicates discussion/notation. Furthermore, the exact changes made to the rasterizer to efficiently calculate all $I_k$ values are not discussed as a very similar discussion is done later for a more complex metric (see Section 6.2.1).

Ignoring some complications surrounding the background also being a contributor and the early stopping of the alpha compositing process, the contributions are normalized at both the scene and camera level:

$$\sum_k I_k = 1 \qquad\qquad \sum_{\mathbf{s}} I_k(\mathbf{s}) = 1 \tag{5.4}$$

# 6

# Splat removal

## 6.1 Analysis

The intuition behind removing splats is the suspicion that splats have varying degrees of importance. Some splats might be very important and should not be removed, while others might be so unimportant that they can be removed with minimal loss in visual quality. The key question determining splat removal's effectiveness as a compression method therefore becomes: "To what degree do splats differ in importance?". One way to define the importance of a splat is as its contribution. A splat whose color is responsible for a large portion of the final synthesized image is considered important, as its removal would fundamentally change how the image was composed. Figure 6.1 shows the cumulative contribution for different fractions of the highest contributing splats. Using this figure, two observations are made:

1. There is a steep initial climb in the cumulative contribution, indicating that a small number of splats make up most of the final image. Only 1% and 10% of the highest contributing splats are respectively responsible for over 50% and 85% of the total contribution.
2. A cumulative contribution of nearly 100% is achieved after only considering 45% of splats. The majority of the splats therefore have little to no impact on the scene.

These two observations show the immense potential of splat removal as a compression technique.

### 6.1.1 Size based metrics

Of the four works in the literature that offer a method for removing splats, two works [9, 7] mention this only briefly, as their primary focus lies elsewhere. The other two works [10, 6] focus significantly on removing splats and use a size-based metric/technique to do so. These works operate on two assumptions:

1. A splat's size is a good proxy for its contribution.
2. Removing low contributing splats has minimal effect on the scene's quality.

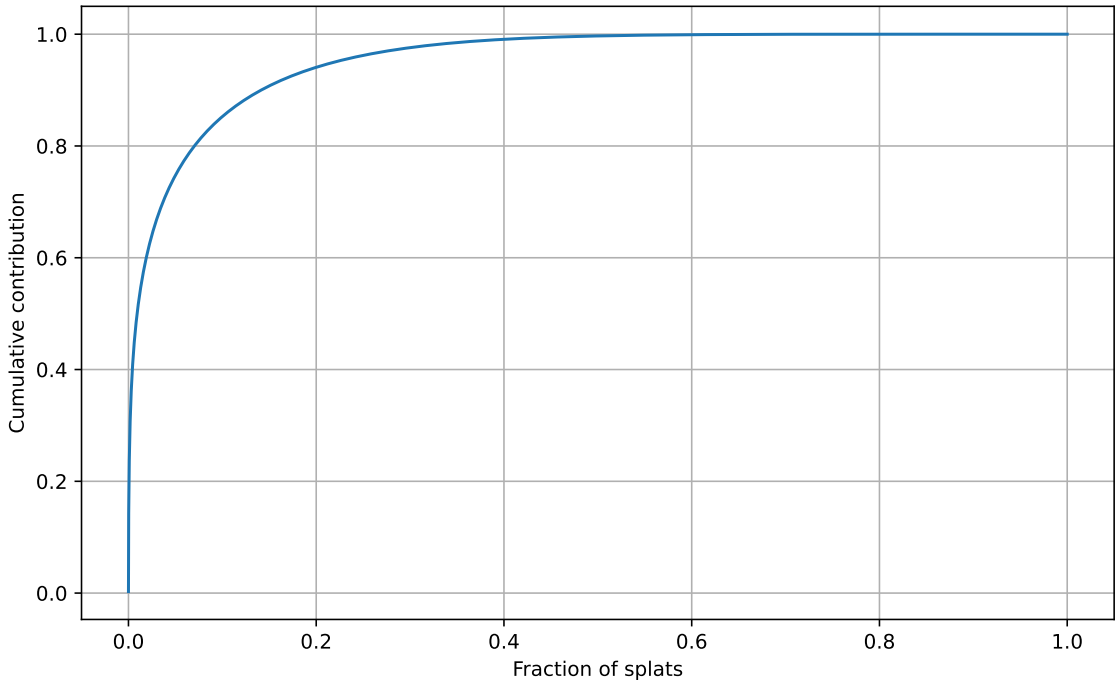



Figure 6.1: Cumulative contribution of the fraction of the highest contributing splats for the scene Truck.

However, neither work provides evidence for the above assumptions beyond their achieved compression results. While the second assumption is trivially true so long the contribution is near zero, the first assumption is not as clear-cut. Therefore, I calculate both the splats' contributions and several size-based metrics to check the soundness of the first assumption. A good proxy for a splat's contribution should be able to select $n$ splats such that their cumulative contribution is close to the highest cumulative contribution achievable using $n$ splats. Figure 6.2 shows how a myriad of size-based factors (dotted or dashed lines) and other factors not related to size (full lines) perform. The vector $\mathbf{s}$ is the scaling vector and fully defines a splat's shape. For example, a splat's volume [1] is proportional to $\mathbf{s}_0\mathbf{s}_1\mathbf{s}_2$. To understand this figure, consider that as soon as a curve goes straight to the top-right corner, it has lost any resolving power as it behaves equivalent to randomly selecting splats. An example is the height of a splat, which has decent success in determining some of the highest contributing splats as 80% cumulative contribution is reached with only 40% of the splats. However, additional splats beyond the 40% mark are chosen quasi-randomly, as the curve goes straight to the top-right. This means the height of the splat is a useless metric to order splats according to their contribution for the remaining 60% of splats. The shown size-based metrics in Figure 6.2 are plausible metrics I propose myself, except $\gamma(\mathbf{\Sigma})$ which was proposed by Fan et al. [10]. Surprisingly, $\gamma(\mathbf{\Sigma})$ performs worse than simpler size-based metrics and even performs worse than random in selecting up to 30% of splats as it goes below the 1st bisector in Figure 6.2.

If the goal is to remove splats in a near-imperceptible way, then the relevant metric is how quickly the cumulative contribution gets to almost 100%. Therefore, I find that **size-based metrics show extremely poor resolving power in determining splats with negligible contributions**, with trivial metrics, such as the height of the splat, performing equivalently or better.

[1] Splats in theory do not have a volume, but colloquially, splats are often thought of as ellipsoids, as is the case here.

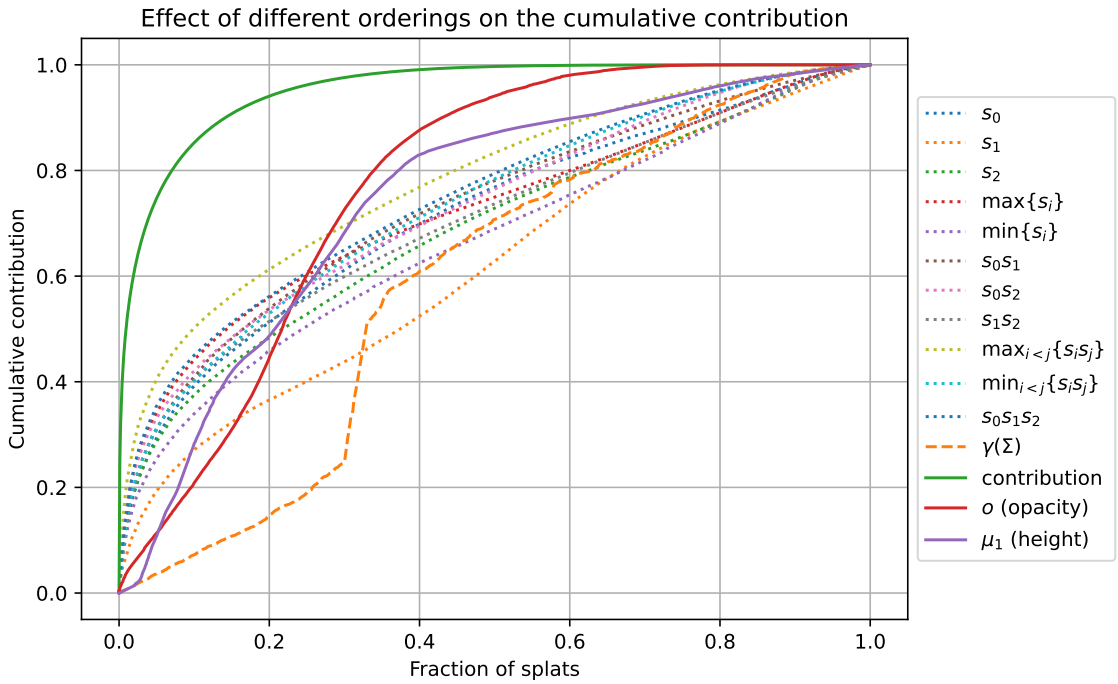



Figure 6.2: Comparison of different metrics on their effectiveness at ordering splats to quickly reach a high cumulative contribution.

The only two metrics that reach the near 100% mark early are (1) the contribution, which is definitionally optimal, requiring only 45% of splats, and (2) the opacity, requiring only 70% of splats. Note that this already provides a way to near-losslessly cull 55% or 30% of splats respectively. The ability of opacity to remove 30% of splats near-losslessly is the same observation Morgenstern et al. [7] made.

To summarize, size-based metrics as a proxy for a splat's contribution perform poorly at identifying low-contributing splats. This provides contradictory evidence to claims made by literature [6]. Additionally, complex size-based metrics suggested by literature [10] are found to perform worse than simple size-based metrics, even performing worse than random in some scenarios.

## 6.2 Effect of a splat's removal

An initial reflex might be to use a splat's contribution to determine whether or not a splat should be removed. A splat with a negligible contribution is a sufficient condition for its removal to have an insignificant impact on the visual quality, however, it is not a necessary one. For example, removing a red splat might reveal another red splat behind it, making its removal effect very minor, regardless of the contribution of the splat. Thus, the actual value we are interested in is how the pixel values of the rendered images change due to a splat's removal. Let therefore the cull difference $\mathbf{CD}_j(\mathbf{x})$ be the change

in a pixel's color due to the $j$-th splat's removal. $\mathbf{CD}_j(\mathbf{x})$ can be calculated as follows:

$$\begin{aligned}\mathbf{CD}_j(\mathbf{x}) &= \tilde{\mathbf{c}}_j(\mathbf{x}) - \mathbf{c}(\mathbf{x}) \\ &= \sum_{k=0}^{j-1} T_k \alpha_k \mathbf{c}_k + \frac{1}{1-\alpha_j} \sum_{k=j+1}^{K} T_k \alpha_k \mathbf{c}_k - \mathbf{c}(\mathbf{x}) \\ &= -T_j \alpha_j \mathbf{c}_j + \frac{\alpha_j}{1-\alpha_j} \sum_{k=j+1}^{K} T_k \alpha_k \mathbf{c}_k\end{aligned} \tag{6.1}$$

where $\mathbf{c}(\mathbf{x})$ and $\tilde{\mathbf{c}}_j(\mathbf{x})$ are the color of the pixel, prior and post the removal of the $j$-th splat respectively. The cull difference describes the impact of a splat's removal but is not yet a value one can optimize for directly. The broader goal is to keep the rendered images' quality as high as possible, thus we are interested in how the rendered images compare to the ground truth images w.r.t. a loss function. This work uses the squared error as it is simple and the basis for PSNR. Additionally, instead of calculating and minimizing the squared error directly, I opt to calculate and minimize the difference in the squared error. This difference in the squared error as a result of the removal of the $j$-th splat for a single pixel is denoted as $\triangle\mathbf{SE}_j(\mathbf{x})$ and calculated using the cull difference (all operations are element-wise):

$$\begin{aligned}\triangle\mathbf{SE}_j(\mathbf{x}) &= (\tilde{\mathbf{c}}_j(\mathbf{x}) - \mathbf{c}_{\text{gt}}(\mathbf{x}))^2 - (\mathbf{c}(\mathbf{x}) - \mathbf{c}_{\text{gt}}(\mathbf{x}))^2 \\ &= (\tilde{\mathbf{c}}_j(\mathbf{x}) - \mathbf{c}(\mathbf{x}))(\tilde{\mathbf{c}}_j(\mathbf{x}) + \mathbf{c}(\mathbf{x}) - 2\mathbf{c}_{\text{gt}}(\mathbf{x})) \\ &= (\mathbf{CD}_j(\mathbf{x}))^2 + 2\,\mathbf{CD}_j(\mathbf{x})\,(\mathbf{c}(\mathbf{x}) - \mathbf{c}_{\text{gt}}(\mathbf{x}))\end{aligned} \tag{6.2}$$

We are interested in the effect a splat's removal has on the scene as a whole, not solely on a single pixel. The total impact the $j$-th splat's removal has on the squared error of the scene in accordance to $Q_c$ is denoted as $\triangle\text{SE}_j$ and defined as the sum of $\triangle\mathbf{SE}_j(\mathbf{x})$ over all pixels and all channels of all (training/testing) images:

$$\triangle\text{SE}_j = \sum_{\mathbf{x}} \triangle\mathbf{SE}_j(\mathbf{x}) \cdot \mathbf{1} \tag{6.3}$$

When all $\triangle\mathbf{SE}_j(\mathbf{x})$ values are averaged instead of summed, the mean squared error $\triangle\text{MSE}_j$ is calculated instead. Note that $\triangle\text{SE}_j$ can be negative due to the second term of $\triangle\mathbf{SE}_j(\mathbf{x})$, namely $2\,\mathbf{CD}_j(\mathbf{x})\,(\mathbf{c}(\mathbf{x}) - \mathbf{c}_{\text{gt}}(\mathbf{x}))$, being an error correcting term. If a pixel is brought closer to the ground truth, $\triangle\mathbf{SE}_j(\mathbf{x})$ will be negative and the squared error will shrink. If the squared error were applied directly to the cull difference, this term would disappear and $\triangle\text{SE}_j$ would always be positive. This phenomenon should not be neglected, as negative $\triangle\text{SE}_j$ values are more common than one might expect. For example, 27% of the splats in the scene Truck initially have a negative $\triangle\text{SE}_j$. I hypothesize this results from scenes being trained using L1 loss and SSIM, instead of the squared error this work optimizes for.

Finally, compare $\triangle\text{SE}_j$ to $I_j$. Both are splat-based metrics, but $\triangle\text{SE}_j$ is substantially richer, encompassing information about the ground truth, color, and loss function. For example, changing a splat's color will not change any $I_j$ values, but can change any number of $\triangle\text{SE}_j$ values.

### 6.2.1 Practical implementation

A naive approach to calculating all $\triangle\text{SE}_j$ values would be to remove a single splat from the model, render all training images, and calculate the squared error. This is not done as this would require millions of render passes (the number of splats times

the number of cameras). Instead, the CUDA render function of the differentiable rasterizer is altered to calculate the effect of removing each splat, all at once, during a single render pass. The result is that each camera needs to be rendered only once.

To implement this efficiently, I introduce a quantity already available during the render pass: the partial color $\mathbf{P}_j$. This is the color of a pixel after the $j$-th splat has been considered in the alpha compositing process and is defined as follows:

$$\mathbf{P}_j = \sum_{k=0}^{j} T_k \alpha_k \mathbf{c}_k \tag{6.4}$$

For notational simplicity, the dependence of $\mathbf{P}_j$ on $\mathbf{x}$ is again omitted, just as with $T_k$ and $\alpha_k$. The cull difference can be rewritten in terms of partial colors and $\alpha_j$:

$$\begin{aligned} \mathbf{CD}_j(\mathbf{x}) &= -(\mathbf{P}_j - \mathbf{P}_{j-1}) + \frac{\alpha_j}{1-\alpha_j}(\mathbf{P}_K - \mathbf{P}_j) \\ &= \mathbf{P}_{j-1} - \frac{1}{1-\alpha_j}\mathbf{P}_j + \frac{\alpha_j}{1-\alpha_j}\mathbf{P}_K \end{aligned} \tag{6.5}$$

Before the $j$-th splat is processed, the value of $\mathbf{P}_{j-1}$ is available, and after processing the $j$-th splat, $\mathbf{P}_j$ becomes known alongside $\alpha_j$. This is used to partially calculate $\mathbf{CD}_j(\mathbf{x})$ by storing the sum of the first two terms in Equation 6.5. After all splats have been processed, $\mathbf{P}_K$ also becomes known and is used with the stored value of $\alpha_j$ to finalize the $\mathbf{CD}_j(\mathbf{x})$ calculations. To store $\alpha_j$ and the intermediate result of $\mathbf{CD}_j(\mathbf{x})$ for each encountered splat during the alpha compositing process, space must be allocated beforehand. The required space is proportional to $K$, but $K$ is unknown beforehand and differs from pixel to pixel. To remedy this, a generous upper bound for $K$ is chosen at 128 splats per pixel. Only approximately 1% of splats exceed this upper bound, with the most common value of $K$ being 52 as seen in Figure 6.3. Figure 6.3 has a truncated long tail with the highest value of $K$ observed at 463. Note how simple the implementation becomes when using the partial colors, consider if this was not done, how many values need to be stored, and how many additional calculations need to be done.

Using $\mathbf{CD}_j(\mathbf{x})$ and $\mathbf{c}(\mathbf{x}) = \mathbf{P}_K$, I calculate all $\triangle\mathbf{SE}_j(\mathbf{x})$ values using Equation 6.2. All these values must be accumulated per splat to arrive at all $\triangle\text{SE}_j$ values. For the scene Truck, this comes down to reducing around 20 billion floats [2] to 2.5 million $\triangle\text{SE}_j$ values. Accumulating these values in global memory increases the time per render pass from 20 ms to 90 ms, making it by far the most expensive step in calculating $\triangle\text{SE}_j$. Therefore, the values are first aggregated at a thread block level in shared memory before being aggregated in global memory. However, the time for a single render pass only reduces from 90 ms to 60 ms as substantial complexity had to be added to achieve the thread-block level aggregation, with the limited shared memory being a major constraint. The exact implementation of the aggregation is omitted as it is not the main focus of this work. I suspect someone with more CUDA experience can still substantially speed up the aggregation process.

Finally, it should be noted that the above implementation is deliberately designed to be interoperable with the standard forward-backward render pass used during training. The forward pass used in gradient descent comes at no extra cost if the altered render pass is already executed to calculate the splats' removal effects.

[2] pixels per image $\times$ number of channels $\times$ average $K$ $\times$ number of training cameras

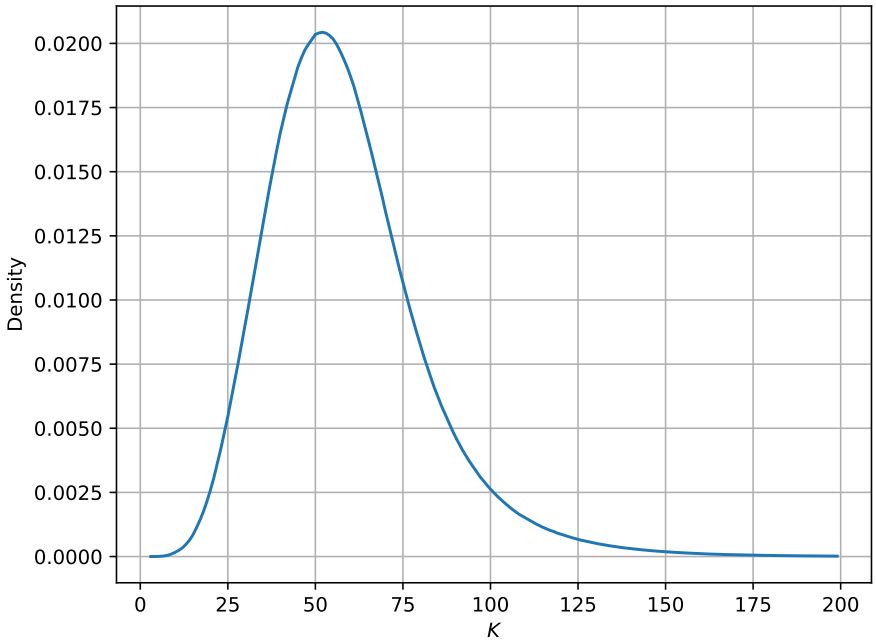



Figure 6.3: Density function of $K$, based on all training pixels of the scene Truck. Truncated at $K = 200$.

### 6.2.2 Simplifications

Due to various simplifications, $\triangle \text{SE}_j$ is calculated *almost* exactly. Empirically, the combined effect of all simplifications is minor, with the estimated $\triangle \text{SE}_j$ value falling well within a 1% error margin of the real value. An example of such a simplification is assuming $K$ remains unchanged after removing a splat. Alpha blending stops when $T_k$ would go below 0.0001, removing a splat changes subsequent values of $T_k$, affecting when this threshold is reached. Another source of simplifications is how 'out-of-bounds' pixels are handled. These are pixels whose RGB color is not between 0 and 1, which complicates the aforementioned calculations.

## 6.3 Culling controller

Knowing all $\triangle \text{SE}_j$ values does not yet provide a methodology on *how* and *which* splats to remove. This is what the culling controller is responsible for. Surprisingly, this task is more difficult than determining $\triangle \text{SE}_j$ due to $\triangle \text{SE}_j$ values changing as a result of other splats' removals. A proper greedy algorithm would remove the splat with the lowest $\triangle \text{SE}_j$ and then recalculate all $\triangle \text{SE}_j$ values before removing the next splat. Sadly, this takes too much time, thus, the culling controller needs to decide how many and which splats to remove simultaneously before recalculating all [3] $\triangle \text{SE}_j$ values. The key assumption the proposed method therefore relies on is that the effect of removing a collection of splats $S$ is approximately equal to the

[3] This work assumes all cameras are re-rendered after removing a batch of splats. Alternatives, e.g. removing a small batch of splats after each render, are not considered or studied.

sum of the individual cull effects. This approximation is called the culling approximation and is formally defined as follows:

$$\triangle \text{SE}_S \approx \sum_{j \in S} \triangle \text{SE}_j \tag{6.6}$$

The larger $S$ becomes, the more the composition of the scene changes and the less accurate we expect the culling approximation to be. However, even when $S$ consists of only 2 splats, the culling approximation can already be completely wrong. Consider the following example: suppose two opaque splats produce a bright white spot while the ground truth is slightly darker. Removing any of the two splats individually would reveal the black background slightly more, darkening the spot. The culling controller detects this by observing that $\triangle \text{SE}_j$ is negative for both splats and naively culls both splats. As a result, only the black background is visible. In this example, removing two splats with a negative $\triangle \text{SE}_j$ caused the squared error to increase significantly, highlighting that the size of $S$ can be a poor measure of the culling approximation's accuracy. A splat's removal can drastically change some splats' $\triangle \text{SE}_j$ while leaving others completely unchanged. Determining exactly how a splat's removal will change another's $\triangle \text{SE}_j$ is too computationally expensive. Luckily, crude approximations and heuristics are sufficient to adhere to the culling approximation. Some examples in increasing order of complexity:

1. **Dispersing negative $\triangle \text{SE}_j$**
   Empirically, splats whose $\triangle \text{SE}_j$ is negative often concern the same pixels. Actively dispersing splats with negative $\triangle \text{SE}_j$ values across multiple culling iterations is an inexpensive and effective way to better adhere to the culling approximation.

2. **Distance between splats: 3D**
   A splat's removal primarily affects spatially close splats' $\triangle \text{SE}_j$ as being spatially close is a sufficient condition for being close in screen space. Using the spatial closeness of splats in constructing $S$ can help to better adhere to the culling approximation as splats close in screen space will affect the same pixels and change each other's $\triangle \text{SE}_j$. Spatial closeness is not trivially defined or calculated as splats are anisotropic and differ in size. A possible metric for the spatial closeness of the $k$-th and $j$-th splat is the integral of the product of the their multivariate Gaussians:

   $$S_{kj}^{\text{3D}} = \int \mathcal{G}_k(\mathbf{x}) \mathcal{G}_j(\mathbf{x}) \, d\mathbf{x} \tag{6.7}$$

   An important result is that $S_{kj}^{\text{3D}}$ is itself a multivariate Gaussian (see Appendix B) in the position of either of the splats. $S_{kj}^{\text{3D}}$ thus decreases super-exponentially in terms of their (anisotropic) distance.

3. **Distance between splats: 2D**
   Being spatially close is a sufficient, but not a necessary condition to be close in screen space. An example that highlights this is when one looks through a translucent object. A practical measure for how much the $k$-th and $j$-th splat overlap in screen space for a camera $\mathbf{s}$ is the sum, over all pixels, of the product of two splats' contributions. This quantity is denoted as $G_{kj}(\mathbf{s})$ and will later return during spherical harmonics energy compaction:

   $$G_{kj}(\mathbf{s}) = \sum_{\mathbf{x}} T_k \alpha_k T_j \alpha_j \tag{6.8}$$

$G_{kj}(\mathbf{s})$ can be used to define its scene-wide equivalent $G_{kj}$, similarly to how $S^{\text{3D}}_{kj}$ is also a scene-wide metric:

$$G_{kj} = \sum_{\mathbf{s}} G_{kj}(\mathbf{s}) = \sum_{\mathbf{s}} \sum_{\mathbf{x}} T_k \alpha_k T_j \alpha_j \tag{6.9}$$

$G_{kj}$ and $G_{kj}(\mathbf{s})$ are intimately connected to $S_{kj}$ (as shown in Appendix B):

$$G_{kj}(\mathbf{s}) \approx S^{\text{2D}}_{kj}(\mathbf{s}) I_k(\mathbf{s}) I_j(\mathbf{s}) \tag{6.10}$$

Constructing $S$ has one additional difficulty. The culling controller should prioritize removing as many splats as quickly as possible since this speeds up all future render passes. This leads to the counter-intuitive result that a good culling controller does not start by trying to remove the splats with the lowest $\triangle\text{SE}_j$.

Removing splats also has a non-negligible upfront cost, independent of the size of $S$. Therefore, splats are first 'disabled' by setting their opacity to zero which does have a negligible cost but does not bring all the performance benefits that a splat's removal brings. The proposed method removes the disabled splats when they make up more than 5% of all splats. This threshold was empirically chosen and not analyzed in depth.

To summarize, the culling controller determines how and which splats to remove. I formulate constructing a culling controller as a multi-objective optimization problem, with the culling controller in each iteration striving to:

- Adhere to the culling approximation.
- Retain the quality.
- Make $S$ as large as possible (especially early on).

### 6.3.1 Shortcomings

Before discussing possible culling controller designs, it is important to realize that the presented framework is just one of many. This framework makes numerous assumptions to facilitate thinking about and designing a culling controller, for example, the heavy focus on adhering to the culling approximation. It is important to be mindful of the downsides of the proposed framework. Specifically, I identify two major limitations:

1. The proposed culling controller never reintroduces removed splats. As a result, it has to be very careful and pessimistic. An alternate approach could be to more optimistically disable splats, and use some heuristics to reintroduce some of the removed splats afterwards. Two examples of such heuristics: (1) look at which pixels deviated more than expected, then reintroduce the removed splats that contributed to these pixels, or (2) do not disable splats outright, instead set them to a very low opacity, if the gradient of the opacity of the pseudo-disabled splat is highly negative, then reintroduce the splat.

2. $\triangle\text{SE}_S$ is sometimes smaller than the sum of its parts. For example, consider a pixel that alpha-blends a white, black, and grey splat (in any order), where the ground truth color is grey. The white and black splats' $\triangle\text{SE}_j$ is positive, but

removing both splats together will result in the target grey again. The proposed culling controller will get stuck in a local optimum if $\triangle\text{SE}_j$ is too large for both the black and white splat. Possible solutions to such cases might be to search for overlapping splats that counteract each other, or to cull more optimistically.

While it is hard to say how prevalent such cases are in real scenes, I have found some empirical evidence for this phenomenon. In the late stages of culling, when only splats with a high $\triangle\text{SE}_j$ remain, removing around 50 to 200 splats will cause a splat with a negative $\triangle\text{SE}_j$ to appear out of nowhere. A possible explanation is the above phenomenon, but further experiments should be done to verify this.

### 6.3.2 Proposed designs

Two culling controller implementations were implemented and tested. Both implementations start from an empty $S$ and add splats one by one to $S$. To define the quality-compression trade-off, the maximal error that can be introduced by removing a splat $\triangle\text{MSE}_{\texttt{MAX}}$ is given. Both culling controllers try to end up with little to no splats where $\triangle\text{MSE}_j < \triangle\text{MSE}_{\texttt{MAX}}$.

#### Design 1

The first culling controller utilizes the contribution of a splat as a proxy for the effect of its removal. Once the 'culling budget' is reached, $S$ is finalized and all splats in $S$ are removed. The culling budget is calculated as follows:

$$\text{cull\_budget} = \frac{1}{\#\text{ iters left}} \sum_{k \in \{\, j \mid \triangle\text{MSE}_j < \triangle\text{MSE}_{\texttt{MAX}} \,\}} I_k \tag{6.11}$$

The culling budget is reached when:

$$\text{cull\_budget} \leq \sum_{k \in S} I_k \tag{6.12}$$

Splats are added to $S$ is in ascending order of their $m(\frac{\triangle\text{MSE}_j}{\triangle\text{MSE}_{\texttt{MAX}}})$ value. The mapping function $m(x)$ is chosen so splats with a small $|x|$ are removed first, with the exact sign of $x$ not being important so long that $x$ stays sufficiently small. The preference shifts towards negative $x$ values as $|x|$ grows. Practically, the following function was chosen:

$$m(x) = \begin{cases} x & x \geq 0 \\ \frac{a^2}{2(1-a)}(\sqrt{1 - \frac{4(1-a)}{a^2}x} - 1) & x < 0 \end{cases} \tag{6.13}$$

The meaning of $a$ is conveyed in Figure 6.4 which also visualizes $m(x)$ in the relevant range of values. This mapping function naturally disperses negative splats across culling iterations, helping to better adhere to the culling approximation per the earlier discussed technique.

#### Design 2

The second design does not use the splats' contributions to adhere to the culling approximation. Instead, $S$ is constructed to be as large as possible while ensuring all splats in $S$ are sufficiently spatially dispersed. Doing pair-wise comparisons to

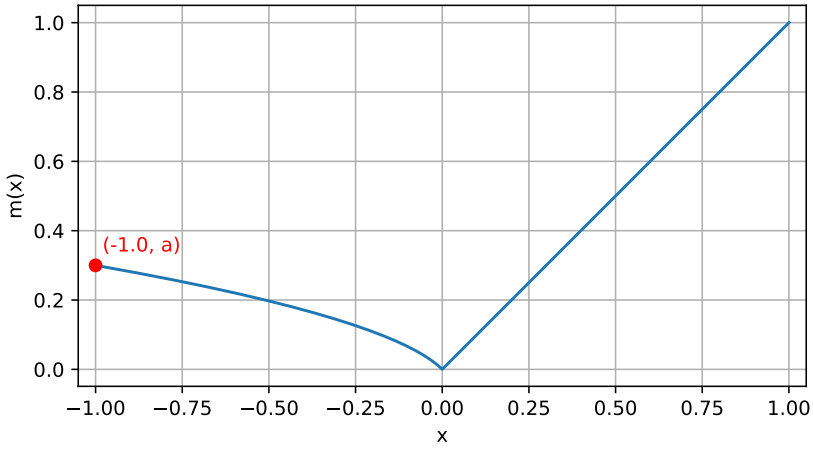



Figure 6.4: The mapping function $m(x)$ with $a = 0.3$.

ensure splats are sufficiently far from each other scales quadratically with the size of $S$ and is too expensive. To avoid this, all splats whose $\triangle\text{MSE}_j$ is less than $\triangle\text{MSE}_{\text{MAX}}$ are put into a custom k-d tree where a splat only descends if it is sufficiently far away from the splitting plane. The nodes are repeatably split until each leaf contains at most one splat. A sufficient condition for all splats in $S$ to be pair-wise sufficiently far from each other is if any path from the top node to any leaf encounters at most one splat in $S$. Given a splitting plane defined by a point $P$ and a normal parallel to the $i$-th axis, a splat with a center $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$ is sufficiently far from the splitting plane if:

$$|\boldsymbol{\mu}_i - P_i| > A \cdot \boldsymbol{\Sigma}_{ii} \tag{6.14}$$

where $A$ is some scalar that defines how strict the condition is. The splitting axis is chosen to be the axis with the highest variation, and the splitting plane's position is determined using the splats' average position along the normal of the splitting plane. The expected time complexity of constructing this tree is $\mathcal{O}(n \log n)$ with $n = |\{\, j \mid \triangle\text{MSE}_j < \triangle\text{MSE}_{\text{MAX}} \,\}|$. Using this tree, as many splats as possible are added to $S$ from the bottom up, maximizing the size of $S$.

## 6.4 Results

Empirically, the first design works better than the second design. I suspect ideas from both could be combined to create a better culling controller, but my limited initial attempts did not achieve meaningful better results. I therefore leave this to future works and from this point on, the culling controller is always assumed to be of the first design.

The first area of interest is the non-time-constrained, raw capabilities of the culling controller. Figure 6.5 shows the relation between the training PSNR and the fraction of splats removed given an excessive (512) number of culling iterations. Due to a substantial number of splats with a negative $\triangle\text{SE}_j$, the PSNR spikes after removing around 45% of splats, after this point, the curve can be seen as the Pareto frontier. Observe that approximately 65% of splats can be removed losslessly and that removing further splats has a limited effect on the PSNR. At the extreme, with 90% of the splats removed, the scene's

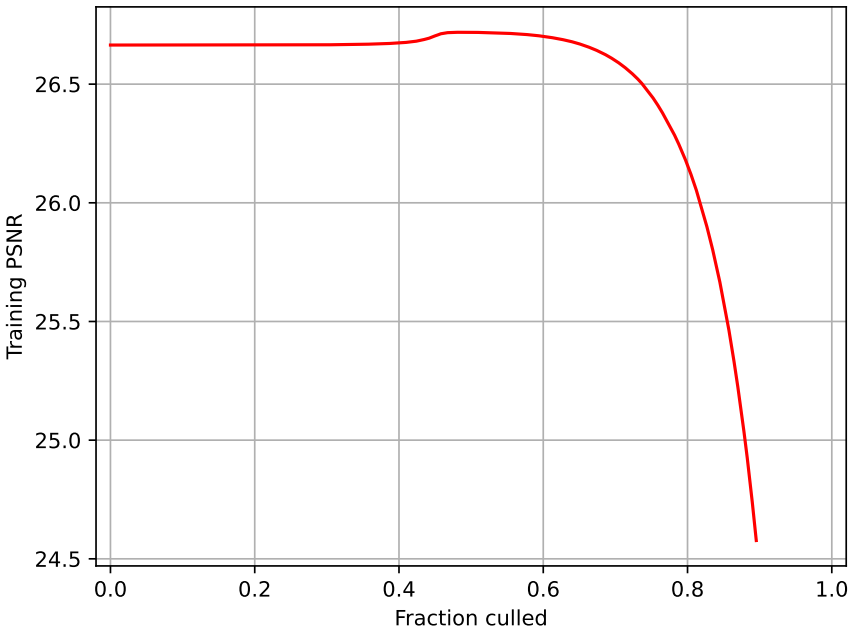



Figure 6.5: Relation between the fraction of splats removed and the training PSNR given ample (512) culling iterations for the scene Truck.

quality is still surprisingly high, as shown in Figure 6.6, considering only 1 out of 10 splats is shown. However, the degradation has become substantial enough that I do not look further beyond the 90% mark. I focus on compression with minor visual artifacts, which corresponds to removing 70% to 75% of the splats for the scene Truck.

Executing the 512 culling iterations used to construct Figure 6.5 requires over 100 minutes using a GTX 1660 Ti. Reducing the number of culling iterations greatly speeds up the culling process, but can hurt the achievability of the Pareto frontier. To study this effect, consider the scenario where $\triangle \text{MSE}_{\text{MAX}} = 5 \cdot 10^{-10}$. Figure 6.7 shows the (intermediate) results for differing numbers of culling iterations. For this configuration, using more than 32 culling iterations has negligible benefits. Also, while using only 4 culling iterations degrades the quality of the scene significantly more than the other configurations, the overall degradation is still relatively minor, going only from 26.66 to 26.16 PSNR while removing nearly 74% of splats.

(a) 0% removed (PSNR=26.66)

(b) 73% removed (PSNR=26.53)

(c) 90% removed (PSNR=24.58)

Figure 6.6: Effect of removing different fractions of splats given ample culling iterations.

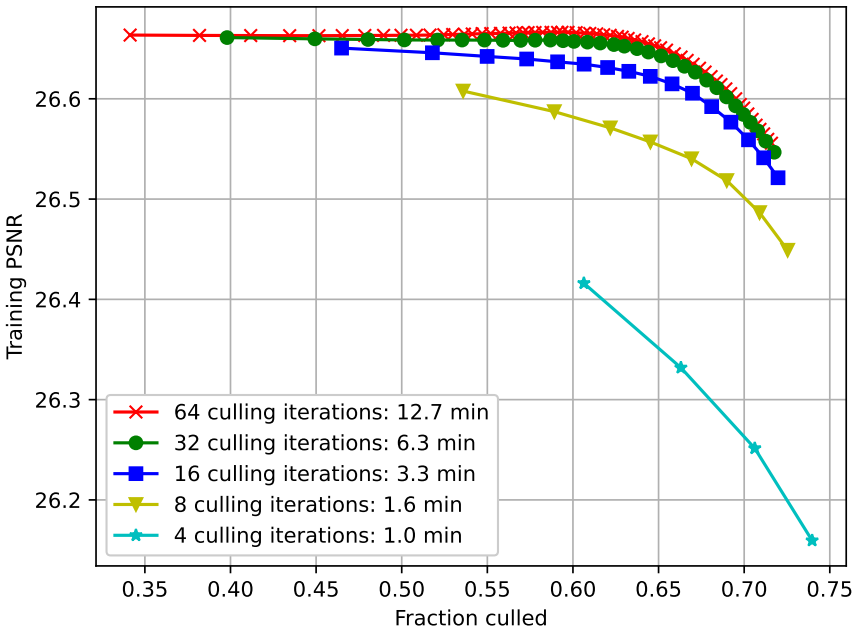



Figure 6.7: Effect of the number of culling iterations on the training PSNR of the scene Truck ($\triangle \text{MSE}_{\texttt{MAX}} = 5 \cdot 10^{-10}$).

The culling approximation was central to the design of the culling controller, we can evaluate how well it was adhered to by comparing the estimated change in error $\triangle \text{MSE}_{\text{est}}$ to the real change in error $\triangle \text{MSE}_{\text{real}}$ for each culling iteration. For each configuration presented in Figure 6.7, Figure 6.8 shows how well the culling approximation was adhered to. As expected, the culling approximation is adhered to less for configurations where $|S|$ is larger on average. The average mistake per splat is also always negative, indicating that the culling controller consistently underestimates the error it introduces.

Culling will be the first step in the proposed encoder, and thus, the starting point for further compression steps. To simplify all upcoming discussions, I choose a default starting point, namely 32 culling iterations with $\triangle \text{MSE}_{\texttt{MAX}} = 5 \cdot 10^{-10}$. The default configuration's quality resembles the one shown in Figure 6.6b. An overview of this configuration's results on test images using three quality metrics (PSNR, SSIM, and LPIPS [31]) is presented in Table 6.1. Additionally, the change in the number of splats and frames per second (FPS) is given, highlighting the effectiveness of the proposed method at both compression and faster rendering, with some scenes even more than doubling their frames per second.

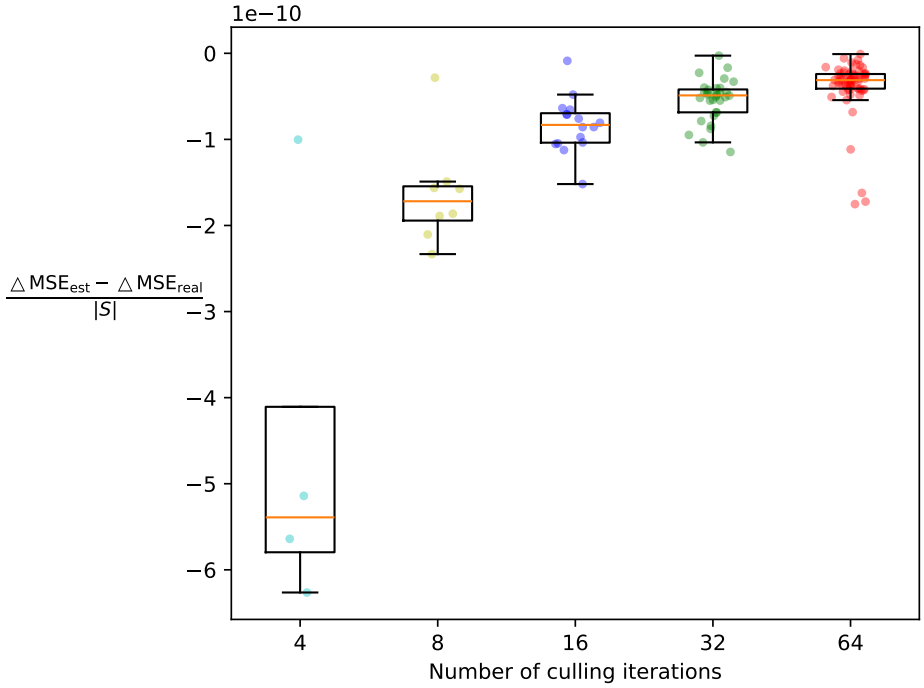



Figure 6.8: Effect of culling speed on the accuracy of the predicted introduced error.

| Scene | PSNR$^{\uparrow}$ (+/-$^{\uparrow}$) | SSIM $^{\uparrow}$ (+/- $^{\uparrow}$) | LPIPS $^{\downarrow}$ (+/- $^{\downarrow}$) | # Splats $^{\downarrow}$ (% removed $^{\uparrow}$) | FPS $^{\uparrow}$ (+/- $^{\uparrow}$) | Runtime $^{\downarrow}$ |
|---|---|---|---|---|---|---|
| Drjohnson [32] | 29.53 (-0.59) | 0.889 (-0.007) | 0.271 (+0.023) | 600,180 (82%) | 220 (+117) | 10.5 min |
| Playroom [32] | 29.64 (-0.29) | 0.888 (-0.013) | 0.271 (+0.026) | 419,181 (84%) | 220 (+125) | 7.2 min |
| Train [30] | 22.12 (+0.35) | 0.793 (-0.011) | 0.238 (+0.020) | 467,167 (54%) | 125 (+35) | 6.7 min |
| Truck [30] | 24.87 (-0.08) | 0.858 (-0.013) | 0.171 (+0.017) | 717,686 (72%) | 121 (+48) | 6.3 min |

Table 6.1: The proposed method's post-culling results for various scenes. The FPS and runtime were measured using a GTX 1660 Ti.

# 7

# Spherical Harmonics Energy Compaction

## 7.1 Unequal sampling

Spherical harmonics facilitate the splats' colors being view-dependent. The self-evident implication/assumption is that splats are looked at from different angles, but as it turns out, this is not self-evident at all, with most splats only ever being looked at from a narrow range of directions. This unequal sampling is the result of one or more of the following factors:

1. **Dispersed splats and concentrated $Q_c$**
   When control of the camera is given to a user, the design of a scene needs to accommodate this by extending the scene beyond the immediate boundaries of the camera's allowable freedom. Consequently, within 3DGS, the splats are spatially more dispersed than $Q_c$ is, leading to numerous splats that the user can solely look at from afar. The more distant such a splat is, the narrower the range of directions a splat can be queried from.

2. **Unevenly distributed $Q_c$**
   Even when a splat is not such an 'out-of-reach' splat, there is still no guarantee that $Q_c$'s mass evenly/fully surrounds the splat. For example, the truck in the scene Truck is clearly at the center of $Q_c$'s mass but no top-down or bottom-up images are taken of the truck. This turns out to be more of a scene than a splat property. To visualize this, Figure 7.1 shows a heatmap of all splats' observation directions, for all training cameras, for the scene Truck. A splat's observation direction for a given camera corresponds to the vector going from the splat center to the camera. This scene has an equator-like sampling characteristic, due to images being taken by circling at eye level. No samples exist around the sphere's poles since the camera never looks directly down or up. While not all scenes will have an equator-like sampling characteristic, e.g. aerial footage, some other characteristic will take its place.

3. **Lack of visibility**
   Even when a splat is not an 'out-of-reach' splat and $Q_c$ is evenly distributed, chances are still meager to have anything resembling uniform sampling due to splats occluding one another, making the sampling biased.

To help quantify the severity of this phenomenon, the average angular deviation $\overline{\Delta\theta}$ is defined as the average angle between a splat's observation direction for a given camera and the splat's mean observation direction. We talk about the weighted average angular deviation if the mean observation direction and average deviation are weighed by some vector $\mathbf{w}$. The average angular deviation reduces a set of directions to a single scalar, making it inherently less dimensional. As a result, it

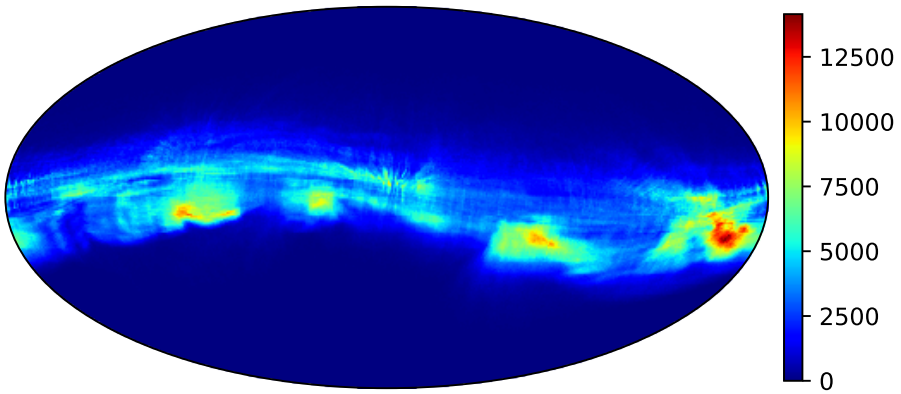



Figure 7.1: Heatmap of all splats' observation directions, for all training cameras, for the scene Truck.

is not a perfect measure, with numerous alternatives existing, e.g. the norm of the mean observation direction, the angular variance, and PCA-based metrics. Like all its alternatives, $\overline{\Delta\theta}$ has downsides, the primary one is its rotational invariance w.r.t. the mean observation direction. Samples on a line are the least random configuration of all rotational variants, making $\overline{\Delta\theta}$ a pessimistic measure for scenes with an equator-like sampling characteristic. Compare this with the most random variant where the rotation around the mean observation direction is uniformly distributed. For example, consider a uniformly sampled spherical cap $S_\alpha$ with a cone angle $\alpha$. The spherical cap gives a conceptually easy reference for a worst-case equivalent of a splat's sampling properties. $S_{180°}$ corresponds with a uniformly sampled sphere, $S_{90°}$ corresponds with a uniformly sampled hemisphere, and so forth. For clarity, $\alpha$ is always denoted in degrees, while $\overline{\Delta\theta}$ is denoted in radians. Values for $\alpha$ and $\overline{\Delta\theta}$ should never be compared directly. With this background, Figure 7.2 shows a histogram of $\overline{\Delta\theta}$ for all splats. For the weighted case, the weighting vector is chosen as the splat's contribution for every training camera, i.e. $\mathbf{w}_k = (I_k(\mathbf{s}_1), ..., I_k(\mathbf{s}_N))^T$. Due to the NeRF-like capture where a camera circles a central object, the splats are naturally partitioned into two groups: splats inside the capturing circle (high unweighted $\overline{\Delta\theta}$), and splats outside the capturing circle (low unweighted $\overline{\Delta\theta}$). After bringing the camera contributions into account, the histogram shifts dramatically towards lower $\overline{\Delta\theta}$, with a splat's weighted and unweighted $\overline{\Delta\theta}$ being highly correlated (pearson=0.87, spearman=0.85).

## 7.2 Energy compaction

Coupling back to spherical harmonics and compression, a small $\overline{\Delta\theta}$ indicates that a splat is only looked at from a narrow range of directions. Changing a splat's color for observation directions outside this meaningful region will not have any visual impact as the splat is never looked at from said region. Therefore, the smaller $\overline{\Delta\theta}$ is, the more combinations of lighting coefficients will produce near identical results. The key idea is to find an alternate set of lighting coefficients that is easier to compress. Such a set typically has low energy and high sparsity. This alternate representation does not have to exactly produce the same colors for all the training cameras, a small error, inversely proportional to the splat's camera

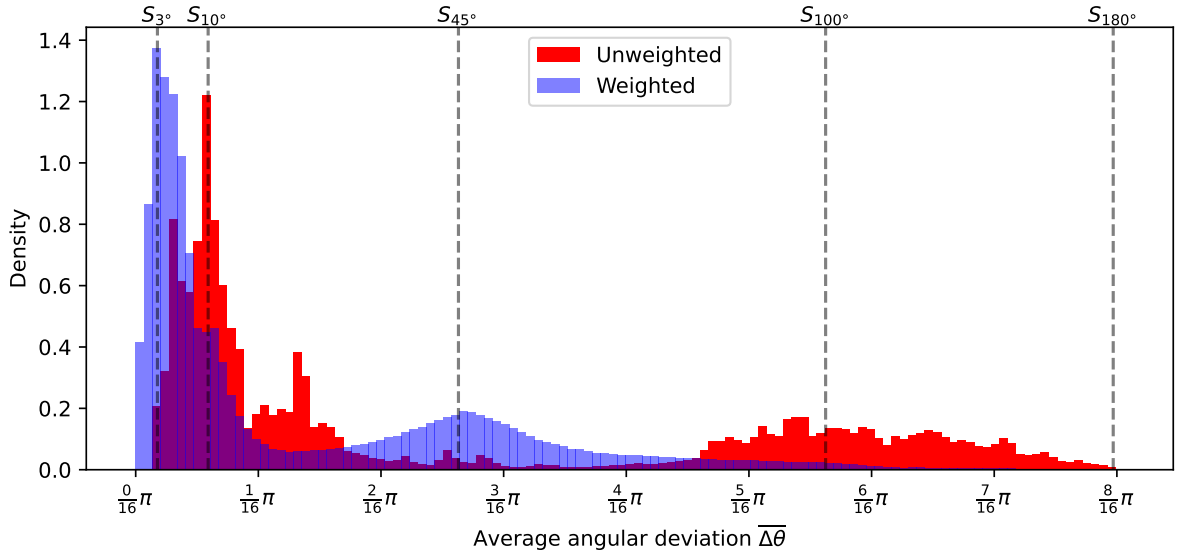



Figure 7.2: Histogram of the average angular deviation $\overline{\Delta\theta}$ for all splats in the scene Truck. The unweighted and weighted versions ($\mathbf{w}_k = (I_k(\mathbf{s}_1), ..., I_k(\mathbf{s}_N))^T$) are shown. Some reference uniformly sampled spherical caps $S_\alpha$, with cone angle $\alpha$, are marked.

contribution, can be tolerated. Since $\overline{\Delta\theta}$ correlates with a splat's contribution (spearman=0.56), lighting coefficients should be even more compressible the lower $\overline{\Delta\theta}$ is.

### 7.2.1 Comparison with culling

The energy compaction problem differs from the splat removal problem in three fundamental ways:

1. **Binary versus continuous optimization**
   Removing a splat is a binary decision, a splat is either kept or removed. Energy compaction is a continuous problem in which each lighting coefficient can take on any real value.

2. **Effect on geometry**
   Removing a splat changes the scene's geometry in a complex, non-linear fashion. Altering lighting coefficients leaves the geometry of a scene untouched as the geometry of a scene depends only on the non-color attributes.

3. **Complex Pareto frontier**
   The number of splats is an excellent proxy for the final file size. Compare this with the more hazy connection a low-energy set of lighting coefficients has to the final file size. For example, suppose vector quantization is used to compress the lighting coefficients (as is common in literature). Energy compaction will reduce the entropy of each element in the code book but might disperse clusters, negatively impacting the final size. There are simply

more factors at play during energy compaction than culling, making the 'goodness' of a set of lighting coefficients comparatively more ill-defined and harder to calculate.

## 7.3 Direct approach: Using a pixel-based loss metric

A first approach directly relates a splat's color to a pixel-based loss metric. This continues the train of thought outlined by the culling approach, however, as we now enter the continuous domain, this boils down to computing the gradient w.r.t. to the lighting coefficients. Therefore, an initial energy compaction method is to train a scene after adding a regularization term in the loss function. Since this work focuses on out-of-loop compression techniques, this method, and other gradient descent-based methods, are not discussed further. Instead, I try to solve for the globally optimal splat colors, which will lead to theoretical insights but is of limited practical use.

First, assume a splat's color is unconstrained per camera, i.e. ignore lighting coefficients exist. The lowest squared error solution is found when the derivate of the squared error is zero, for every splat's color, for every camera. This results in (a full derivation is provided in Appendix A):

$$\frac{\partial \sum_{\mathbf{s}} \sum_{\mathbf{x}} \mathsf{SE}(\mathbf{x}, \mathbf{s})}{\partial \mathbf{c}_k(\mathbf{s}_l)} = 0 \Longrightarrow \sum_{j=0}^{P-1} \mathbf{c}_j(\mathbf{s}_l) G_{kj}(\mathbf{s}_l) = \mathbf{H}_k(\mathbf{s}_l) \tag{7.1}$$

$$G_{kj}(\mathbf{s}_l) = \sum_{\mathbf{x}} T_k \alpha_k T_j \alpha_j \tag{7.2}$$

$$\mathbf{H}_k(\mathbf{s}_l) = \sum_{\mathbf{x}} \mathbf{c}_{\mathsf{gt}}(\mathbf{x}, \mathbf{s}_l) T_k \alpha_k \tag{7.3}$$

where $G_{kj}(\mathbf{s}_l)$ and $\mathbf{H}_k(\mathbf{s}_l)$ are both geometry-based factors that are not affected by changes to color attributes. Therefore, taking the derivate for each $(\mathbf{c}_k, \mathbf{s}_l)$ pair results in a system of equations that is linear in the splat colors. We can introduce lighting coefficients by taking the derivate with respect to the $l$-th lighting coefficient of the $k$-th splat (for all color channels), denoted as $\mathbf{L}_l(k)$. The system of equations changes slightly as a result (a full derivation is provided in Appendix A):

$$\frac{\partial \sum_{\mathbf{s}} \sum_{\mathbf{x}} \mathsf{SE}(\mathbf{x}, \mathbf{s})}{\partial \mathbf{L}_l(k)} = 0 \Longrightarrow \sum_{j=0}^{P-1} \sum_{m=1}^{M} \mathbf{L}_m(j) \tilde{G}_{kj,lm} = \tilde{\mathbf{H}}_{k,l} \tag{7.4}$$

$$\tilde{G}_{kj,lm} = \sum_{\mathbf{s}} Y_l(\mathbf{s}(k)) Y_m(\mathbf{s}(j)) G_{kj}(\mathbf{s}) \tag{7.5}$$

$$\tilde{\mathbf{H}}_{k,l} = \sum_{\mathbf{s}} Y_l(\mathbf{s}(k)) \mathbf{H}_k(\mathbf{s}) \tag{7.6}$$

This again results in a, albeit smaller, linear system as $\tilde{G}_{kj,lm}$ and $\tilde{\mathbf{H}}_{k,l}$ are both independent of the lighting coefficients.

Both systems of equations are linear only because the squared error was used as a loss function. 3DGS uses a combination of L1 loss and SSIM during training, which do not have corresponding linear systems of equations. Here, the linearity provides hope that the systems of equations are easily solvable, but, sadly, both linear systems are prohibitively large to solve.

For example, the scene Truck after culling can still easily have over 625000 splats (corresponding to 30 million lighting coefficients, and by extension, equations). Solving a dense linear system of this size is practically infeasible. Therefore, this approach is abandoned initially.

## 7.4 Indirect approach: Using a parameter-based loss metric

An alternative to a pixel-based loss metric is one that instead operates on the perceived splat colors for each camera. This greatly simplifies the calculations, but was initially avoided as it limits/complicates fully exploiting knowledge about $Q_c$ and $\mathbf{c}_{\text{gt}}$.

First, consider a single channel of a splat's color for the $i$-th camera. A channel's value is defined using its $M$ lighting coefficients:

$$C(\mathbf{s}_i) = \sum_{l=1}^{M} L_l Y_l(\mathbf{s}_i) \tag{7.7}$$

Equation 7.7 holds for all $N$ cameras. All the corresponding equations for a single splat can be combined using a matrix multiplication as follows:

$$\underbrace{\begin{pmatrix} C(\mathbf{s}_1) \\ C(\mathbf{s}_2) \\ C(\mathbf{s}_3) \\ \vdots \\ C(\mathbf{s}_N) \end{pmatrix}}_{\mathbf{C}} = \underbrace{\begin{pmatrix} Y_1(\mathbf{s}_1) & Y_2(\mathbf{s}_1) & Y_3(\mathbf{s}_1) & \cdots & Y_M(\mathbf{s}_1) \\ Y_1(\mathbf{s}_2) & Y_2(\mathbf{s}_2) & Y_3(\mathbf{s}_2) & \cdots & Y_M(\mathbf{s}_2) \\ Y_1(\mathbf{s}_3) & Y_2(\mathbf{s}_3) & Y_3(\mathbf{s}_3) & \cdots & Y_M(\mathbf{s}_3) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ Y_1(\mathbf{s}_N) & Y_2(\mathbf{s}_N) & Y_3(\mathbf{s}_N) & \cdots & Y_M(\mathbf{s}_N) \end{pmatrix}}_{\mathbf{Y}} \times \underbrace{\begin{pmatrix} L_1 \\ L_2 \\ L_3 \\ \vdots \\ L_M \end{pmatrix}}_{\mathbf{L}} \tag{7.8}$$

The matrix $\mathbf{Y}$ is a known constant matrix, independent of any lighting coefficients. It need not be saved as $\mathbf{s}_i$ depends only on the splat's and camera's positions. In contrast, the vector $\mathbf{L}$ is saved and its properties are important for compression. Finally, the vector $\mathbf{C}$ is calculated from $\mathbf{Y}$ and $\mathbf{L}$ and defines the channel's value for each of the $N$ cameras, which in turn affects the scene's quality.

Energy compaction aims to find a new $\mathbf{L}$, denoted as $\mathbf{L}'$, located on the Pareto front of 'quality' and 'compressibility'. Changing $\mathbf{L}$ to $\mathbf{L}'$ causes the corresponding vector $\mathbf{C}$ to change to $\mathbf{C}'$ with the change in a splat's channel value for the $i$-th camera being $\mathbf{C'}_i - \mathbf{C}_i$. Since the number of cameras is normally much larger than the number of lighting coefficients, i.e. $N \gg M$, the linear system presented in Equation 7.8 is *overdetermined*, with no alternate solutions expected/guaranteed to exist. The general assumption is therefore that $\mathbf{L}' \neq \mathbf{L} \implies \mathbf{C}' \neq \mathbf{C}$, meaning that energy compaction becomes a trade-off between the introduced error $\mathbf{C}' - \mathbf{C}$, and the change in compressibility between $\mathbf{L}$ and $\mathbf{L}'$. I use ridge regression (also known as Tikhonov regularization, or L2 regularization) to define the optimal trade-off:

$$\mathbf{L}' = \underset{\mathbf{x} \in \mathbb{R}^{M \times 1}}{\arg\min} \left( \|\mathbf{Y}\mathbf{x} - \mathbf{C}\|_2^2 + \|\mathbf{\Gamma}\mathbf{x}\|_2^2 \right) \tag{7.9}$$

where $\|\cdot\|_2$ is the Euclidian norm of a vector, and $\mathbf{\Gamma}$ denotes the Tikhonov matrix which defines how $\mathbf{x}$ is regularized. The Tikhonov matrix is chosen as follows:

$$\mathbf{\Gamma} = \sqrt{\lambda}\left(\begin{array}{c|c} 0 & \mathbf{0}_{1\times M-1} \\ \hline \mathbf{0}_{M-1\times 1} & \mathbf{I}_{M-1\times M-1} \end{array}\right) \tag{7.10}$$

where $\mathbf{I}$ denotes the identity matrix. The top-left element is zero as we do not wish to regularize the DC lighting coefficient.

This choice of formulation carries with it two implications. First, the deviation of $\mathbf{C}'$ from $\mathbf{C}$ is punished using the squared difference. This falls within the train of thought of PSNR, which uses the squared error on a per-pixel basis. Second, the 'compressibility' of the AC lighting coefficients is defined using their squared magnitude. While this choice has merit, many other regularization techniques exist, e.g. L1 regularization. The benefit of ridge regression, and the reason it was chosen for this work as opposed to other regularization techniques, is its closed-form solution:

$$\mathbf{L}' = (\mathbf{Y}^T\mathbf{Y} + \mathbf{\Gamma}^T\mathbf{\Gamma})^{-1}\mathbf{Y}^T\mathbf{C} \tag{7.11}$$

A closed-form solution is essential as $\mathbf{L}'$ has to be calculated for each splat, with scenes having hundreds of thousands to millions of splats.

### 7.4.1 Full exploitation

The central idea and method behind energy compaction is fairly simple, but fully exploiting the technique's strengths requires careful adaptations of the above model:

#### Contribution

A splat's contribution to a camera differs greatly from camera to camera. Additionally, some splats are more important than others as their contribution to the scene as a whole is larger. Equation 7.9 punishes each deviation, $\mathbf{C}'_i - \mathbf{C}_i$, equally for all cameras. A more fair loss function would integrate $I_k(\mathbf{s})$ into the loss function, e.g. if $I_k(\mathbf{s}_i) = 10 \cdot I_k(\mathbf{s}_j)$ then a deviation from $\mathbf{C}_i$ should be punished 10 times as harshly as a deviation from $\mathbf{C}_j$. This can be achieved by doing the following substitutions in Equation 7.9 and 7.11:

$$\mathbf{C} \leftarrow \mathbf{C} \odot_{\text{col}} (I_k(\mathbf{s}_1), ..., I_k(\mathbf{s}_N))^T \tag{7.12}$$

$$\mathbf{Y} \leftarrow \mathbf{Y} \odot_{\text{col}} (I_k(\mathbf{s}_1), ..., I_k(\mathbf{s}_N))^T \tag{7.13}$$

where $\odot_{\text{col}}$ represents a column element-wise multiplication. This adaptation considers both the contribution differences between cameras and splats.

#### Color model

3DGS uses RGB as its color model, unsurprisingly, lighting coefficients corresponding to the same basis function, but a different color channel, correlate highly. Moving to a luminance/chrominance-based color model, e.g. YUV, has two benefits: (1) the collective energy is reduced, and (2) the chrominance channels can be more harshly regularized as humans are comparatively less perceptive to changes in chrominance. I did not implement the latter as doing so without subjective quality assessments is not straightforward. Empirically, the difference between different color spaces with a single luminance channel and two chrominance channels, e.g. YCbCr and YCoCg, is negligible in terms of compression.

Surprisingly, to the best of my knowledge, this is the first compression work to use a non-RGB color model.

#### Sparsity

Ridge regression comes with the notable downside that the magnitude of lighting coefficients is punished super-linearly. As a result, energy is dispersed more than desirable across multiple lighting coefficients. Consider the following toy example [1]:

$$\mathbf{C} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} \qquad \mathbf{Y} = \begin{pmatrix} 2.01 & 1.01 & 0 \\ 2 & 1 & 0 \\ 2 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \qquad \mathbf{\Gamma} = \begin{pmatrix} 0.1 & 0 & 0 \\ 0 & 0.1 & 0 \\ 0 & 0 & 0.1 \end{pmatrix}$$

The first and second columns of $\mathbf{Y}$ are two almost parallel vectors, meaning that so long that $2L'_1 + L'_2 = 1$ remains constant, the exact values of $L'_1$ and $L'_2$ will negligibly impact $C'$. Using ridge regression will thus lead to the lowest energy solution $\mathbf{L}' \approx \begin{pmatrix} 0.4 & 0.2 & 1 \end{pmatrix}^T$ instead of the sparsest solution $\mathbf{L}' \approx \begin{pmatrix} 0.5 & 0 & 1 \end{pmatrix}^T$. Compare this with sub-linear regularization methods, which would give the sparsest solution.

Besides sparse solutions being more desirable due to their typically lower entropy, a second problem emerges from the above example. When vectors are so parallel that they are practically the same vector, they should be treated as such. However, ridge regression will disperse the energy across multiple vectors, leading to under-regularization. In the toy example, this is observed as $0.4^2 + 0.2^2 < 0.5^2$. This would not be a problem if the vectors were randomly distributed, as high dimensional vectors tend not to be parallel [2]. However, due to the unequal sampling, nearly parallel column vectors are the rule rather than the exception. For example, if all samples lay perfectly on the equator, the vectors corresponding to $Y_{1,-1}$ and $Y_{0,0}$ are the same.

To promote sparsity and prevent under-regularization, the $i$-th column of $\mathbf{Y}$ is set to zero (which leads to $L'_i = 0$) if a $j$-th column vector exists which is 'sufficient parallel' and where $j < i$. Sufficiently parallel is defined as having a cosine similarity whose absolute value is larger than some constant $\alpha$. The cosine similarity of the $i$-th and $j$-th column is

[1] The example is purely for explanatory reasons, it can not occur in practice.

[2] https://math.stackexchange.com/a/995678

defined as the dot product between their normalized forms, and is easily found for all pairs at once using a single matrix multiplication:

$$\text{cosine\_similarity}(i, j) = (\mathbf{Y}_{\text{norm}}^T \mathbf{Y}_{\text{norm}})_{ij} \tag{7.14}$$

where each column of $\mathbf{Y}$ has been normalized to form $\mathbf{Y}_{\text{norm}}$. A cosine similarity near 1 or -1 indicates near parallel vectors, while a cosine similarity near 0 indicates near perpendicular vectors.

This method is simple, easily implementable, and has the notable benefit of favoring lower-degree spherical harmonics as if two column vectors are parallel, only the lowest degree one is kept. A major downside is that it does not consider that vectors being 'sufficiently parallel' is not transitive. Suppose $i < k < j$ and that the $k$-th column is sufficiently parallel with both the $i$-th and $j$-th column, then only the $i$-th column will be kept, while there is no guarantee that the $i$-th column is sufficiently parallel with the $j$-th column. Future works could look at more advanced algorithms that, for example, rely on a clique-based graph argument.

#### Two-pass regularization

Removing parallel columns using $\alpha = 0.9$ for the scene Truck sets 60% of AC coefficients to zero. Ridge regression forces an additional 34% of AC coefficients to be *almost* zero such that these lighting coefficients will be quantized to zero in Chapter 8. To reduce the quantization loss and to present ridge regression with a more true representation of its flexibility, a two-pass regularization system is used. The first pass identifies all almost zero AC coefficients and forces them to zero using the same technique presented in the sparsity section. The second pass finds the optimal values for the remaining non-zero lighting coefficients.

#### Numerical instabilities

Single-precision floats cause numerical instabilities when the model is barely regularized. If double-precision floats are used instead, this problem does not occur. The same parallel vectors mentioned in the sparsity section are the main culprit for the numerical instabilities. Applying the proposed sparsity modification also resolves most numerical instabilities. Single precision floats are preferable as they are significantly, empirically around a third, faster to work with.

### 7.4.2 Performance

After calculating all $I_k(\mathbf{s}_i)$ values, a single core on my Intel® Core™ i7-9750H can process approximately 6000 splats per second. As a result, it takes roughly two minutes to compute all new lighting coefficients for the scene Truck. However, there is still immense potential to speed up this process as the proposed energy compaction algorithm can be performed on all splats simultaneously. This is left for future works as two minutes is sufficiently short for this initial work.

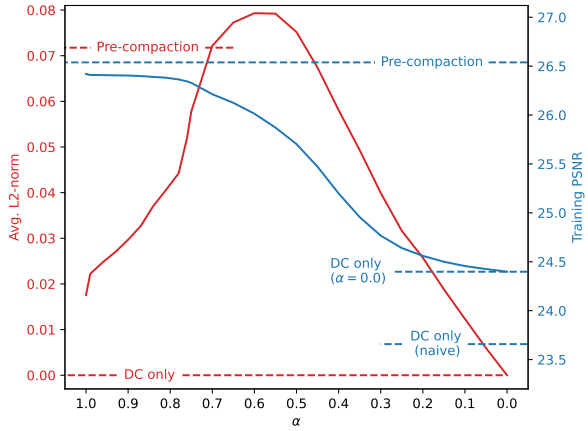
## 7.5 Results

Only the indirect approach will be discussed as the direct approach was abandoned. The proposed energy compaction technique has two hyperparameters: the regularization strength $\lambda$ and the threshold for vectors to be 'sufficiently parallel' $\alpha$. The main area of interest is the relationship between $\lambda$ or $\alpha$, and, the L2-norm or sparsity of the AC lighting coefficients. I will not consider every combination of $\lambda$ and $\alpha$, instead, I alter each hyperparameter while keeping the other at its default value. The default values, $\lambda = 1.0$ and $\alpha = 0.9$, were obtained by hand-optimizing the encoder such that the quality-compression trade-off roughly equals the default configuration's culling quality-compression trade-off.

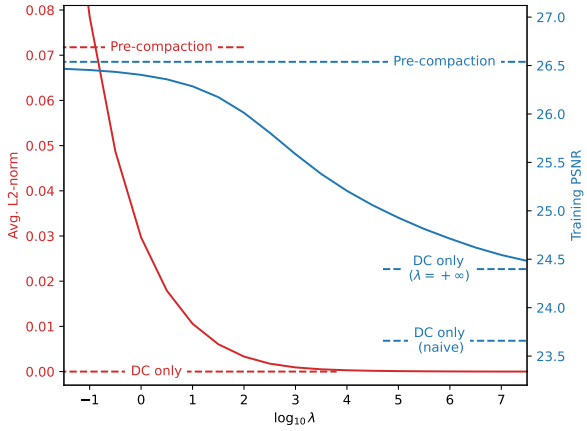
Figure 7.3 shows how $\lambda$ and $\alpha$ affect the L2-norm, sparsity, and quality of a scene. Except for Figure 7.3a, all the subfigures' shapes are fairly predictable/uneventful. Therefore first consider $\alpha$'s relationship with the L2-norm. When $\alpha$ is $1$, no column vectors are removed and the AC lighting coefficients energy can be spread across parallel vectors leading to a small L2-norm. This circumvents the desired regularization and leads to under-regularized splats. As $\alpha$ decreases, parallel vectors are removed and the energy becomes more concentrated, leading to a higher L2-norm and effective regularization. When $\alpha$ becomes too low, the removed vectors are not all quasi redundant anymore and their removal limits the model's freedom resulting in degraded quality. In comparison, $\lambda$ has a simple relation with the L2-norm. Figure 7.3b shows the effect of $\lambda$ on both the average L2-norm and the training PSNR. The L2-norm can be significantly lowered while keeping the loss in quality to a minimum. Note that the L2-norm can exceed the original L2-norm due to $\alpha = 0.9$ raising the baseline L2-norm.

Second, consider how sparsity is affected by $\lambda$ and $\alpha$. To form a fair comparison between pre- and post-compaction sparsity, the criterion for an AC lighting coefficient to be regarded as zero is broadened to better reflect post-quantization sparsity (see Chapter 8). Still, even with this more generous criterion, Figures 7.3c and 7.3d show that the sparsity can be hugely increased at a minimal cost in quality. This is despite ridge regression not naturally leading to sparsity. However, due to the proposed modifications, lighting coefficients susceptible to sparsity are identified and exploited. For example, $Y_{1,1}$ and $Y_{1,-1}$ are respectively the AC bases with the most and least non-zero coefficients with less than one non-zero $L_{1,-1}$ existing for every 6 non-zero $L_{1,1}$ coefficients. This phenomenon directly results from the Truck scene's equator sampling characteristic as $Y_{1,1}$ changes substantially along the equator, while $Y_{1,-1}$ doesn't change at all along the equator. Similarly, different color channels have varying levels of sparsity. The default configuration for the scene Truck reaches an overall sparsity of 94.8%, but the Y, Co, and Cg color channels have a respective sparsity of 87.7%, 97.5%, and 99.3%. This adaptability of the model also extends on a per-splat basis with Figure 7.4 showing that while every type of splat gets more sparse after energy compaction, splats outside the camera circle get markedly more sparse. All these factors compound, for example, the Cg channel's $L_{1,-1}$ coefficients for splats outside the camera circle are only non-zero for 0.32% of splats (or about 2300 splats), while the Y channel's $L_{1,1}$ coefficients for splats inside the camera circle are non-zero for 58% of splats (or about 100,000 splats), with other even more extreme combinations existing on either end.

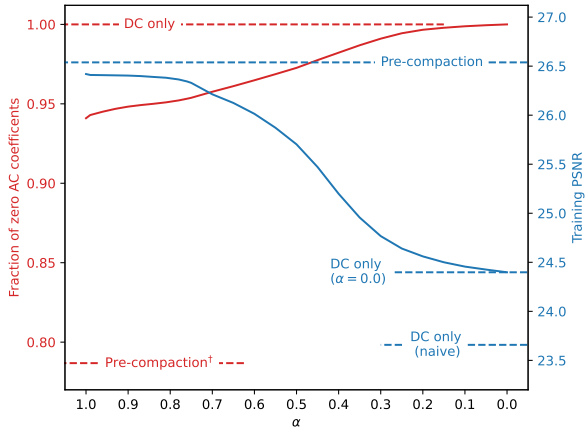
To provide some practical examples of how energy compaction affects the splats of the scene, three splats are studied more closely in Figures 7.5, 7.6, and 7.7. Here, a blurred, top-down view, of the scene Truck is shown with a red cross marking the splat in question. Small black arrows represent the position and direction of the training cameras with the size of the green dot they are placed upon indicating the cameras' relative contribution. Below this top-down view, Mollweide projections visualize the colors the marked splat can take on, before and after compaction, with each green dot representing a camera



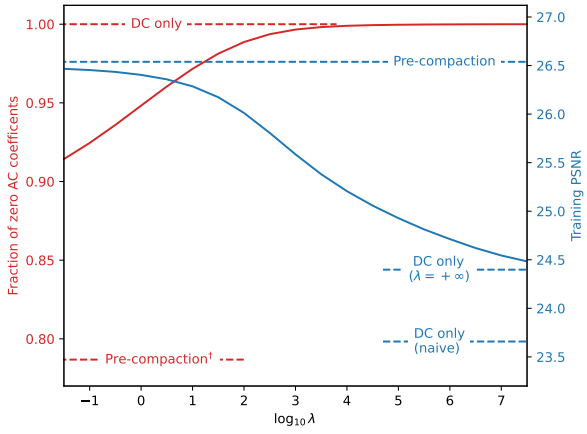

(a) Effect of $\alpha$ on the L2-norm and quality ($\lambda = 1.0$).




(b) Effect of $\lambda$ on the L2-norm and quality ($\alpha = 0.9$).

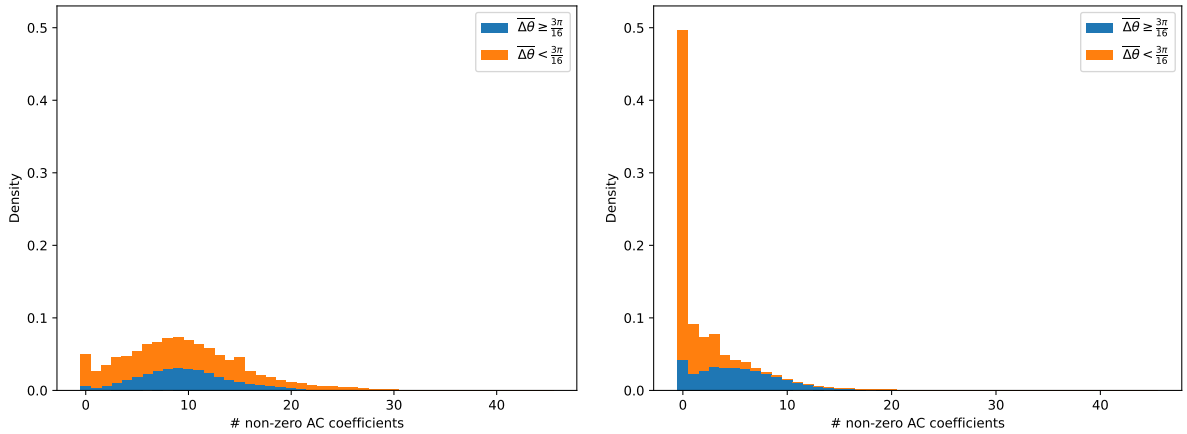



(c) Effect of $\alpha$ on the sparsity and quality ($\lambda = 1.0$).




(d) Effect of $\lambda$ on the sparsity and quality ($\alpha = 0.9$).

Figure 7.3: Effect of $\alpha$ and $\lambda$ on the L2-norm, sparsity, and quality.
† denotes that an AC coefficient is regarded as zero if $|L_i| < \frac{1}{32}$.




(a) Before energy compaction ($L_i$ is non-zero if $|L_i| < \frac{1}{32}$)

(b) After energy compaction ($L_i$ is non-zero if $L_i \neq 0$)

Figure 7.4: Stacked density histogram of the number of non-zero AC lighting coefficients per splat. Splats outside the camera ring ($\overline{\Delta\theta} < \frac{3\pi}{16}$), which are more unequally sampled, have a significantly higher sparsity after energy compaction.

and their relative contribution.

In conjunction with subsequent steps presented in Chapter 8, energy compaction leads to smaller file sizes. Were energy compaction not used, the final file size of the scene Truck would be approximately twice as large due to the cost of encoding the AC lighting coefficients more than doubling.

## 7.6 Direct approach: revisited

The direct approach was deemed infeasible because the underlying linear system was too unwieldy. But now, the indirect approach reveals that a great deal of sparsity can be introduced, essentially already solving a large part of the linear system. For the scene Truck, 95% of AC lighting coefficients can be set to zero using the indirect approach, resulting in a linear system that is approximately 750 times faster to solve. The calculations can be further sped up 9x by considering each color channel independently. These two modifications hugely reduce the linear system's size but do not make it practically solvable. Another technique to speed up solving the linear system is identifying and exploiting the sparsity of the coefficients of the linear system. Here, the linear system's coefficients are primarily determined by $G_{kj}(\mathbf{s})$ which can be approximated as follows (see Appendix B):

$$G_{kj}(\mathbf{s}) \approx S^{\text{2D}}_{kj} I_k(\mathbf{s}) I_j(\mathbf{s}) \tag{7.15}$$

where $S^{\text{2D}}_{kj}$ is a multivariate Gaussian in the screen-space position of either of the splats and thus decreases super-exponentially in terms of their (anisotropic) screen distance. As a result, $G_{kj}(\mathbf{s})$ will be (approximately) zero for most splat pairs, as splats typically only cover a small part of the screen. This sparsity brings this linear system within the realm of plausibly solvable problems.

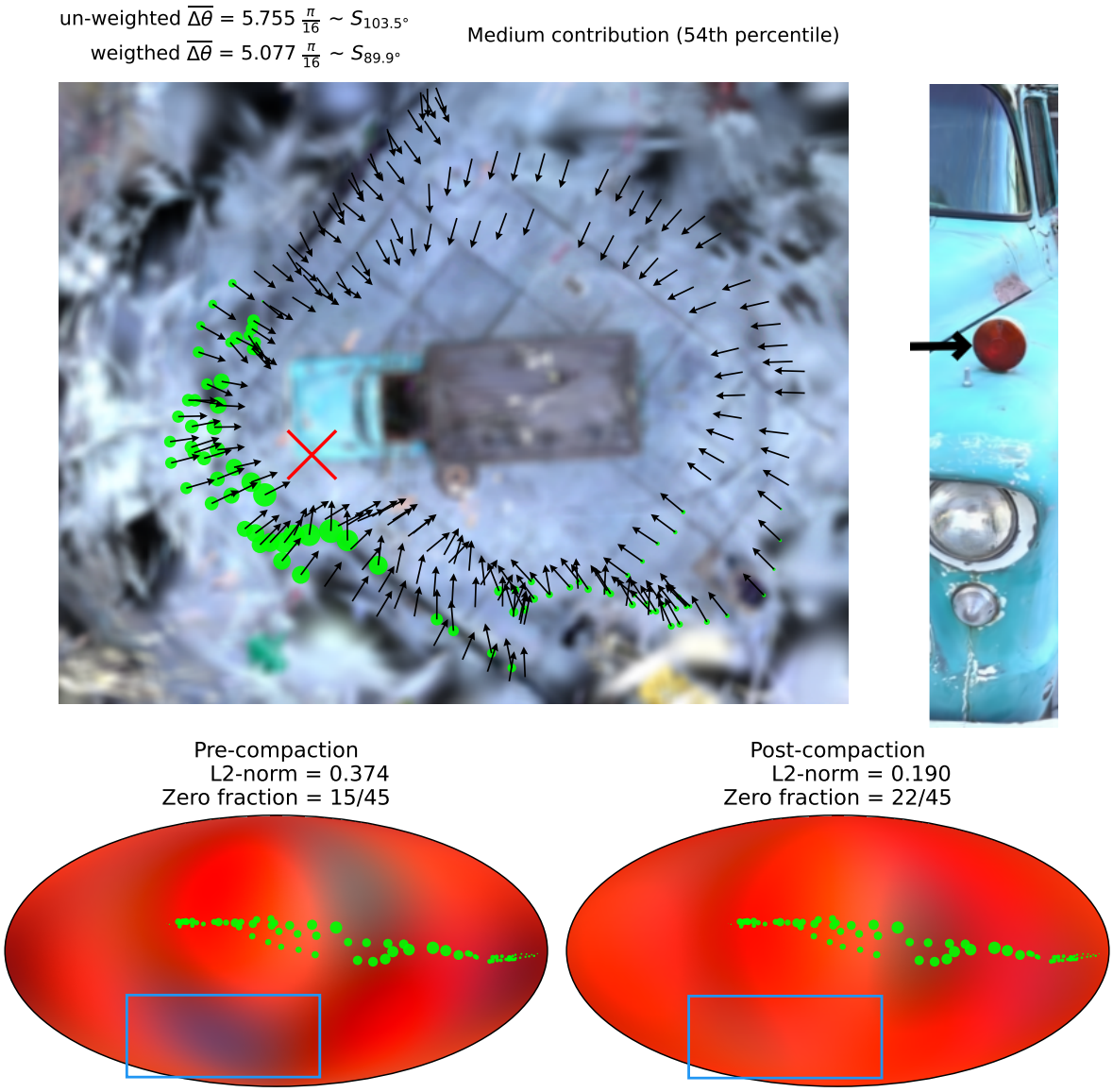



Figure 7.5: [Scenario 1] A medium contributing splat, part of the truck's red reflector, is considered. After energy compaction, the L2-norm decreased, the sparsity increased, and a hallucinated blue color (highlighted using a blue rectangle) was removed from an unsampled viewing area.

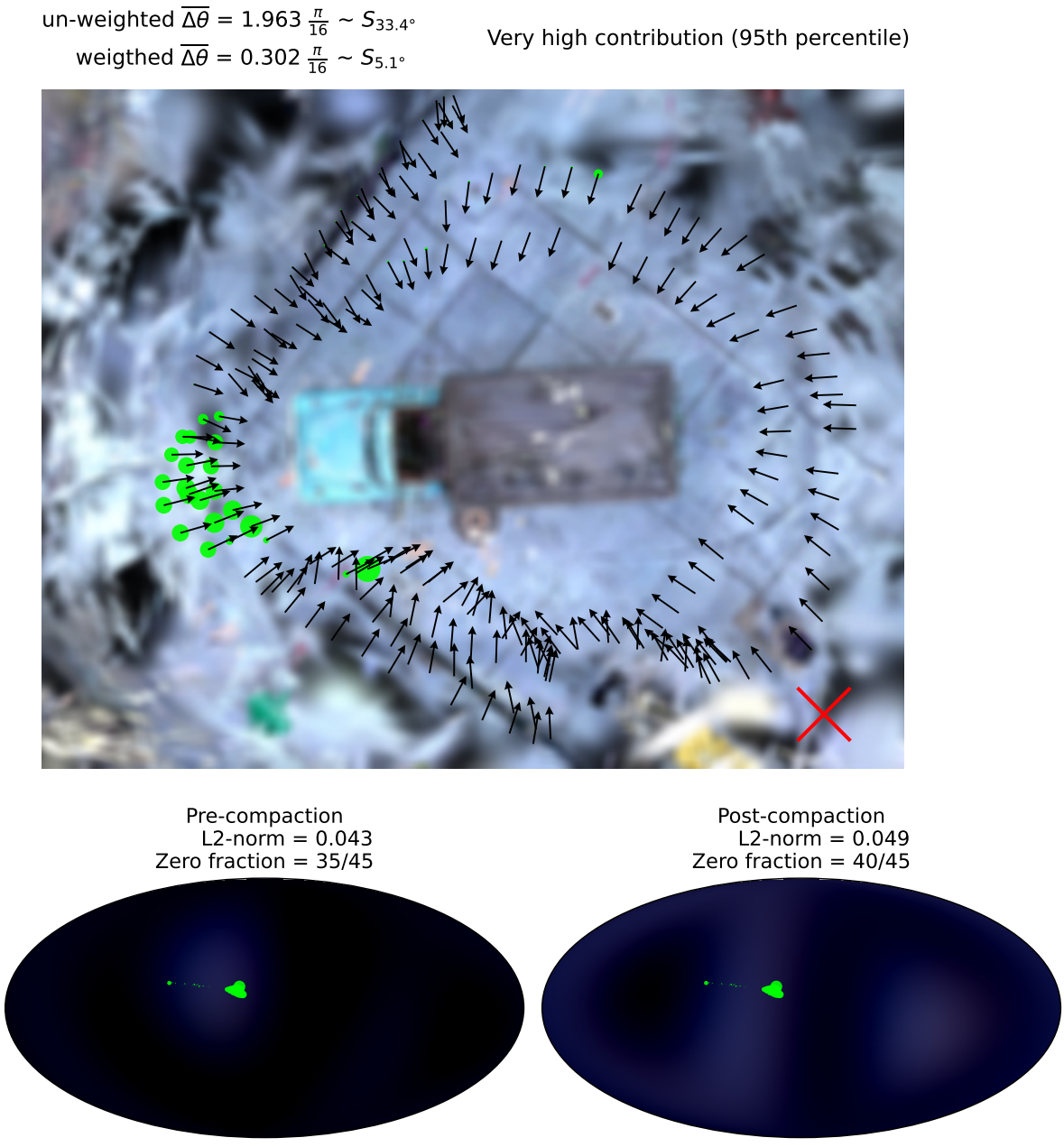



Figure 7.6: [Scenario 2] A very high contributing splat is considered. The splat is regularized minimally as it has a high contribution. Still, the number of non-zero AC coefficients was halved, primarily due to the splat's extremely unequal sampling.

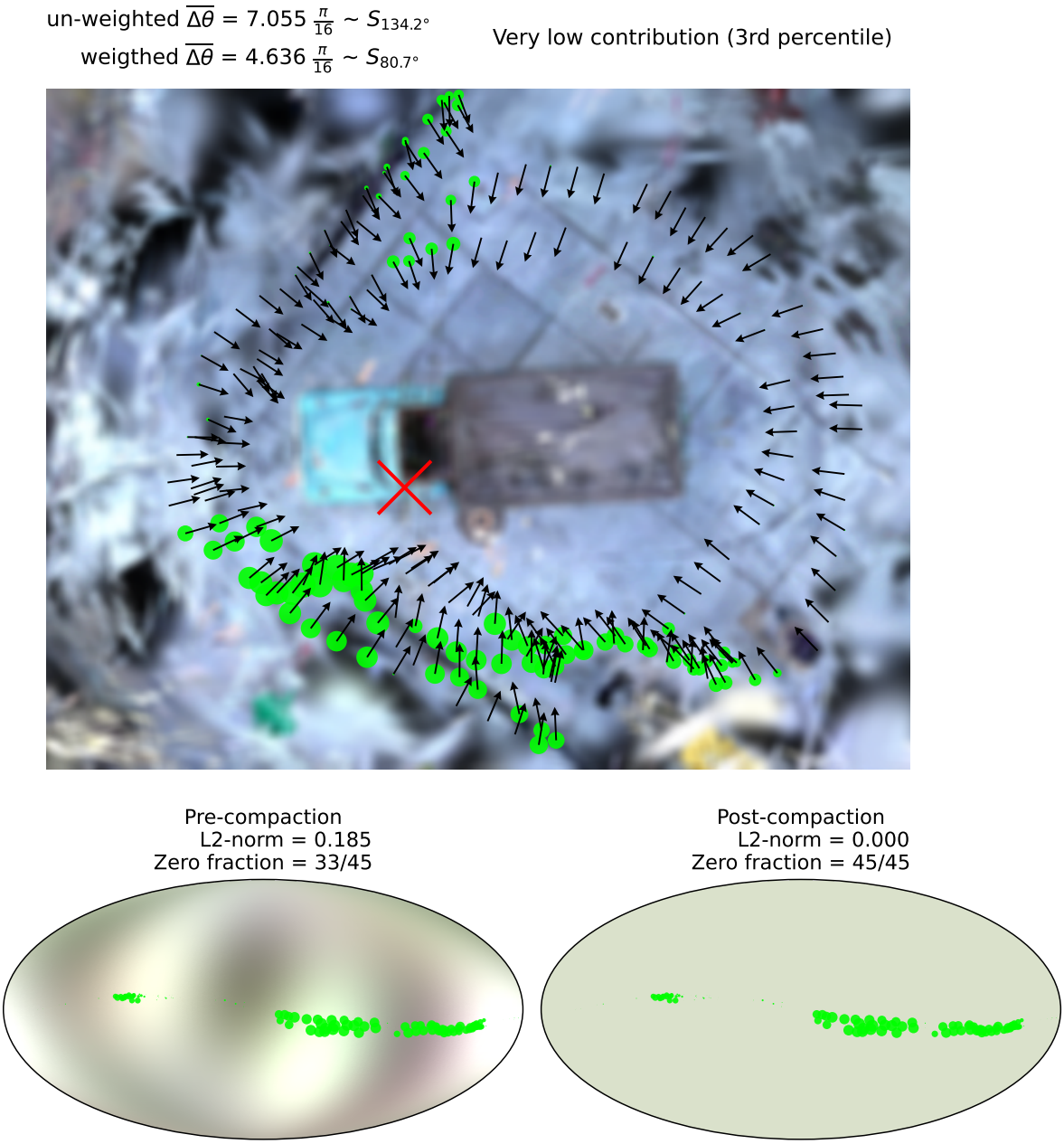



Figure 7.7: [Scenario 3] A low contributing splat is considered. The splat is regularized harshly with only the DC lighting coefficients being non-zero. As a result, the splat's sampled colors change significantly, but, the scene's quality is largely unaffected due to the splat's meager contribution.

Using these insights, an attempt was made to solve the linear system. Sadly, just calculating the coefficients using a modified rasterizer already required more than 4 hours, putting into question the practical applicability. As a result, I leave the study of the direct approach to future works.

# 8

# Encoder design

This chapter constructs an encoder (and decoder) using the compression techniques presented in chapters 6 and 7. Some tangental topics are briefly discussed including quantization, serialization, and lossless compression. The proposed encoder gives the final compression, quality, and runtime results.

## 8.1 Quantization

All attributes are represented as single precision floats, each taking up 4 bytes. This level of precision is unnecessary and detrimental to lossless compression. All attributes are therefore quantized to resolve this, tailoring the quantization on a per-attribute basis. Quantization can be tailored further on a per-splat basis, taking advantage of the observed differences in importance between splats (see Chapter 6), utilizing quantization as a less extreme form of culling. During culling, a splat was either important (and kept) or unimportant (and removed). Quantization can use a degree of importance. Additionally, an attribute's sensitivity can differ wildly from splat to splat, independent of the splat's importance. For example, a far-away splat is much less sensitive to an inaccuracy in its position than one close to a camera.

First, consider the positional attributes, as they are handled differently than the other attributes. The quantized positions of the splats are represented by the centers of the leaves of an octree, where each leaf corresponds with exactly one splat. To increase the precision of the $j$-th splat's position, the leaf in which it resides can be repeatably split until the Euclidean distance between the leaf center and the $j$-th splat's position is smaller than some threshold $\gamma_j$. Using the hyperparameter $\mathrm{PF}$ and the Euclidean distance between the $j$-th splat and its closest camera $l_j^{\mathrm{min}}$, the proposed encoder sets $\gamma_j$ to:

$$\gamma_j = \mathrm{PF} \cdot l_j^{\mathrm{min}} \tag{8.1}$$

This is based on the fact that if a splat is a distance $l$ removed from a camera, then a small change $dx$ in a positional attribute will cause the projected splat in screen space to move approximately proportional to $\frac{dx}{l}$. The base precision factor $\mathrm{PF}$ is a hyperparameter, I found $\mathrm{PF} = 3 \cdot 10^{-5}$ to work well for the scene Truck. Note that the contribution of a splat is not used in the positional quantization, as this empirically did not help improve compression performance further.

The non-positional attributes are uniformly quantized by converting them to integers using the following quantization function:

$$q(x) = \left\lfloor \frac{1}{2} + x \cdot \mathrm{PF} \right\rfloor \tag{8.2}$$

De-quantizing such an attribute is as simple as dividing the quantized value by the hyperparameter $\mathrm{PF}$. The precision factor $\mathrm{PF}$ differs from attribute to attribute. Differing them further on a per-splat basis has great compression potential, however, the decoder needs to know what value of $\mathrm{PF}$ was used to quantize each splat's attribute. Signaling this value is likely too expensive, but a good heuristic might exist to infer these values. Designing such a heuristic is hard as only limited information is available during de-quantization. This work does not alter $\mathrm{PF}$ on a per-splat basis, but future works could explore this avenue. I suspect this will primarily be of use for attributes other than the AC lighting coefficients, as energy compaction already ensures that less important splats are regularized more harshly, largely circumventing the need to apply harsher quantization.

Ideally, the underlying distribution of quantized values should be consistent across all attributes. A straightforward method that moves towards this ideal is subtracting the attribute's mean from each sample and dividing this by the attribute's standard deviation. This work does no such operation and also leaves this to future works.

## 8.2 Serialization

Before compressing and saving the scene representation, it is first converted into a bitstream. The bitstream starts with general information about the representation and the quantization. This includes:

1. Number of splats, spherical harmonics, and color channels.
2. The used color space.
3. All $\mathrm{PF}$ values.

Afterward, the positional information and splat order are communicated simultaneously by sending an octree representation of the splats' positions. The octree is serialized by iterating depth first over all nodes and serializing the 'occupancy' of each encountered node. The occupancy of a node is a single byte where the $i$-th bit denotes whether it has a child in the $i$-th octant. The resulting ordering of the splats places spatially close splats, close together. The quantized non-positional attributes are subsequently serialized using the octree's splat order. Each attribute is serialized contiguously, iterating over all splats before serializing the next attribute with the hope that lossless compression can exploit any present spatial correlation. For some attributes, additional 'tricks' are used:

1. **DC lighting coefficients**
   Differential encoding is used to encode the DC lighting coefficients. Surprisingly, this has a minor effect on the final file size. I hypothesize that: (1) lossless compression already detects the correlation of subsequent quantized DC lighting coefficients without differential encoding, and (2) splats' DC lighting coefficients are less spatially correlated than, for example, the DC coefficients of JPEG blocks.

2. **Quaternion values**
   The rotation of a splat is encoded using a quaternion representation which consists of 4 values. These values are normalized, allowing one value to be omitted with only the omitted value's sign needing to be signaled. There are

different approaches to choosing the value to omit. For example, you could select this dynamically and signal which value was omitted. The proposed encoder simply omits the first value. Further nuances concerning the effect of quantization on this process are not discussed.

3. **Opacity**
   The decoder boosts the opacity during de-quantization by using reconstruction values off-center from the quantization regions. Practically, where previously de-quantization was done using $\frac{q(x)}{\text{PF}}$, now, with a boost $b$, the reconstructed value becomes $\frac{q(x)+b}{\text{PF}}$. Boosting is only used for the opacity, where $b = 0.33$ was found to work well. This value can also be signaled in the bitstream.

## 8.3 Bitstream compression

The serialized representation can be losslessly compressed using lossless compression techniques, e.g. Huffman coding and run-length encoding. I use zstd [1], an out-of-the-box lossless compression algorithm/standard, which takes in a bitstream and outputs a compressed bitstream. Throughout this work, I use the default zstd settings with the maximum compression level (22). The size of the compressed bitstream is the final compressed file size.

## 8.4 Order of steps

The proposed encoder has three major, loss introducing, steps. The order in which they are executed is important as their properties differ:

- **Culling**
  Slow, but speeds up all operations to come, with large initial gains. Later iterations are sensitive to changes to the scene's representation.
- **Spherical harmonics energy compaction**
  Faster than culling, but still slow. Fairly insensitive to changes to the scene's representation.
- **Quantization**
  Fast and insensitive to the scene's representation.

Due to culling speeding up other operations, the encoder starts by running most of its allocated culling iterations. After achieving the bulk of the performance benefits from culling, all attributes, except the lighting coefficients, are quantized and de-quantized. This provides the subsequent energy compaction step with a representation already very close to the final representation. The new lighting coefficients are quantized and de-quantized, just as the other attributes, such that

[1] https://github.com/facebook/zstd

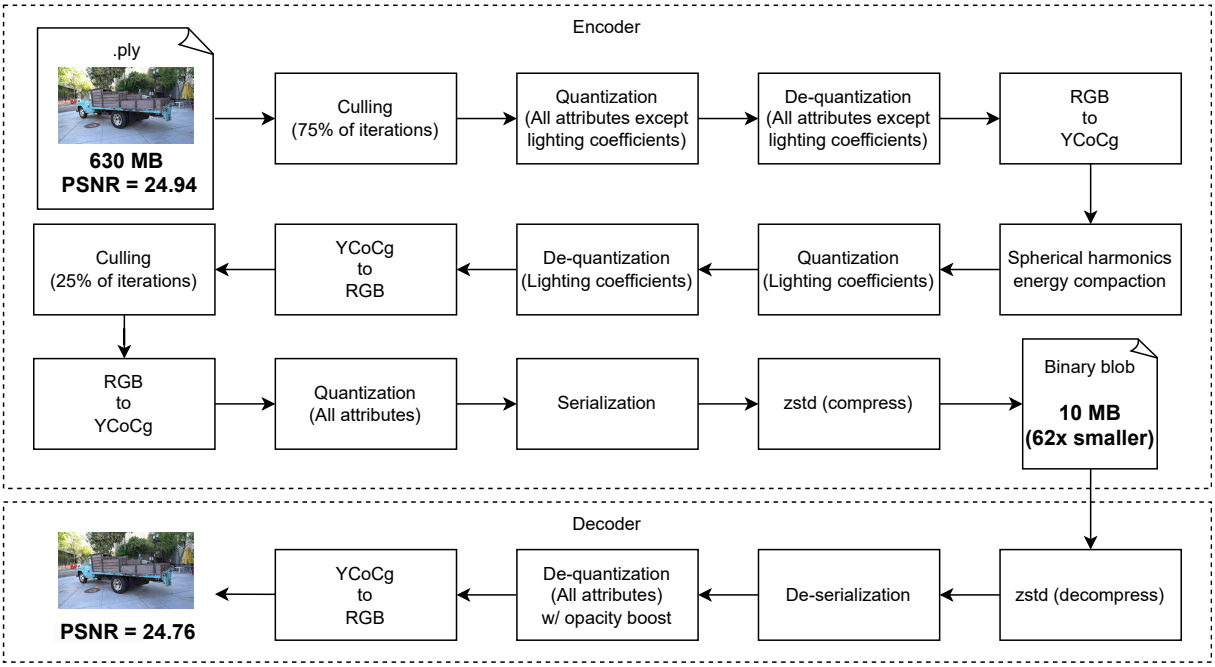



Figure 8.1: Schematic diagram of the proposed encoder and decoder.

the remaining culling iterations are given as close to the decoded representation as possible. After the last culling iteration, all lossless steps are finished and the quantized scene is serialized. This bitstream is then losslessly compressed using zstd. The proposed decoder decompresses, deserializes, and de-quantizes the bitstream in that order. Additionally, the de-quantization step boosts the opacity (as discussed previously). Figure 8.1 provides an overview of the encoder and decoder with achievable metrics for the scene Truck.

## 8.5 Results

All PF values were hand-optimized such that their quality-compression trade-off was comparable to that of culling. Doing this accurately is hard as the proposed encoder is not deterministic due to race conditions in the accumulation of floats during culling, with final filesize differing as much as $\pm$ 5% between subsequent runs. Additionally, zstd can be rather noisy when reacting to small hyperparameter changes. Nevertheless, these hand-optimized values become part of the default settings. Table 8.1 lists the proposed method's result on various scenes using these default settings. In conjunction, Figure 8.2 gives an idea of the achieved quality and introduced artifacts. The Truck scene was extensively used to set all hyperparameters and test/validate ideas and gives a good idea of the achievable quality if the hyperparameters are tuned. In contrast, all other scenes are instances of 'in-the-wild' performance. The achieved compression factor differs substantially between scenes, primarily due to varying degrees of culling effectiveness (as seen in Table 6.1). Depending on the scene, the non-culling compression steps achieve a compression factor of around 13x to 16x. To understand how this was accomplished, I further analyze the compression of each attribute class individually by observing how much smaller the final size would be were that attribute class removed from the bitstream. This analysis is somewhat spurious as compressed bitstreams can not

| Scene | PSNR$^{\uparrow}$ (+/-$^{\uparrow}$) | SSIM$^{\uparrow}$ (+/-$^{\uparrow}$) | LPIPS$^{\downarrow}$ (+/-$^{\downarrow}$) | Size$^{\downarrow}$ (compression factor$^{\uparrow}$) | FPS$^{\uparrow}$ (+/-$^{\uparrow}$) | Runtime$^{\downarrow}$ |
|---|---|---|---|---|---|---|
| Drjohnson [32] | 29.48 (-0.54) | 0.887 (-0.009) | 0.275 (+0.027) | 8.8 MB (96x) | 220 (+117) | 14.7 min |
| Playroom [32] | 29.47 (-0.45) | 0.884 (-0.017) | 0.274 (+0.030) | 6.9 MB (91x) | 220 (+125) | 8.9 min |
| Train [30] | 22.06 (+0.29) | 0.786 (-0.019) | 0.246 (+0.028) | 8.3 MB (35x) | 125 (+35) | 9.2 min |
| Truck [30] | 24.76 (-0.18) | 0.854 (-0.017) | 0.177 (+0.022) | 10 MB (62x) | 121 (+48) | 9.7 min |

Table 8.1: The proposed encoder/decoder's results for various scenes.

| Attribute class | Post-culling size (% of total) | Final compressed size (% of total) | Compression factor |
|---|---|---|---|
| Position | 8.4 MB (5.1%) | 2.5 MB (24.0%) | 3.4x |
| Scale | 8.4 MB (5.1%) | 1.7 MB (16.6%) | 4.9x |
| Opacity | 2.8 MB (1.7%) | 0.5 MB (5.1%) | 5.4x |
| Rotation | 11.2 MB (6.8%) | 1.9 MB (18.9%) | 5.8x |
| DC lighting coefficients | 8.4 MB (5.1%) | 1.8 MB (17.5%) | 4.6x |
| AC lighting coefficients | 125.9 MB (76.3%) | 1.8 MB (17.9%) | 68x |
| Total | 165.1 MB (100%) | 10.2 MB (100%) | 16x |

Table 8.2: Effect of the non-culling steps on the size of different attributes for the scene Truck. An attribute's compressed size is approximated through an ablation study.

be dissected into different parts due to lossless compression operating on the whole bitstream. Nevertheless, Table 8.2 presents these results and reveals that AC lighting coefficients are compressed approximately 68x while all other attributes are compressed anywhere from 3x to 6x. The lighting coefficients' high compression factor is partly the result of spherical harmonics energy compaction, without which, the compression factor would be more than halved.

The default configuration focuses on quality over runtime, requiring around 9 minutes for 3 of the 4 scenes. This runtime can easily be pushed under a minute by using fewer culling iterations, using one-pass regularization, porting the energy compaction to the GPU, and using a high-end GPU (compared to my GTX 1660 Ti).

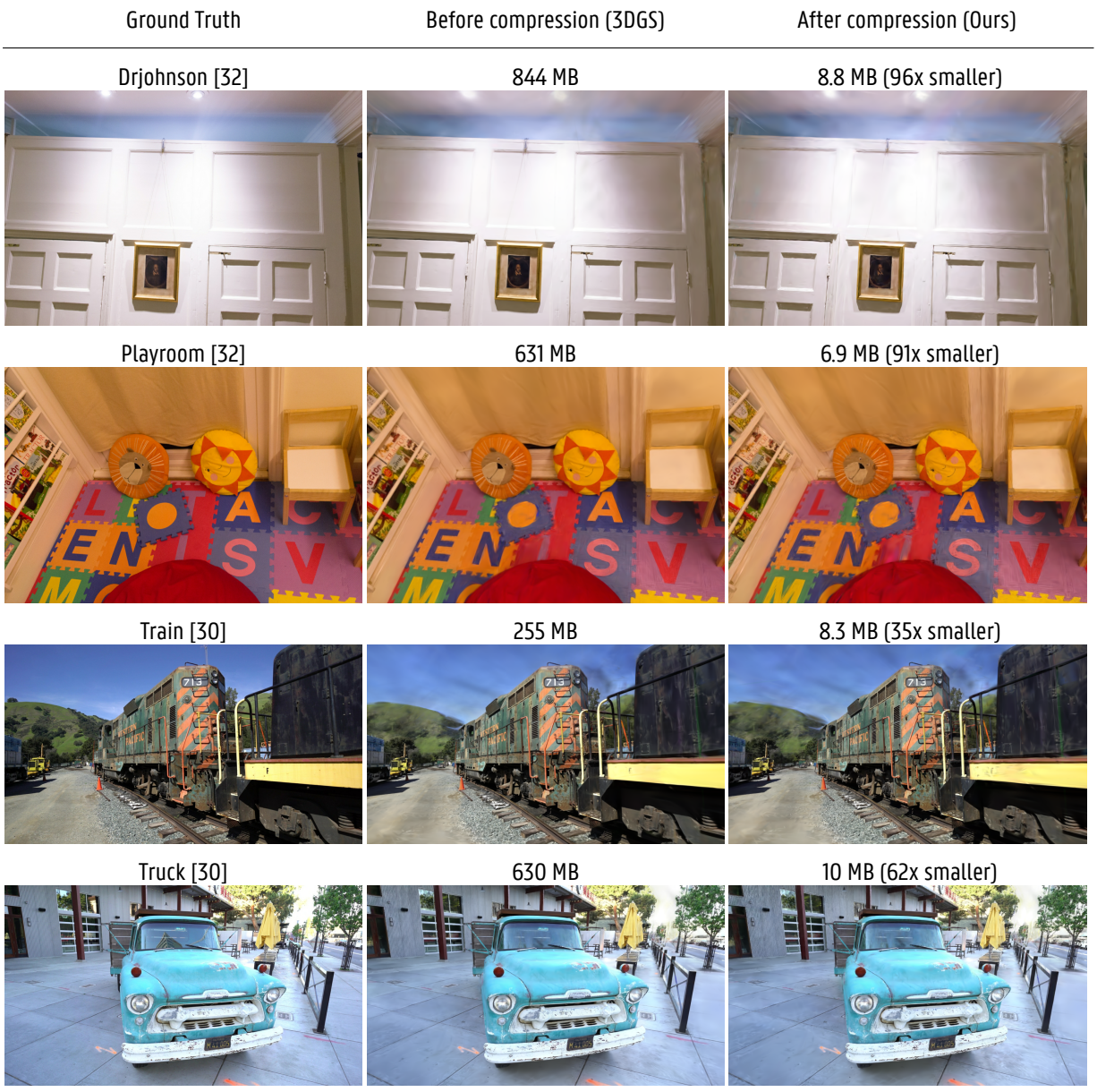



Figure 8.2: A comparison of test images before and after compression with corresponding ground truth images. The test images correspond with those presented by 3DGS [2].

# 9

# Future works

Some directions for future works were already mentioned in the previous chapters. An overview is given:

- (Section 5.2) Develop a method to generate $Q_c$ exclusively based on the scene without relying on camera information.
- (Section 6.4) Explore alternate culling controller designs.
- (Section 6.3) Investigate selectively re-rendering (parts of) frames after each culling iteration, removing splats in between render passes, and incorporating culling into the training loop.
- (Section 6.3.1) Decrease the number of culling iterations required by adopting a more optimistic approach.
- (Section 7.4.1: Sparsity) Explore more advanced algorithms, e.g. clique-based graph algorithms, to reduce the number of column vectors during spherical harmonics energy compaction.
- (Section 7.4.2) Investigate parallelizing the proposed spherical harmonics energy compaction technique.
- (Section 7.6) Further study the direct approach to energy compaction, including solving the underlying system if feasible.
- (Section 8.1) Explore splat tailored quantization for non-positional attributes.
- (Section 8.1) Study the effect of the quantized values' distribution changing throughout the bitstream and ways to unify the distribution.

Besides the above research directions, I also propose the following future research directions:

1. **Hyper parameters**
   This work uses numerous hyperparameters ($\triangle\text{MSE}_{\texttt{MAX}}$, $a$, $\lambda$, $\alpha$, all PF values, number of culling iterations, all zstd settings). Correctly setting these values is an arduous task, this work focussed on finding a single good working set of hyperparameters for the compression of the scene Truck. Future research could analyze how the hyperparameters, for a given scene, could be set given a compression level and speed.

2. **Rotational symmetry**

   3D Gaussians exhibit varying degrees of rotational symmetry, each having at least reflective symmetry concerning three planes, leading to 8 'symmetric variants'. Additional symmetries may arise, for example, an isotropic splat is rotationally invariant. Each of these symmetric variants leads to the same geometry. An encoder could confine itself to a single variant, reducing the range of attribute values. However, nearly all rotations already align with a single variant in the initial configuration. Consequently, I anticipate the potential for compression to be minimal. Nonetheless, a more comprehensive analysis would be insightful.

3. **Using CPU and GPU simultaneously**

   During culling iterations, the CPU remains idle when the GPU is engaged. Adapting the encoder to harness this idle CPU time could significantly enhance performance.

4. **Other loss metrics**

   This work predominantly focussed on squared error as a loss metric. Future works could investigate the applicability of the proposed technique to other loss metrics such as L1 loss and SSIM, both of which are used by 3DGS.

5. **Scene alignment**

   The Truck scene is not level, Figure 7.1 shows this as the samples do not follow the equator exactly. An aligned scene has two notable benefits: (1) planes of splats in the scene will be more axis-aligned, resulting in a more regular octree, and (2) the spherical harmonics will be sampled in a more axis-aligned way, increasing the sparsity of lighting coefficients due to additional parallel column vectors. Scene alignment also poses challenges, requiring adjustments to the positions and rotations of both splats and cameras. Furthermore, the real-valued spherical harmonics must be rotated, which is non-trivial.

# Conclusion

This dissertation proposes two novel compression techniques and an associated encoder/decoder for 3D Gaussian splatting scene compression. All presented techniques are post-processing steps, operating independently of the training loop. Additionally, this is the first compression work not to utilize gradient descent-based steps. Despite this, experiments on 4 scenes show that compression factors from 35x to 96x are achievable with minor visual artifacts.

The first novel compression technique removes between 54% and 84% of splats, leveraging a novel modified 3DGS rasterizer to efficiently and accurately calculate a splat's contribution and its removal impact on the PSNR. Removing this many splats also significantly increases the framerate of a scene, with some scenes more than doubling their framerate. The second technique lowers the entropy of the AC lighting coefficients using a heavily modified version of ridge regression. Experiments show that the proposed method can identify and exploit compression opportunities, increasing the sparsity of AC lighting coefficients to 95% for some scenes. The second technique's applicability outside of compression is also briefly demonstrated by, for example, removing a hallucinated blue color from an inherently red object. Both techniques are combined into an encoder-decoder pair that handles tangential topics such as quantization and lossless compression.

With 255-844 MB scenes being compressed to 7-10 MB, 3DGS is now comparable with NeRF regarding space requirements, solving 3DGS's largest comparative downside. Moreover, this work only scratched the surface of 3DGS scene compression, with numerous yet unexplored research directions.

# Bibliography


[1] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, "Nerf: Representing scenes as neural radiance fields for view synthesis," in *ECCV*, 2020.

[2] B. Kerbl, G. Kopanas, T. Leimkühler, and G. Drettakis, "3d gaussian splatting for real-time radiance field rendering," *ACM Transactions on Graphics*, vol. 42, no. 4, July 2023. [Online]. Available: https://repo-sam.inria.fr/fungraph/3d-gaussian-splatting/

[3] J. T. Barron, B. Mildenhall, M. Tancik, P. Hedman, R. Martin-Brualla, and P. P. Srinivasan, "Mip-nerf: A multiscale representation for anti-aliasing neural radiance fields," *ICCV*, 2021.

[4] J. T. Barron, B. Mildenhall, D. Verbin, P. P. Srinivasan, and P. Hedman, "Mip-nerf 360: Unbounded anti-aliased neural radiance fields," *CVPR*, 2022.

[5] ——, "Zip-nerf: Anti-aliased grid-based neural radiance fields," *ICCV*, 2023.

[6] J. C. Lee, D. Rho, X. Sun, J. H. Ko, and E. Park, "Compact 3d gaussian representation for radiance field," *arXiv preprint arXiv:2311.13681*, 2023.

[7] W. Morgenstern, F. Barthel, A. Hilsmann, and P. Eisert, "Compact 3d scene representation via self-organizing gaussian grids," 2023.

[8] K. Navaneet, K. P. Meibodi, S. A. Koohpayegani, and H. Pirsiavash, "Compact3d: Compressing gaussian splat radiance field models with vector quantization," 2023.

[9] S. Niedermayr, J. Stumpfegger, and R. Westermann, "Compressed 3d gaussian splatting for accelerated novel view synthesis," 2023.

[10] Z. Fan, K. Wang, K. Wen, Z. Zhu, D. Xu, and Z. Wang, "Lightgaussian: Unbounded 3d gaussian compression with 15x reduction and 200+ fps," 2023.

[11] G. Lippmann, "Épreuves réversibles donnant la sensation du relief," *J. Phys. Theor. Appl.*, vol. 7, no. 1, pp. 821–825, 1908.

[12] E. H. Adelson and J. R. Bergen, "The plenoptic function and the elements of early vision," in *Computational Models of Visual Processing*. MIT Press, 1991, pp. 3–20.

[13] S. E. Chen and L. Williams, "View interpolation for image synthesis," in *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '93. New York, NY, USA: Association for Computing Machinery, 1993, p. 279–288. [Online]. Available: https://doi.org/10.1145/166117.166153

[14] T. Müller, A. Evans, C. Schied, and A. Keller, "Instant neural graphics primitives with a multiresolution hash encoding," *ACM Trans. Graph.*, vol. 41, no. 4, pp. 102:1–102:15, Jul. 2022. [Online]. Available: https://doi.org/10.1145/3528223.3530127

[15] W. Hu, Y. Wang, L. Ma, B. Yang, L. Gao, X. Liu, and Y. Ma, "Tri-miprf: Tri-mip representation for efficient anti-aliasing neural radiance fields," in *ICCV*, 2023.

[16] N. Snavely, S. Seitz, and R. Szeliski, "Photo tourism: exploring photo collections in 3d. acm trans graph 25(3):835-846," *ACM Trans. Graph.*, vol. 25, pp. 835–846, 07 2006.

[17] M. Zwicker, H. Pfister, J. van Baar, and M. Gross, "Ewa volume splatting," in *Proceedings Visualization, 2001. VIS '01.*, 2001, pp. 29–538.

[18] R. Verhack, T. Sikora, G. Van Wallendael, and P. Lambert, "Steered mixture-of-experts for light field images and video: Representation and coding," *IEEE Transactions on Multimedia*, vol. 22, no. 3, pp. 579–593, 2020.

[19] R. Verhack, N. Madhu, G. Van Wallendael, P. Lambert, and T. Sikora, "Steered mixture-of-experts approximation of spherical image data," in *2018 26th European Signal Processing Conference (EUSIPCO)*, 2018, pp. 256–260.

[20] Y.-C. Guo, D. Kang, L. Bao, Y. He, and S.-H. Zhang, "Nerfren: Neural radiance fields with reflections," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2022, pp. 18 409–18 418.

[21] R. Martin-Brualla, N. Radwan, M. S. M. Sajjadi, J. T. Barron, A. Dosovitskiy, and D. Duckworth, "NeRF in the Wild: Neural Radiance Fields for Unconstrained Photo Collections," in *CVPR*, 2021.

[22] R. Wu, B. Mildenhall, P. Henzler, K. Park, R. Gao, D. Watson, P. P. Srinivasan, D. Verbin, J. T. Barron, B. Poole, and A. Holynski, "Reconfusion: 3d reconstruction with diffusion priors," *arXiv*, 2023.

[23] C. Reiser, S. Peng, Y. Liao, and A. Geiger, "Kilonerf: Speeding up neural radiance fields with thousands of tiny mlps," in *International Conference on Computer Vision (ICCV)*, 2021.

[24] H. Turki, D. Ramanan, and M. Satyanarayanan, "Mega-nerf: Scalable construction of large-scale nerfs for virtual fly-throughs," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2022, pp. 12 922–12 931.

[25] J. Lin, Z. Li, X. Tang, J. Liu, S. Liu, J. Liu, Y. Lu, X. Wu, S. Xu, Y. Yan, and W. Yang, "Vastgaussian: Vast 3d gaussians for large scene reconstruction," in *CVPR*, 2024.

[26] Z. Chen, T. Funkhouser, P. Hedman, and A. Tagliasacchi, "Mobilenerf: Exploiting the polygon rasterization pipeline for efficient neural field rendering on mobile architectures," 2023.

[27] S. Shin and J. Park, "Binary radiance fields," 2023.

[28] R. Ramamoorthi and P. Hanrahan, "An efficient representation for irradiance environment maps," in *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, 2001, pp. 497–500.

[29] Sara Fridovich-Keil and Alex Yu, M. Tancik, Q. Chen, B. Recht, and A. Kanazawa, "Plenoxels: Radiance fields without neural networks," in *CVPR*, 2022.

[30] A. Knapitsch, J. Park, Q.-Y. Zhou, and V. Koltun, "Tanks and temples: Benchmarking large-scale scene reconstruction," *ACM Transactions on Graphics*, vol. 36, no. 4, 2017.

[31] R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang, "The unreasonable effectiveness of deep features as a perceptual metric," in *CVPR*, 2018.

[32] P. Hedman, J. Philip, T. Price, J.-M. Frahm, G. Drettakis, and G. Brostow, "Deep blending for free-viewpoint image-based rendering," *ACM Transactions on Graphics (SIGGRAPH Asia Conference Proceedings)*, vol. 37, no. 6, November 2018. [Online]. Available: http://www-sop.inria.fr/reves/Basilic/2018/HPPFDB18

# Appendices

## Appendix A: Derivation optimal color attributes

Using the alpha compositing definition (Equation 3.6) and the ground truth color $\mathbf{c}_{\text{gt}}(\mathbf{x})$, the squared error for a single pixel $\mathbf{x}$ is defined as follows:

$$\begin{aligned}\mathbf{SE}(\mathbf{x}) &= (\sum_{i=0}^{K-1} T_i \alpha_i \mathbf{c}_i - \mathbf{c}_{\text{gt}}(\mathbf{x}))^2 \\ &= \sum_{i=0}^{K-1}\sum_{j=0}^{K-1} T_i \alpha_i T_j \alpha_j \mathbf{c}_i \mathbf{c}_j - 2\mathbf{c}_{\text{gt}}(\mathbf{x}) \sum_{i=0}^{K-1} T_i \alpha_i \mathbf{c}_i + \mathbf{c}_{\text{gt}}(\mathbf{x})^2\end{aligned} \tag{1}$$

$\mathbf{c}_{\text{gt}}(\mathbf{x})^2$ is a constant value and is moved to the left-hand side as it will become irrelevant upon taking a derivative. Now consider the square error of all pixels for a single camera [1]:

$$\begin{aligned}\sum_{\mathbf{x}} \mathbf{SE}(\mathbf{x}) - \mathbf{c}_{\text{gt}}(\mathbf{x})^2 &= \sum_{i=0}^{P-1}\sum_{j=0}^{P-1} \mathbf{c}_i \mathbf{c}_j \sum_{\mathbf{x}} T_i \alpha_i T_j \alpha_j - 2\sum_{i=0}^{P-1} \mathbf{c}_i \sum_{\mathbf{x}} \mathbf{c}_{\text{gt}}(\mathbf{x}) T_i \alpha_i \\ &= \sum_{i=0}^{P-1}\sum_{j=0}^{P-1} \mathbf{c}_i \mathbf{c}_j G_{ij} - 2\sum_{i=0}^{P-1} \mathbf{c}_i \mathbf{H}_i\end{aligned} \tag{2}$$

$G_{ij}$ and $\mathbf{H}_i$ are shorthand for geometry-based factors that are independent of all color attributes. The splat colors are monochromatic per frame and can thus be taken outside of the sum across all pixels. The amount of splats considered also substantially increases to the number of splats in the scene $P$ as a frame can contain hundreds of thousands of splats, compared to a single pixel which typically only considers around 50 splats (see Figure 6.3). Finally, also consider multiple viewpoints:

$$\sum_{\mathbf{s}}\sum_{\mathbf{x}} \mathbf{SE}(\mathbf{x},\mathbf{s}) - \mathbf{c}_{\text{gt}}(\mathbf{x},\mathbf{s})^2 = \sum_{i=0}^{P-1}\sum_{j=0}^{P-1}\sum_{\mathbf{s}} \mathbf{c}_i(\mathbf{s})\mathbf{c}_j(\mathbf{s}) G_{ij}(\mathbf{s}) - 2\sum_{i=0}^{P-1}\sum_{\mathbf{s}} \mathbf{c}_i(\mathbf{s})\mathbf{H}_i(\mathbf{s}) \tag{3}$$

where a splat's color is defined using spherical harmonics

$$\mathbf{c}_i(\mathbf{s}) = \sum_{k=1}^{M} Y_k(\mathbf{s}(i))\mathbf{L}_k(i) \tag{4}$$

[1] This equations notation is more conceptual, as the definitions of $T_i$ and $\alpha_i$ are slightly different.

which in turn leads to

$$\begin{aligned}\sum_{\mathbf{s}}\sum_{\mathbf{x}}\mathbf{SE}(\mathbf{x},\mathbf{s})-\mathbf{c}_{\mathsf{gt}}(\mathbf{x},\mathbf{s})^2 &= \sum_{i=0}^{P-1}\sum_{j=0}^{P-1}\sum_{\mathbf{s}}(\sum_{l=1}^{M}Y_l(\mathbf{s}(i))\mathbf{L}_l(i))(\sum_{m=1}^{M}Y_m(\mathbf{s}(j))\mathbf{L}_m(j))G_{ij}(\mathbf{s})\\ &\quad-2\sum_{i=0}^{P-1}\sum_{\mathbf{s}}(\sum_{l=1}^{M}Y_l(\mathbf{s}(i))\mathbf{L}_l(i))\mathbf{H}_i(\mathbf{s})\\ &=\sum_{i=0}^{P-1}\sum_{j=0}^{P-1}\sum_{l=1}^{M}\sum_{m=1}^{M}\mathbf{L}_l(i)\mathbf{L}_m(j)\sum_{\mathbf{s}}Y_l(\mathbf{s}(i))Y_m(\mathbf{s}(j))G_{ij}(\mathbf{s})\\ &\quad-2\sum_{i=0}^{P-1}\sum_{l=1}^{M}\mathbf{L}_l(i)\sum_{\mathbf{s}}Y_l(\mathbf{s}(i)\mathbf{H}_i(\mathbf{s})\\ &=\sum_{i=0}^{P-1}\sum_{j=0}^{P-1}\sum_{l=1}^{M}\sum_{m=1}^{M}\mathbf{L}_l(i)\mathbf{L}_m(j)\tilde{G}_{kj,lm}-2\sum_{i=0}^{P-1}\sum_{l=1}^{M}\mathbf{L}_l(i)\tilde{\mathbf{H}}_{k,l}\end{aligned} \tag{5}$$

$$\tilde{G}_{kj,lm}=\sum_{\mathbf{s}}Y_l(\mathbf{s}(k))Y_m(\mathbf{s}(j))G_{kj}(\mathbf{s}) \tag{6}$$

$$\tilde{\mathbf{H}}_{k,l}=\sum_{\mathbf{s}}Y_l(\mathbf{s}(k))\mathbf{H}_k(\mathbf{s}) \tag{7}$$

## Appendix B: Derivation approximation $G_{kj}(\mathbf{s})$

$G_{kj}(\mathbf{s})$ can be approximated as follows:

$$\begin{aligned}G_{kj}(\mathbf{s})&=\sum_{\mathbf{x}}T_k\alpha_kT_j\alpha_j\\ &\approx\overline{T_k}\,\overline{T_j}\sum_{\mathbf{x}}\alpha_k\alpha_j\\ &\approx\overline{T_k}\,\overline{T_j}S_{kj}^{\mathsf{2D}}(\mathbf{s})(\sum_{\mathbf{x}}\alpha_k)(\sum_{\mathbf{x}}\alpha_j)\\ &\approx S_{kj}^{\mathsf{2D}}(\mathbf{s})(\sum_{\mathbf{x}}T_k\alpha_k)(\sum_{\mathbf{x}}T_j\alpha_j)\\ &=S_{kj}^{\mathsf{2D}}(\mathbf{s})I_k(\mathbf{s})I_j(\mathbf{s})\end{aligned} \tag{8}$$

$$S_{kj}^{\mathsf{2D}}(\mathbf{s})=\frac{1}{2\pi}\frac{\exp(-\frac{1}{2}\boldsymbol{\eta}_k^T\boldsymbol{\Sigma}_k\boldsymbol{\eta}_k)}{\sqrt{|\boldsymbol{\Sigma}_k|}}\frac{\exp(-\frac{1}{2}\boldsymbol{\eta}_j^T\boldsymbol{\Sigma}_j\boldsymbol{\eta}_j)}{\sqrt{|\boldsymbol{\Sigma}_j|}}\frac{\exp(\frac{1}{2}(\boldsymbol{\eta}_k+\boldsymbol{\eta}_j)^T(\boldsymbol{\Sigma}_k^{-1}+\boldsymbol{\Sigma}_j^{-1})^{-1}(\boldsymbol{\eta}_k+\boldsymbol{\eta}_j))}{\sqrt{|\boldsymbol{\Sigma}_k^{-1}+\boldsymbol{\Sigma}_j^{-1}|}} \tag{9}$$

$$\boldsymbol{\eta}_i=\boldsymbol{\Sigma}_i^{-1}\mathbf{p}_i \tag{10}$$

where two approximations are applied. First, the transmittance $T_k$ and $T_j$ remain roughly constant across a frame, allowing for some frame-level transmittance, $\overline{T_k}$ and $\overline{T_j}$, to be defined that is independent of $\mathbf{x}$. Second, the grid of pixels is assumed to be sufficiently dense such that $\sum_{\mathbf{x}}\alpha_k\alpha_j\propto\int\alpha_k\alpha_j$. Therefore, the product of $\sum_{\mathbf{x}}\alpha_k\alpha_j$ can be split using the fact

that the product of two normalized 2D-Gaussians is an unnormalized 2D-Gaussian with a scaling factor $S_{kj}^{2D}(\mathbf{s})$ [2]. As we are working in screen space, the covariance matrices and positions of the splats are the projected version for the given viewpoint $\mathbf{s}$. Also, note that $S_{kj}^{2D}(\mathbf{s})$ is itself again a 2D-Gaussian in both $\mathbf{p}_k$ and $\mathbf{p}_j$.

[2] https://web.archive.org/web/20190524151044/http://www.tina-vision.net/docs/memos/2003-003.pdf