Efficient and Interpretable Natural Image Representation with Structured 2D Gaussians

by

SoumyaDeep Chowdhury

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2026

© SoumyaDeep Chowdhury 2026

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Recent advances in generative modeling, and the representation of high-fidelity visual content have motivated the demand for efficient, interpretable, and geometrically meaningful image representations. This work investigates the use of 2D Gaussian primitives as an explicit representation for images. The central research question explores whether image-derived Gaussian geometry and residual-driven refinement can improve reconstruction relative to random initializations, in low-medium resolution natural images.

Unlike dense neural representations or JPEG, a Gaussian image representation describes image content with geometrically interpretable primitives with qualities of position, scale, orientation, and colour parameters. The ability to discern these qualities in the primitive grant physical meaning. This makes the Gaussian representation directly editable (an advantage over more established high-performance codecs), grants the ability to be rendered at any desired resolution, and can be viewed as a structured parameter space for images, while providing meaningful compression.

The main contribution of this work is a structure-aware initialization and refinement pipeline for Gaussian primitives to ultimately convert images to Gaussian representations, referred to as GS-Sketch. While pioneering methods utilize random initializations, a drawing-inspired process is formulated to heuristically propose Gaussian locations, scales, and orientations using multiscale image cues in a coarse-to-fine approach. The drawing-inspired approach first captures coarse coverage, then intermediary shading, and lastly fine-detail and high-frequency areas. Residuals, edge maps, Laplacian, and coherence responses are all used in a discrete method to guide primitive placement. Geometric division is proposed between disk-like Gaussians in smooth regions, particularly targeting coverage, and pin-like Gaussians in edge-dense areas and fine details.

Fully heuristically defining geometry allows the problem to become linear in Gaussian colour coefficients. Then, the coefficient fitting stage can be posed as a convex optimization problem which can be rapidly solved with an Alternating Direction Method of Multipliers (ADMM) formulation with pixel, gradient, Laplacian, inter-stage consistency, and reconstruction loss terms. This allows for edge fidelity, reconstruction fidelity and resource allocation to aid in later joint optimization. The initialization is treated with hole-patching algorithms to maintain coverage in sparse regions. During training a densification and pruning stage is proposed to reallocate Gaussians towards residual-driven high-error regions.

A secondary contribution is an image-parameter codec that maps Gaussian tensors to attribute images that pack the position, scale, rotation, and colour qualities. This pipeline has slower post-training encode times than common learned entropy or codebook encoders; however, it completely avoids per-image post-fit quantization training, significantly reducing first-compress time, which is often relevant in practical applications. To enforce smoothness in the images, and therefore increased storage reduction in final image formats (PNG, and WebP), a sorting contest is conducted by the codec that measures bitrate reductions from reconstruction-safe sorts. In these regimes, WebP offers a lower bitrate, higher encoding cost path, while PNG offers a higher bitrate but lower time.

Representation experiments against the GaussianImage benchmark show the structured initialization and refinement pipeline improves PSNR-related reconstruction terms across all tested Gaussian counts against the GaussianImage baseline. Across the Kodak dataset, the evaluated regimes are averaged to be 2.2k, 4.6k, 9.0k, and 12.1k Gaussians, with ranges. GS-Sketch outperforms the baseline in PSNR results with peak PSNR results at each regime improving from 27.47 to 28.09 dB, 29.56 to 30.81 dB, 32.04 to 33.49 dB, and 33.40 to 34.67 dB respectively for each budget regime. A sum variant of the pipeline is also proposed, which provides larger full pipeline speedup, and maintains higher PSNR results compared to the baseline at every regime. The overall pipeline with the sum-based variant produces speedups of approximately 6.8 times with PNG storage and 5.5 times with WebP storage. This speedup considers first-time compression of images, the crossing time for equivalent fit PSNR, and compares learned codebook quantization in the baseline and the proposed sorting contest in the proposed pipeline. This is compared against the baseline's final-fit PSNR time, and the subsequent learned compression stage.

Low and medium bitrate regimes are designed for the image codec, paired with the final WebP format, denoted as WebP Low and WebP Medium. For GS-Sketch, WebP Low reaches 28.496 dB at 0.731 bpp, and WebP Medium reaches 30.115 dB at 0.800 bpp, compared with native GaussianImage quantization (applied on the GS-Sketch fits) at 26.811 dB and 0.653 bpp. The 30 dB operating point is competitive with the published GaussianImage rate-distortion curve which achieves comparable Kodak PSNR at ≈ 1 bpp. Contextual results show the increased performance at these mid-range bpps. The results suggest that 2D Gaussian image representations provide a compact representation, while maintaining an interactive and geometrically interpretable state that can be used as image parameter representations for fast rendering and interactive editing applications.

Acknowledgements

I would like to thank my supervisor Dr. Zhou Wang, for his guidance and the freedoms granted in research directions. Committee member Dr. Oleg Michailovich for his guidance and technical insight throughout my terms. I would like to thank committee member Dr. Pin-Han Ho for their aid in the committee on short notice. Lastly, I would like to thank MASc Program Coordinator/Advisor Amber Beaudoin for their aid in organizing the thesis proceedings. Without these people this work would not have been possible.

I gratefully acknowledge the financial support provided by the Natural Sciences and Engineering Research Council of Canada (NSERC) through the Canada Graduate Scholarship Master's (CGS M) program.

Dedication

This is dedicated to my family who have always supported me.

Table of Contents

Author's Declaration ii

Abstract iii

Acknowledgements v

Dedication vi

List of Figures x

List of Tables xiii

1 Introduction 1

1.1 Motivations 1

1.2 Contributions 3

1.3 Organization of Thesis 4

2 Literature Review on Gaussian Image Representations 5

2.1 Established Image Compression 5

2.2 Learned and Implicit Compression Techniques 6

2.3 Relevance of Gaussian Image Representations 10

2.4 Gaussian Splatting Techniques 12

2.5 Conceptual Motivations from Prior Work . . . 17
2.6 Gaussian Image Methods . . . 17
2.7 Contribution Positioning . . . 19

3 GS-Sketch: a Structured Image Representation Method Using Gaussian Splats 21

3.1 Methods . . . 21
3.1.1 Problem Reformulation . . . 21
3.1.2 Region and Geometry Proposal . . . 27
3.1.3 Hole-Patching . . . 35
3.1.4 Training Integration . . . 37
3.1.5 Quantization Contest . . . 42
3.1.6 Experiment Details . . . 51
3.2 Results . . . 57
3.2.1 Low-Primitive Representation . . . 57
3.2.2 Compression Approaches . . . 60
3.3 Discussion . . . 63
3.3.1 Sharpness-Coverage Tradeoff . . . 63
3.3.2 Normalized Rendering . . . 65
3.3.3 Sorting Quantization Approaches . . . 66
3.3.4 Sorting Artifact and Behaviour Analysis . . . 68
3.3.5 Refinement with Densification and Pruning . . . 70
3.3.6 Practical Relevance of Gaussian Representations . . . 71
3.3.7 Established Image Compression Regimes . . . 74

4 Conclusions 87

4.1 Summary of Method and Results . . . 87
4.1.1 Primary Contributions . . . 87

4.1.2 Primary Results . . . 88
4.1.3 Conceptual Conclusions . . . 89
4.2 Further Work . . . 92
4.2.1 Limitations . . . 93
4.2.2 Implementation Improvements . . . 95

References **98**

APPENDICES **104**

A Supplementary Material **105**
A.1 Comparison Tables . . . 105
A.2 Further Discussion on Sorting Artifacts . . . 106
A.3 Relevant Code Snippets . . . 109
A.3.1 Computing geometry proposals . . . 109
A.3.2 ADMM Application . . . 112

List of Figures

2.1 Comparison of 3DGS[20] and GaussianImage [45] pipelines, illustrating the reduction in dimension, lack of initialization and refinement robustness, and additional compression elements in GaussianImage [45]. 15

2.2 ArXiv papers related to Gaussian image representations aggregated by year. 18

3.1 Disk and pin geometry splitting for LF (disks) and HF (pin) features . . . 29

3.2 Visual flow of hot initialization process, showing fine scale, medium scale, and large scale contributions. Note that targets for first pass only are shown. 32

3.3 Visualization of the several sorting methods outlined for the contest. Features are saved as RGB, x-y-rotation is combined as RGB, and scale channels are separated. 49

3.4 GS-Sketch, GS-Sketch(sum), and baseline GI systems at various Gaussian counts on Kodak Image 10. The quality of both initializations and final fits increases drastically with increasing Gaussian counts. At higher Gaussian counts, GS-Sketch consistently produces higher PSNR and edge PSNR over the other tested methods. 58

3.5 Effects of final fit from different systems on Kodim23.png with 2206 Gaussians. The proposed method provides emphasis on important high-frequency information, the normalized rendering is seen to adequately map smooth surface features with large Gaussians without artifacts. 60

3.6 Budget-Quality Curves for proposed GS-Sketch methods and baseline GI. . 76

3.7 Post-compression codec renders for kodim09.png, showing our method retains features in visually impactful regions of the image, after compression. Bpp benchmarks show that the method can be competitive with learned-codebook approaches, while providing less PSNR drop against the implemented native quantization approach. 77

3.8 Visual tradeoff in subtle texture and visual sharpness of prominent features in the human case of Kodim04.png, where our models achieve slightly lower MS-SSIM, particularly due to skin-texture representation as large smooth Gaussians, or uniform small Gaussians in random initializations. This shows our method trades smoothness in skin-like textures for sharpness in other facial feature aspects. 78

3.9 Texture tradeoff in low/stringent Gaussian counts, sparsity may result in black artifacts from lack of Gaussians in certain regions, however small textures such as the window pane features may be better represented. Random initializations generally have more Gaussians to provide more uniform texture coverage. 79

3.10 Benefit of normalized rendering in representation of uniform colour with sparse Gaussians. Note that in the actual rendering implementation, Gaussians have a maximum radius of effect, so they appear as ellipsoids for normalized rendering, but still can represent smooth colour in their regions. . 80

3.11 Benefit of normalized rendering in representation of edges along with large structured Gaussians, where summation causes colour brightening, normalization allows for a smooth transition to the intended brightness without complexity. 81

3.12 Training for 30000 iterations with random initializations, achieving final PSNR <12 dB for random initializations. This illustrates the convergence has a strong dependence on initialization. 81

3.13 Distribution of parameter values in a 16-bit linear quantized histogram for (a) minor scales (b) major scales, (c) (d) (e) R, G, B feature channels of GS-Sketch, GS-Sketch(Sum) and GI (baseline). 82

3.14 Sorting artifacts in kodim17.png at 4633 Gaussians. 83

3.15 Visualization of training effect of densification, with and without growth, on kodim01, using 2005 (2406 for growth allotted) Gaussians. Final and peak PSNR values are displayed. This illustrates the value of refinement in dealing with black artifacts common with normalized rendering in sparse scenarios. (Final fit values for 30000 iterations, while visuals for up to 20000 iterations are displayed). 84

3.16 Visualization of structure identification, colour transformations, and tied scale-structural edits. Edge brightening increases the identified edge Gaussians colour coefficients, Non-Edge Blur increases scales of the Gaussians not identified as edges, while Edge Blur blurs the identified edge Gaussians only. 85

3.17 Denoising (backward) steps with training MLP from initialization on fruits for 50 denoising steps on a randomized start 85

3.18 Visualization of forward steps, original image, reverse start, and output image 86

3.19 Rate distortion plots for the presented image metrics (c) MS-SSIM [40], (a) Edge-PSNR defined in Section 3.1.6, and (b) PSNR [18]. Taken at the regimes from N1000 to N7000, as defined in Table 3.3 86

4.1 Comparison between timing of the proposed GS-Sketch and GS-Sketch(Sum) variants compared to the original GaussianImage-RS [45] pipeline at N3000 (refer to Table 3.3) on the Kodak dataset [11]. Encoding time for baseline considers the per-image codebook learning time. Crossing times are taken from proportional training times of crossing iterations relative to final iterations. PSNR and bpp listed. 90

A.1 Pattern of sorting artifacts in high Gaussian count regions, tiled density estimation heatmap shown, and the final artifacts from Feature PCA (non-spatial) sorting. It can be seen that artifacts are specifically in regions with 256+ Gaussians contributing. 107

List of Tables

2.1 Scope and targets of previous 2D Gaussian Image works, highlighting differences in target Gaussian complexity, type of images, and resolution of images. 20

3.1 Quantization settings for parameters before image-codec saving at different regimes, high, medium, and low. Note resultant PSNR values differ from high and medium/low ranges disproportionately due to testing and optimization of quantization shown in Section 3.3.3. 51

3.2 MATLAB hot-initialization Gaussian count ranges before training and count matching. 54

3.3 Final fitted/evaluated Gaussian count ranges for each budget regime after densification growth. 54

3.4 Average final-fit reconstruction quality and training time by budget regime and system. 59

3.5 Average peak-fit PSNR and peak iteration by budget regime and system. . 59

3.6 Average crossing-fit PSNR and proportional training time by budget regime and system. 61

3.7 Average compression quality for the high-rate image-codec setting. Native GaussianImage quantization is repeated as a reference; the improved transform quantization results are discussed in the following low/medium-rate comparison. 62

3.8 Average timing for the high-rate image-codec setting. Native quantization includes codebook learning/quantization for total time, while winner encode reports only the final selected compression step. 62

3.9 Average compression quality for the low- and medium-rate transform image-codec settings, with native GaussianImage quantization repeated for comparison. . . . 63

3.10 Average timing for the low- and medium-rate transform image-codec settings, with native GaussianImage quantization repeated for comparison. For image codecs, total time includes the full adaptive contest and winner encode is only the selected packing/order. . . . 64

3.11 Average transform quantization diagnostics for the low codec setting. . . . 68

3.12 Average bitrate of sorting methods on WebP regime on low settings. Time is direct-sort safety test plus encoding time; for no sort it is encoding time only. . . . 69

3.13 Renderer-safe sorting counts and bitrate bounds on Kodak N03000 using WebP. Successful and failed sorts count direct no-restore sort attempts across all images and candidate orderings. . . . 70

3.14 Training snapshots for the practical densification ablation. The refinement run uses prune/train/densify with growth enabled. . . . 71

3.15 Training snapshots for the fixed-budget densification ablation. The refinement run uses prune/train/densify with zero net growth. . . . 71

A.1 Matched-bpp comparison between GS-Sketch WebP Medium and JPEG on Kodak. JPEG quality is the average quality setting selected to match the GS-Sketch bitrate. . . . 105

A.2 Fit fps aggregated by system and budget. Values are the mean of the fit fps column across images. . . . 106

A.3 Per budget PSNR results across the 24 Kodak images. Strict wins are every image that system got the best score of all 3 tested systems, while "beats GI" refers to when a structured system beats the GI baseline. . . . 106

Chapter 1

Introduction

This chapter covers the general motivations, primary contributions, and summarizes the organization of the remainder of the thesis.

1.1 Motivations

Images are often represented using uniform/regularly sampled grids of colour values. Normally, established image formats and codecs exploit the spatial, structural, transformed, and perceptual redundancies in the representations to reduce image storage requirements. An emerging alternative direction is to represent an image as the parameters of a function, or collection of explicit primitives. These representations do not merely store the raster samples, but they describe the process through which an image can be reconstructed at a given pixel coordinate.

Neural implicit representations model the process using a learned function, while Gaussian image representations utilize a collection of two-dimensional Gaussian primitives. Each Gaussian is described with its center, scale, orientation, and colour coefficient. Therefore, they provide a geometrically and spatially interpretable image state. The representation can be rasterized quickly, and evaluated on the continuous image plane. These properties are what separate Gaussian image representation from conventional and other newer codecs, whose decoded outputs are generally still fixed raster arrays, and from neural representations, which are represented as network parameters and not necessarily well-correlated with the visual content.

Nevertheless, Gaussian primitives can introduce a resource-allocation issue. For constrained Gaussian counts, representations require both broad coverage of smooth regions, and sufficient precisions for textures, edges, and other high-frequency regions. Randomly initialized primitives lack prior information about these structures. Subsequent gradient optimizations can naturally move Gaussians towards efficient areas, but the final solution may remain sensitive to its initialization, particularly at constrained primitive counts.

This thesis investigates whether image structure can be incorporated as prior information before joint Gaussian optimization. The central research question is then: whether a completely image-derived initialization of Gaussian geometry, and its continual allocation refinement using these features, can improve the representation of ordinary-resolution natural images under constrained budgets relative to random initializations. The latter process is reproduced using the GaussianImage [45] pipeline which applies such random initializations.

To investigate these concepts, the proposed GS-Sketch pipeline constructs Gaussian centers, scales, and orientations using a multiscale sketching inspired procedure. Large stages prioritize the broad coverage and levels of an image, intermediate scales begin shading the model and targeting regional variation, and fine stages target the edges and other high-frequency details. A heuristic split between disk-like Gaussians which are assigned to smooth regions, and anisotropic pin-like Gaussians which are assigned to coherent edges and fine structures. Residual, gradient, Laplacian, coherence, and local-scale responses are utilized to guide the geometry and center proposals of primitives. With fixed geometry, Gaussian coefficient fitting becomes a convex quadratic subproblem, which is solved using alternating direction method of multipliers (ADMM). The initialization is then refined through hole-patching/repairing, pruning, residual-driven densification, and finally joint optimization of Gaussian parameters.

Furthermore, this thesis investigates the effect of both additive and normalized Gaussian rendering. The normalized rendering formula allows more Gaussians to overlap and blend using relative Gaussian contributions, often representing smooth regions and prominent edges better. However, it also can introduce more sensitivity due to insufficient coverage, and is more reliant on proper initializations for convergence. These rendering techniques are compared to provide further insight into the relationship between initializations, coverage, and Gaussian blending.

A further contribution concerns the post-training compression stage. The raw fitted model tensors corresponding to each Gaussian parameter are transformed and scalar-quantized to

be packed into parameter images. These parameter images are stored using lossless PNG or lossless WebP formats. An adaptive sorting contest is used to evaluate alternative Gaussian row-orders which increase the spatial smoothness and reduce the final post-compression payload.

1.2 Contributions

The primary contributions of the thesis are summarized below:

Multiscale structure-aware Gaussian initialization. This work develops a coarse-to-fine approach to Gaussian geometry and center proposal, fully initializing Gaussian geometry from image content. The method utilizes gradient, Laplacian, residual, coherence, and local-scale responses to allocate disk and pin like primitives in smooth and high-frequency regions respectively.

Fixed-geometry coefficient optimization. When all geometry is defined, image reconstruction can be formulated as a linear combination of Gaussian geometry and colour coefficients. A convex quadratic objective combining pixel, Laplacian, x and y edge responses from Sobel and/or gradient approaches, and coefficient prior terms is formulated and solved using a multi-space ADMM implementation in MATLAB.

Hole-repair, refinement, and renderer analysis. Mitigation procedures for coverage artifacts and residual-driven prune-train-densify refinement procedures are introduced to repair coverage artifacts, and reallocate primitives during the joint parameter optimization phase. The normalized and additive rendering variants are evaluated to compare the reconstruction benefits, convergence behaviour, and sensitivity to sparse coverage.

Parameter-image storage. A secondary image-codec is created using simple scalar-quantization methods on the fitted model tensors. This method packs values into parameter images in the lossless PNG or WebP formats. A row-ordering contest is used to evaluate different discrete sorting procedures and their resultant compression, retaining only reconstruction valid configurations.

Evaluated on the Kodak dataset [11], GS-Sketch improves the peak-fit PSNR from the reproduced GaussianImage [45] baseline at all tested Gaussian count regimes. Using average counts of approximately 2.2k, 4.6k, 9.0k, and 12.1k Gaussians, the peak PSNR increases

from 27.47 to 28.09 dB, 29.56 to 30.81 dB, 32.04 to 33.49 dB, and 33.40 to 34.67 dB respectively. The GS-Sketch pipelines also attain a higher PSNR than the baseline substantially earlier in training. For example, at 5 thousand Gaussians, the final PSNR of the baseline was reached at 39.7 seconds, compared with 133.4 seconds for the complete baseline fit. The results indicate that prior information from explicit image structure can aid in reconstruction quality.

Likewise, the compression pipeline at the same at the medium profile using lossless WebP attains 30.115 dB PSNR and 0.8 bpp (where 30dB PSNR is not attained before 1 bpp for the GaussianImage-RS pipeline in their reported rate-distortion curve [45]). Such procedures also avoid per-scene training of learned quantization/vector-codebook methods.

1.3 Organization of Thesis

Chapter 2 shortly reviews established compression techniques, learned and implicit image representations, and focuses on Gaussian splatting, and recent two-dimensional Gaussian-image methods. Then in Chapter 3 the GS-Sketch formulation is proposed and implemented, evaluating the representation and compression performance on the Kodak natural image dataset. Finally, in Chapter 4 the tradeoffs and implementation decisions are analyzed, and limitations and conclusions are derived from the discussion along with directions for further development.

Chapter 2

Literature Review on Gaussian Image Representations

This chapter provides the necessary material to understand the niche of Gaussian image representations, and identifies the positioning of the contributions relative to other works. This review covers the representation and compression concepts required to position the role of structured 2D Gaussian image models. First we briefly consider established compression formats, then discuss learned and implicit image-compression systems, and then move towards Gaussian Splatting techniques, and specific innovations in representing images with 2D Gaussians. The discussion on representations outside Gaussians is limited, and for the purpose of comparison of fundamental concepts in Section 2.3.

2.1 Established Image Compression

Besides the outlined niche of Gaussian Image representations, it is relevant to define the common state of the art methods, especially those tested against and utilized in later sections of the work. The formats summarized here are for the purpose of general background for the comparison methods or backends, and/or in the experimental evaluation. They are not central to the contributions of this thesis, and only the aspects necessary to interpret the role in the experiments are discussed.

JPEG[19] JPEG is a standard image codec used for compression and designed for natural photographic imagery. The image is first transformed from the spatial domain into

frequency domain blocks using a discrete cosine transform, after which resulting coefficients are quantized and entropy coded. In general entropy coding is a process in which frequently occurring values in domains are represented with less data, while less frequent values are assigned longer sequences, it reduces redundancy without introducing loss. Most compression loss is introduced from the quantization, where high-frequency coefficients are often represented less precisely or discarded, allowing for a substantial reduction in file size. JPEG produces several common artifacts, though, as a result commonly smoothing, ringing, and blocking. JPEG is included as a conventional baseline references to put perspective of the current state of Gaussian-based image representations.

PNG[41] PNG is a lossless image-compression method that attempts to preserve decoded pixel values exactly. It reduces the redundancy using predictive filtering and DEFLATE [8] compression. It combines dictionary-based coding and entropy coding. As no quantization is applied, PNG does not introduce reconstruction error, but its file sizes are typicall larger than JPEG or WebP (both lossy and lossless[12]). Within this work PNG is used as the lossless reference container for Gaussian parameter images, similar to the role of WebP, but it should be noted that PNG encoding is generally faster as seen in Table 3.10.

WebP[12] WebP is a newer image format from Google that supports both lossy and lossless compression. Its lossy mode is designed as an efficiency improvement on JPEG, using block prediction, transform coding, quantization, and entropy coding to reduce file sizes while retrieving the same visual quality. The lossless form is applied to this work as the lossless container medium for quantized Gaussian parameter images. It serves a role similar to PNG, but uses a dedicated spatial prediction along with colour transforms and entropy coding that allow smaller files for suitable image content.

2.2 Learned and Implicit Compression Techniques

Established image codecs tend to be manually designed transformations, predictors, quantizers, and entropy-coding procedures. Conversely, learned image-compression methods optimize these components using a collection of training images. Commonly, learned codecs utilizes some transform to map an image to a latent tensor, quantizes the tensor, and entropy-codes the resulting discrete symbols, furthermore applying a synthesis transform which reconstructs the final image. Another related, but distinct, family uses implicit

neural representations. This family compresses an object utilizing a neural function that returns image values when prompted by pixel coordinates. The image is then represented by the parameters of the coordinate-based function for that image. These two approaches apply useful intermediates between conventional compression, and explicit Gaussian compression.

Balle17, learned transform coding [1] Balle et al. introduced a method to optimize an image compression architecture by training an analysis transform, synthesis transform, and a latent probability model which are all trained jointly. The simplified form of the codec can be written as:

$$\mathbf{y} = g_a(\mathbf{x}; \boldsymbol{\phi}), \qquad \hat{\mathbf{y}} = Q(\mathbf{y}), \qquad \hat{\mathbf{x}} = g_s(\hat{\mathbf{y}}; \boldsymbol{\theta}),$$

here $\mathbf{x}$ represents an input image. g_a is the analysis transform with parameters $\boldsymbol{\phi}$. $\mathbf{y}$ is the continuous latent representation. Then Q is the quantization operator, with $\hat{\mathbf{y}}$ as the discrete latent representation. Lastly, g_s is the synthesis transform with the parameters $\boldsymbol{\theta}$, giving the reconstructed image $\hat{\mathbf{x}}$. The quantization is discontinuous, and during training the quantization transformation, $Q(\mathbf{y})$, is often approximated by adding uniform noise or another differentiable relaxation.

The training is conducted using a rate-distortion objective of the form:

$$\mathcal{L}\left(\boldsymbol{\phi}, \boldsymbol{\theta}, \boldsymbol{\psi}\right) = R\left(\hat{\mathbf{y}}; \boldsymbol{\psi}\right) + \lambda D\left(\mathbf{x}, \hat{\mathbf{x}}\right),$$

Here $R(\hat{\mathbf{y}})$ is an estimate for how many bits are required to encode the tensor, and $D(\mathbf{x}, \hat{\mathbf{x}})$ measures the distortion of the reconstruction, while $\lambda > 0$ controls a tradeoff regarding reconstruction quality and compression rate. ψ is the parameters of the learned latent probability model, Negative log-probability assignment of the quantized latent symbols is utilized to estimate the rate term:

$$R(\hat{\mathbf{y}}) \approx -\log_2 p_{\hat{\mathbf{y}}}\left(\hat{\mathbf{y}}\right).$$

Given $p_{\hat{\mathbf{y}}}$ as a learned entropy model. The accuracy of the probability model is enhanced by assigning shorter entropy codes to common latent values, improving compression while maintaining the reconstruction transform.

Balle18, scale hyperprior [2] The first Balle model encodes an image into the latent tensor $\mathbf{y}$. The subsequent improvement was to add a second scale-hyperprior, as a small latent tensor $\mathbf{z}$. This $\mathbf{z}$ describes the statistical distribution of $\mathbf{y}$ allowing for $\mathbf{y}$

to be more accurately entropy-coded. Although $\mathbf{z}$ introduces additional bits, it may reduce total file size if it saves more bits while coding $\mathbf{y}$.

The scale hyperprior can be formulated using

$$\begin{aligned} \mathbf{y} &= g_a(\mathbf{x}), & \mathbf{z} &= h_a(\mathbf{y}), \\ \hat{\mathbf{y}} &= Q(\mathbf{y}), & \hat{\mathbf{z}} &= Q(\mathbf{z}), & \boldsymbol{\sigma} &= h_s(\hat{\mathbf{z}}). \end{aligned}$$

Again $\mathbf{x}$ is the input image, and the primary latent tensor is $\mathbf{y}$ while $\mathbf{z}$ is the hyper latent from the latent analysis transform h_a. Then after the quantization phase the hyper-synthesis transform h_s is used to predict the scale values $\boldsymbol{\sigma}$ using the quantized hyper latent $\hat{\mathbf{z}}$ to predict the scale values $\boldsymbol{\sigma}$ of the probability model used to entropy-code the $\hat{\mathbf{y}}$.

This means that a small predicted scale signifies a latent value expected to be concentrated in a small, predictable range. A larger scale permits a wider range of values. The cost is adjusted to the cost of the main latent using the distribution predicted from side information, and a second term is introduced representing the cost of transmitting the hyper latent. This allows for a check that the model is effective and the improved probability model reduces the cost of the primary latent more than the increased cost from the new hyper-latent.

COIN, implicit neural representation [9] COIN is a separate learning based implementation of an implicit neural representation, in contrast to the learned transform coding methods [1, 2] detailed above. One of the practical differences is that COIN-type representations represent each image as a small neural network, while Balle [2, 1] uses network architectures to *train* encoders and decoders, representing each image as a latent tensor. Conversely COIN is not representing the image as a tensor, but as the network parameters itself. An implicit representation receives a pixel coordinate, and predicts an RGB value within that coordinate, per image. The quantized network weights are thereby the representation of the compressed image.

For an image, i, this means the coordinate function can be written with:

$$f_{\boldsymbol{\theta}_i} : \mathbb{R}^2 \longrightarrow \mathbb{R}^3, \qquad \mathbf{p} \longmapsto \hat{\mathbf{I}}_i(\mathbf{p}).$$

For an image plane coordinate $\mathbf{p} \in \mathbb{R}^2$, where $f_{\boldsymbol{\theta}_i}$ is the neural network associated with weights and biases $\boldsymbol{\theta}_i$. Prompting the network at that coordinate reveals $\hat{\mathbf{I}}_i(\mathbf{p}) \in \mathbb{R}^3$, the predicted RGB value.

The encoding phase optimizes the network parameters to produce the target image.

$$\boldsymbol{\theta}_i^* = \arg\min_{\boldsymbol{\theta}_i} \frac{1}{|\mathcal{P}_i|} \sum_{\mathbf{p} \in \mathcal{P}_i} \left\| f_{\boldsymbol{\theta}_i}(\mathbf{p}) - \mathbf{I}_i(\mathbf{p}) \right\|_2^2 .$$

Where $\mathcal{P}_i$ is the full set of sampled image coordinates. Yielding the optimized parameter set $\boldsymbol{\theta}_i^*$. So each network (commonly a multi-layer perceptron, MLP [14], network) has a per image training phase, acting as the encoder.

Furthermore, fitted network parameters are then quantized.

COIN++ [10] COIN++ offers a unique extension of COIN by reducing per-image optimization and storage costs. The idea is to utilize a meta-learned coordinate network which is shared across multiple images or signals. In this way **this innovation is essentially a network initializer, very similar to our proposed innovation.** Then instead of fitting the network weights independently, a set of image-specific modulation parameters can be defined to tune the base network to the image state. The coordinate function can be shown for image i as:

$$f_{\boldsymbol{\theta},\boldsymbol{\phi}_i} : \mathbb{R}^2 \longrightarrow \mathbb{R}^3, \qquad\qquad \mathbf{p} \longmapsto \hat{\mathbf{I}}_i(\mathbf{p}).$$

The resulting modulated coordinate network $f_{\boldsymbol{\theta},\boldsymbol{\phi}_i}$ comes from its meta-learned parameters $\boldsymbol{\theta}$ and its modulation parameters $\boldsymbol{\phi}_i$.

For every new image shared parameters are fixed, while the modulation parameters are optimized with:

$$\boldsymbol{\phi}_i^* = \arg\min_{\boldsymbol{\phi}_i} \frac{1}{|\mathcal{P}_i|} \sum_{\mathbf{p} \in \mathcal{P}_i} \left\| f_{\boldsymbol{\theta},\boldsymbol{\phi}_i}(\mathbf{p}) - \mathbf{I}_i(\mathbf{p}) \right\|_2^2 .$$

As only modulation parameters $\boldsymbol{\phi}_i^*$ are optimized, while $\boldsymbol{\theta}$ stays fixed, fewer parameters are required per image than a complete network in COIN. This yields speed and efficiency upgrades in practical use cases.

This also allows for storage size to decrease, as only image-specific modulation values are quantized and entropy coded, resulting in an optimized compressed bitstream. At decode time, image-specific modulation values can simply be applied to a saved shared coordinate network.

These methods lie in two types, learned transform codecs that store image-specific latent tensors and a shared synthesis network[1, 2], and image-specific implicit coordinate networks [9, 10]. These methods offer a tradeoff, while the Balle-style methods [1, 2] achieve generally better rate-distortion results, they are much slower to decode, on the other hand, both are magnitudes slower to decode than GaussianImage [45] methods. While these methods are often those benchmarked and discussed in research, some further improvements for them have also been proposed.

For learned transform codecs, the improvements are generally in the architecture of the analysis and synthesis transforms, and sometimes in the probability models that are used for the encoding of latent tensors. Minnen et al. [25] utilize both hierarchical side information and an autoregressive context model, which allows the improvement of prediction of subsequent symbols from utilizing previously decoded latent values, but may cost more sequential coding. Cheng et al. [5] improve the flexibility of the entropy model for latent values and improve rate distortion performance using discretized Gaussian-mixture likelihoods and attention modules. ELIC from He et al. [13] is an innovation that uses uneven channel grouping paired with combined spatial-channel context models, these serve to balance compression performance and achieve better encoding and decoding speeds. Recently, Liu et al. propose LIC-TCM [22] proposed a combined architecture of convolutional and transformer components to capture local and non-local image dependencies directly within the learned entropy and transform models.

For implicit neural representations, recent work has investigated architectural changes that improve image detail retained by MLPs. SIREN [36] uses sinusoidal activation functions and a corresponding initialization scheme, enhancing the representation of fine spatial detail and signal derivatives. Then Tancick et al. [37] utilized Fourier feature mapping before application of the MLP, reducing the tendency in convolutional networks to learn the low-frequency features of the image structure before the high-frequency detail. COOL-CHIC proposed a hierarchical coordinate-based codec where the learned image function has its parameters entropy-coded, which offers a practical coordinate-based alternative to the normal MLP architectures [21].

2.3 Relevance of Gaussian Image Representations

The compression methods reviewed above offer compromises among reconstruction quality, storage rate, first time and consequent encoding cost, decoding cost, and interpretability.

The conventional codecs studied, like JPEG[19], PNG [41], and WebP [12] are highly effective for image storage, but do not offer an editable or interpretable representation. Learned transform codecs like the Balle series [1, 2] represent an image using a latent using a latent tensor which is decoded using a shared neural network. These methods offer great compression performance, but the individual latent values also do not usually correspond directly to any spatially relevant image primitives. Lastly, the COIN style [9] MLP approaches move further towards a functional representation but still store image-specific neural parameters or modulations. They require neural-network evaluation at pixel coordinates for values, and are not directly associated with explicit image geometry. Where COIN [9] distributes image information throughout its neural network weights, Gaussian representations utilize explicit primitives with centers, scales, orientations, and colour coefficients which are exposed through optimization, and retain meaning throughout different representations for different images.

Conversely the emerging fields of Gaussian image representations provides a distinct position. Akin to the implicit neural representation, they also optimize a collection of continuous parameters rather than raster samples. However, unlike such networks, the parameters also correspond to explicit spatial primitives. This means individual Gaussians can be easily modified through direct operations on the representation itself.

Other work has utilized such qualities for tasks beyond simple scene/raster reconstruction. Time-varying methods have been utilized with Gaussians for learned motion or deformation across timestamps, which enables dynamic scenes which incorporate temporal changes [42]. Pang et al. have explored animations for human representations [29], while other works explore transformations using sparse motion controls or using text-guided scene editing for local transformations [17, 39]. Furthermore, approaches deal with attaching learned semantic features to individual Gaussians, allowing support for semantic segmentation, language prompted editing, and querying [48, 30]. These works all serve to demonstrate the value of Gaussian representations not only as fast rendering elements, but also as carriers for temporal, geometric, and semantic information. **The representation is at a conjunction between compression, rendering speed, editability, and physical interpretability. Thus, Gaussian representations are alluring for their potential in downstream image tasks.**

2.4 Gaussian Splatting Techniques

The representation of visual media is an increasingly relevant field for computer vision and graphics regimes. Traditional image and video representations are typically derived from raster samples; an image is stored as a grid of sample pixels, and compression methods usually exploit spatial patterns, repetitive (frequency-domain) responses, and/or perceptual redundancy. However, recent advances in neural [24] and primitive based [20] representations have changed this regime. Instead of an image (or similar media) being treated as fixed arrays of samples, visual content can be represented as continuous functions, or a set of renderable primitives with useful properties. In the latter approach, such primitives may have parameters optimized to reproduce the visual observations.

A Neural-Radiance Field (NeRF) [24] models a scene using a continuous implicit neural function:

$$F_\Theta : (\mathbf{x}, \mathbf{d}) \mapsto (\sigma, \mathbf{c}),$$

$\mathbf{x} \in \mathbb{R}^3$ is the 3D location, $\mathbf{d}$ is the viewing direction, σ is the volume density, and $\mathbf{c}$ is the view-dependent colour. This means the scene is stored as neither a mesh or an image grid, instead as the parameters Θ of a function. Images can then be generated by querying this function along camera rays.

The downside of this is that it can be costly to render many neural evaluations for every ray. Instead 3D Gaussian Splatting [20] models a field with explicit Gaussian primitives, providing both a functional and explicit definition. A Gaussian primitive can be written as:

$$g_j(\mathbf{x}) = \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_j)^T \Sigma_j^{-1} (\mathbf{x} - \boldsymbol{\mu}_j)\right), \tag{2.1}$$

Where importantly the 3D coordinate $\boldsymbol{\mu}_j \in \mathbb{R}^3$ represents the Gaussian center and Σ_j is the associated covariance matrix. The benefit of this interpretation is that each Gaussian is an explicit primitive with position, shape, opacity, and colour. The representation is therefore interpretable, easily modifiable compared to neural/MLP representations, and renders much faster. The covariance of the Gaussian can be commonly parameterized by the scale and rotation:

$$\Sigma_j = R_j S_j S_j^T R_j^T,$$

where R_j is a rotation matrix and S_j is a diagonal scale matrix. This pliability allows Gaussians to align to surfaces, edges, or elongate to meet the needs of represented structures. Actual image rendering occurs after affine projection outlined in [20]. An image is then rendered as:

$$\hat{\mathbf{C}}(\mathbf{p}) = \sum_{j=1}^{N} T_j(\mathbf{p})\alpha_j(\mathbf{p})\mathbf{c}_j,$$

given, $\mathbf{p}$, the pixel location, $\mathbf{c}_j$ the colour of the j-th Gaussian, $\alpha_j(\mathbf{p})$ represents its opacity contribution at $\mathbf{p}$, and

$$T_j(\mathbf{p}) = \prod_{k<j} (1 - \alpha_k(\mathbf{p}))$$

is a term that defines accumulated transmittance of earlier Gaussians along the viewing direction. This formulation allows for fast rendering and CUDA acceleration [20]. A relevant benefit of Gaussian representations is their efficiency and effectiveness, they not only attain good visual quality [47, 20, 16], but only place primitives where objects are likely located in the 3D space. A very natural question is then, what if the goal is not novel view synthesis, and instead representing only a single image (and not a set). In this case, the scene depth and view-dependence are unnecessary.

The leading work for this research angle, GaussianImage [45] provides such a definition. Firstly, 2D image representation splatting, Gaussians are defined over pixel coordinates as opposed to 3D coordinates in Eq 2.1, with pixel $\mathbf{p} \in \mathbb{R}^2$,

$$g_j(\mathbf{p}) = \exp\left(-\frac{1}{2}(\mathbf{p} - \boldsymbol{\mu}_j)^T \Sigma_j^{-1} (\mathbf{p} - \boldsymbol{\mu}_j)\right), \tag{2.2}$$

with $\boldsymbol{\mu}_j \in \mathbb{R}^2$, a 2D image-plane centre. GaussianImage removes depth-ordering completely and uses a accumulated summation.

$$\hat{\mathbf{I}}(\mathbf{p}) = \sum_{j=1}^{N} \mathbf{c}_j g_j(\mathbf{p}), \tag{2.3}$$

where $\hat{\mathbf{I}}(\mathbf{p})$ is the reconstructed colour at pixel $\mathbf{p}$, N is the number of Gaussians, and $\mathbf{c}_j$ is the colour or coefficient vector of the j-th Gaussian [45]. Since depth ordering is removed, and α compositing is no longer applied, the α response is merged directly into the colour coefficient for each Gaussian, reducing complexity further. This efficiently maps

the 3D case into 2D, allowing every Gaussian to contribute a localized colour response. GaussianImage [45] also provides an implementation of attribute specific quantization, with residual vector quantization for colour coefficients. Position is quantized at 16-bit while covariance/Cholesky factorization parameters are quantized with learned asymmetric integer quantization. For covariance formulation it can be written as

$$
\begin{aligned}
q_j &= \text{round}\left(\frac{x_j - \beta}{\alpha}\right), \\
\hat{x}_j &= \alpha q_j + \beta.
\end{aligned}
$$

Where x_j is a Gaussian attribute, q_j an integer code, α and β are learned scale and offset parameters that are shared across every Gaussian. The residual vector quantization for colour can be defined as

$$
\begin{aligned}
\mathbf{r}_j^{(0)} &= \mathbf{x}_j, \\
k_j^{(m)} &= \arg\min_k \left\| \mathbf{r}_j^{(m-1)} - \mathbf{e}_k^{(m)} \right\|_2^2, \\
\mathbf{r}_j^{(m)} &= \mathbf{r}_j^{(m-1)} - \mathbf{e}_{k_j^{(m)}}^{(m)}, \\
\hat{\mathbf{x}}_j &= \sum_{m=1}^{M} \mathbf{e}_{k_j^{(m)}}^{(m)}.
\end{aligned}
$$

Where $\mathbf{x}_j$ represents the weighted colour coefficient vector for a Gaussian j, $\mathbf{e}_k^{(m)}$ represents the the k-th codeword in the m-th residual vector quantization codebook, $k_j^{(m)}$ is the selected code index, the number of residual quantization stages is given with M, and $\hat{\mathbf{x}}_j$ is the reconstructed quantized colour vector. The parameters and attributes along with the codebooks for colour are jointly refined through the ℓ_2 reconstruction loss of the image. A commitment loss encourages the colour vectors to remain close to their selected codewords. After fine-tuning, the discrete covariance values and RVQ indices are entropy coded using asymmetric numeral systems. The general Gaussian image [45] pipeline in contrast to the 3DGS [20] pipeline can be seen in Figure 2.1.

Although not competitive with established codecs such as JPEG [45] in natural images at medium-low resolution, they are competitive with implicit-neural-representations codecs like COIN [9]. Likewise, the benefit of the representation is that it is interpretable, explicit, and provides compression, where most established codecs only provide compression [19, 12]. Work has already been done using Gaussians in 3D space as a latent space for diffusion [47], as well as image space Gaussian representations [7]. Likewise, Gaussian

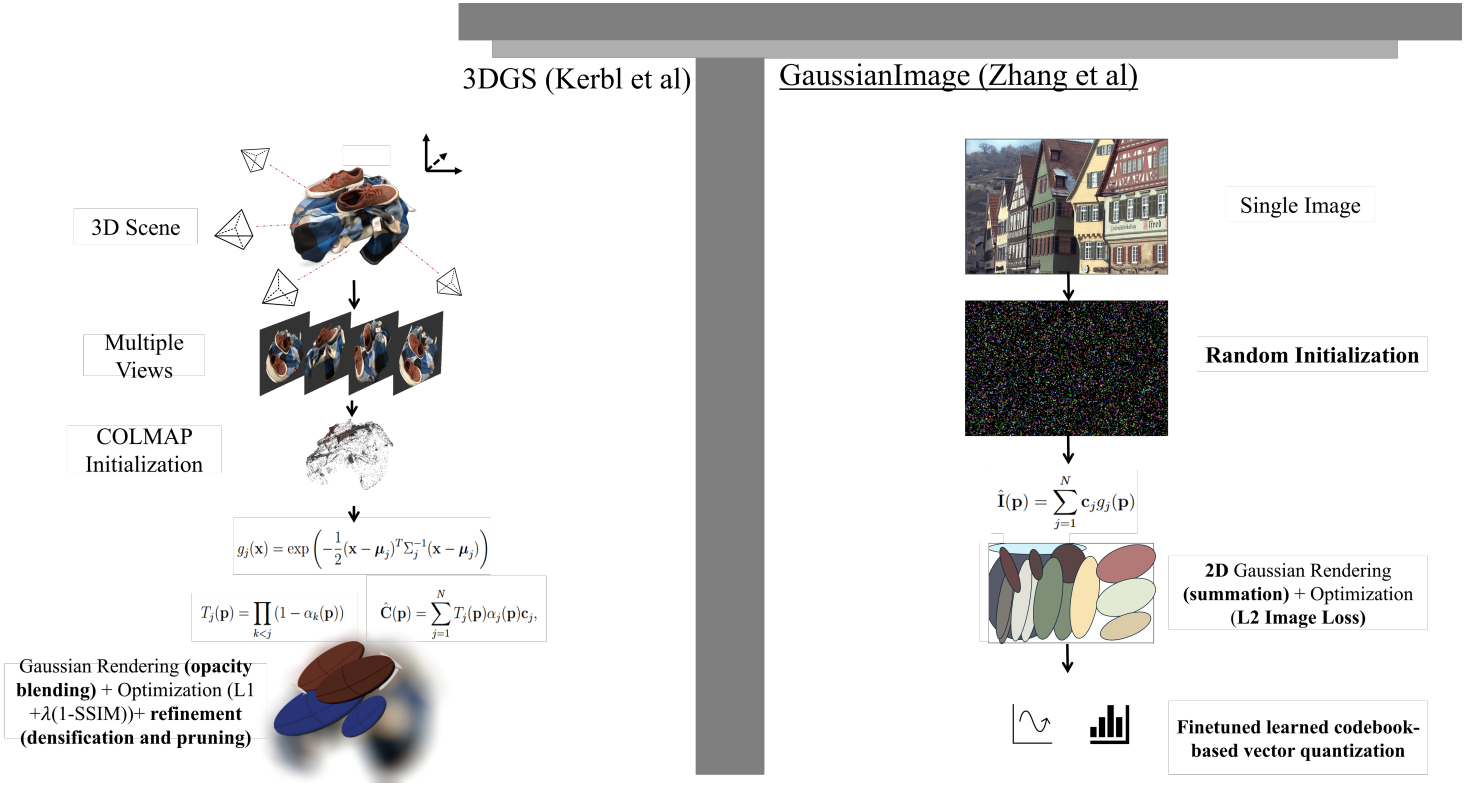


Figure 2.1: Comparison of 3DGS[20] and GaussianImage [45] pipelines, illustrating the reduction in dimension, lack of initialization and refinement robustness, and additional compression elements in GaussianImage [45].

image representations at higher resolutions codecs are much more competitive with baselines like JPEG [46], even despite not applying anything beyond simple 16-bit quantization of parameter values. This motivates the use in tasks such as super resolution [15, 4], representation of large images [49], and as an editable format [38].

Although these emerging uses are exciting, focus has shifted from representation with thousands of Gaussians, to tens of thousands and beyond. The domain has also shifted, mostly abandoning mid-low resolution natural images. However, if Gaussian images prove to be useful latent spaces, such performance issues should be improved upon. A key point in the GaussianImage formulation, is the use of random initialization [45]. As discussed in Section 2.5, initialization regimes for Gaussian optimization can produce varying results.

The observations ultimately motivate the **central question** of the thesis: can the representation of the difficult case of low-medium resolution natural images with Gaussian primitives be improved by structured initializations and refinement? The work explored in this section show that image based splatting approaches are viable for both representation and compression, but random initializations do not enforce image structure for relevance in optimal sparsity and meaningful structured latent representations.

The GS-Sketch method uses a linear formulation with Gaussian geometry fixed, allowing for Alternating Direction Method of Multipliers (ADMM) [3] to be applied to solve Gaussian coefficients. The fixed geometry stage is paired with drawing-inspired methods to produce multi-scale proposals, and a heuristic split between disk-like shapes in smooth regions and pin-like shapes in edge/high-frequency regions. The placement of geometry and its initialized parameters is heuristically retrieved from image transformations like residuals, edge, coherence, and local scale maps. This allows for initializations to contain prior information about image structure before optimization. A normalized rendering formula similar to the top-K rendering from ImageGS [46] as discussed in Section 2.6. Then the pipeline adds refinement through hole-mitigation measures in initializations and densification and pruning during training. Finally fitted Gaussian parameters are packed into image-like arrays, and compressed with PNG [41]/WebP [12] formats using an adaptive sorting contest that discretely sorts images to improve smoothness for compression. I obtain results outperforming the baseline in fitting for constrained budgets on the Kodak dataset [11], along with superior compression results at the 0.8 bits-per-pixel (bpp) range against native quantization, while the proposed codec compression yields better results for the proposed pipelines over baseline. The system also yields significant time gains for fitting to reach the same quality, and against first-time quantization considering the learned codebook regime.

2.5 Conceptual Motivations from Prior Work

The use of initializations to encode prior information before optimization in 3DGS [20] (and specifically 2DGS [16]) for practical applications of Gaussian Splatting techniques has been a recurring theme in our work completed throughout this thesis. Prior adjacent work centered around the use of 3DGS methods for BEM [6, 26]. Where, particularly, a MiDAS[31] based depth densification method was developed, to uniformly densify initializations, yielding smoother final optimizations with closed geometry[6]. These works show that a pivotal consideration for the practical applications of GS techniques is how prior domain/application information can be encoded into the model. For practical modeling, this generally is at odds with the high-frequency bias of optimization, and COLMAP[34, 33] initialization in 3D. High-frequency bias prevents low-frequency but geometrically relevant qualities like walls from being represented properly, leading to sparsity and consequent issues (particularly in proposing closed volumes) for downstream applications. In this sense, these highlight that structuring your initialization and encoding prior knowledge specific to the domain into optimization is a key element to practical applications of GS methods. **This work offers a similar innovation, targeting the weak areas of Gaussian image representation in low-medium resolution natural images [45, 46] with structured initialization and refinement.**

2.6 Gaussian Image Methods

Gaussian representations for images are an emerging field of niche research, in which this work is centered around. Figure 2.2 reflects the current upward trend of research work focused around this niche, showing the growing relevance. Some useful work relevant for understanding the positioning and relevance of the pipeline proposed in this thesis are reflected below.

GaussianImage [45]. This is the aforementioned baseline, it utilizes 2D Gaussian primitives and targets both representation and compression on the Kodak dataset [11]. The usage in constrained Gaussian counts, and for the intersection between compression in representation make this the closest baseline for the thesis goals. The targeted improvement is the use of random initializations, and the sum-based rendering formulation.

MiraGe [38]. This work provides an interesting formulation using a mirror camera (by flipping the image, and defining a camera view point looking at the back-side of

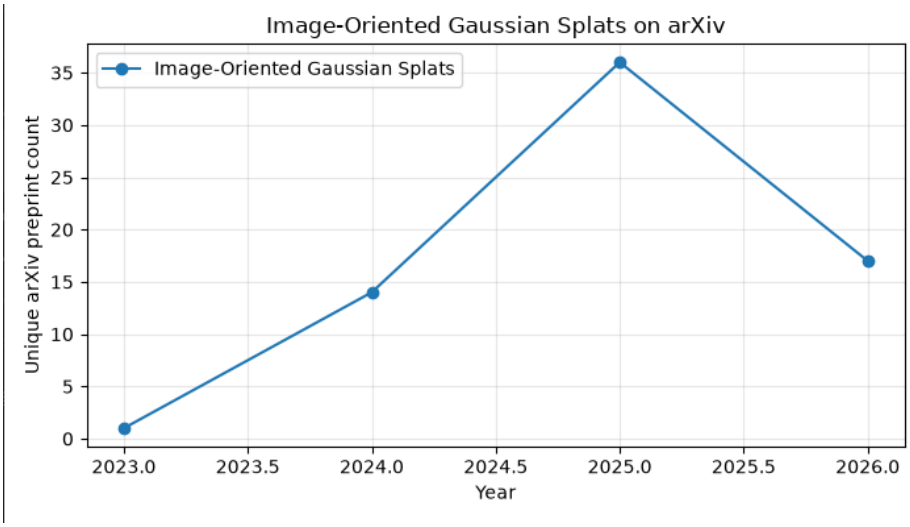


Figure 2.2: ArXiv papers related to Gaussian image representations aggregated by year.

the image). This allows for training with depth, unlike other methods [45, 46]. This formulation allows from some interesting interactions including foreground and background definition, and editable formats. They create a 'soup' of node connections that allow for editing that resembles 3D manipulation using the Gaussian representation. This work shows the viability of the representation methods for more than just compression, but for controlled and interpretable editing.

Image-GS [46]. ImageGS provides two pivotal contributions to the field, the first is the use of Top-K normalized rendering:

(2.4) Where every Gaussian is weighted by the contributions of the top-K Gaussians near the area. While our method also uses normalized rendering of a similar form, it normalizes every contributing Gaussian, and not just the top-K. They also propose adaptive Gaussian allocation, however do not fully initialize geometry details other than position. They target high resolution images, and specifically mention their intended application is not representative for low-medium resolution natural images [46]. They target towards high resolution graphics/image assets and an LOD-style representation, rather than targeting constrained natural image compression.

This approach is directly related to the initialization theme of the work. They propose a neural network that quickly generates coarse Gaussian representations, then applies the

regular training process. They apply a learned predictor from a network architecture to propose initial Gaussians. This method benchmarks for large Gaussian numbers in the tens of thousands to hundreds of thousands scale. A key distinction between this work and the thesis work is the use of heuristic image-processing guided proposal versus learned predictors, and the higher budget application.

The main contribution of this work is scaling 2D Gaussians for very large images using a coarse-to-fine representation approach. It primarily deals with high-count/large image fitting (counts about hundreds of thousands of Gausssians for ultra high-resolution images.)

Hu et al. introduce alluring application of 2D Gaussian Splatting approaches for image super resolution. This work outlines the general process to represent image information using continuous Gaussian fields achieving arbitrary-scale super-resolution. This method utilizes the training of a model using Gaussian style continuous feature fields to reconstruct high-resolution images at an arbitrary scale. This work is evidence for the Gaussian representation as both a latent space and scale-invariant super-resolution approach.

Another super resolution method from 2D Gaussian splatting on images. This method utilizes a network to predict an image-conditioned Gaussian set from a low-resolution image:

$$\mathcal{G}_\theta = \Phi_\theta \left(\mathbf{I}_{\text{LR}}, s \right),$$

which can then be rendered at super-resolution scales, using a scale-aware rasterizer:

$$\mathbf{I}_{\text{SR}} = \mathcal{R}_s \left(\mathcal{G}_\theta \right),$$

This work furthers the usefulness in Gaussian representations in both a learning and image representation space, with practical applications at high resolution ranges.

Applies GaussianImage-RS [45], and studies the latent space for image generation/denoising. The most important contribution is that diffusion can be directly applied on Gaussian parameters.

2.7 Contribution Positioning

Unlike other Gaussian Image methods [46, 15, 49, 4] the goal of this method does not address high resolution, super-resolution, or graphic asset representation. Instead, it focuses solely on low-medium resolution natural images case where other methods struggle [46]. A comparative breakdown of the scope and relative position of the work is shown in Table 2.1.

Table 2.1: Scope and targets of previous 2D Gaussian Image works, highlighting differences in target Gaussian complexity, type of images, and resolution of images.

Method	Reported regime	Main target	Relationship/Scope
GaussianImage [45]	Low-bpp Kodak/DIV2K compression; separate larger-count representation analyses such as fixed-count ablations.	Natural-image representation and compression using 2D Gaussian splatting.	Closest baseline because it studies the same natural-image Gaussian compression setting.
MiraGe [38]	Higher-count Gaussian comparisons, including 70K–100K style reconstruction/editing settings.	Editable 2D Gaussian image representation, depth-inclusive.	Although an interesting exploration on retrieving depth estimation automatically, not relevant towards low-count representation.
Image-GS [46]	Mid-count examples such as 10K–50K Gaussians at high image resolutions.	Content-adaptive graphics/image assets with random access and level-of-detail behaviour.	Related with normalized rendering formulation, and adaptive allocation, but does not instantiate Gaussian geometries, and is targeted towards high-resolution/generated content.
Instant-GI [44]	Adaptive Gaussian count; reported examples are generally in the tens-of-thousands range.	Learned fast initialization for 2D Gaussian image representation.	A relevant initialization, but benchmarked and tested at high Gaussian counts. Utilizes a neural predictor rather than an explicit sketch-inspired proposal.
LIG [49]	Hundreds of thousands to millions of Gaussians in large-image settings.	High-quality large-image representation with levels of 2D Gaussians.	A high resolution targeted regime, that is not well related to the constrained Kodak budgets studied in this work.
GS-Sketch	Approximately 2.1K–13.8K Gaussians in the tested Kodak regimes.	Low-count natural-image Gaussian representation and compression.	Explicitly targets constrained budgets and non-neural structure-guided initialization on natural images.

Chapter 3

GS-Sketch: a Structured Image Representation Method Using Gaussian Splats

This chapter outlines the theoretical formulations, methods, implementation details, experiments, and results of GS-Sketch the proposed structured initialization and refinement pipeline for Gaussian image representation. Discussion is provided for the analysis and critical interpretation of results, which are elaborated further in Sections 4.2, 4.2.1, 4.2.2.

3.1 Methods

This section outlines the problem formulation, design decisions, and overall implementation of the methods of representing images with Gaussians.

3.1.1 Problem Reformulation

In contrast to fully unconstrained joint optimization proposed in GaussianImage [45], the problem can be reformulated into a convex problem which is linear in the Gaussian colour coefficients. By fixing the Gaussian geometry properties using intelligent heuristic proposal, the problem decouples into a constrained convex optimization sub-problem solved using ADMM for the Gaussian coefficients. Each proposed Gaussian 'atom' j is surmised by an anisotropic 2D Gaussian of the form depicted in Eq (2.2), with the primary inputs

$\mathbf{p} = [u \quad v]^T$ as pixel location, the center of the j-th Gaussian as $\boldsymbol{\mu}_j = [\mu_{x,j} \quad \mu_{y,j}]^T$, and the covariance matrix represented with $\Sigma_j \in \mathbb{R}^{2\times 2}$. The complexity can be reduced by coupling the opacity, α and colour coefficients, c, as suggested by GaussianImage [45], which can be written in our formulation as $\mathbf{x}{=}\alpha_i c^T$. The Gaussian geometry matrix can then be multiplied by x yielding the rendering formulation and retrieving the coloured and modulated primitives for rendering.

The covariance can be written using the major axis $\sigma_{\text{maj},j}$, minor axis $\sigma_{\text{min},j}$, and orientation θ_j,

$$\Sigma_j = R(\theta_j) \begin{bmatrix} \sigma_{\text{maj},j}^2 & 0 \\ 0 & \sigma_{\text{min},j}^2 \end{bmatrix} R(\theta_j)^T$$
$$R(\theta_j) = \begin{bmatrix} \cos\theta_j & -\sin\theta_j \\ \sin\theta_j & \cos\theta_j \end{bmatrix}.$$

where $\sigma_{\text{maj},j}$ and $\sigma_{\text{min},j}$ are the scales, and θ_j is the in-plane rotation angle.

For image rendering, a normalized blending formula is proposed to utilize more interactions between Gaussians and map more complex features, while still mapping sparse regions of varying Gaussian density. The raw Gaussian matrix can be defined as $G \in \mathbb{R}^{N\times n}$ by

$$G_{ij} = g_j(\mathbf{p}_i),$$

and the row-normalized blending operator

$$B = \text{diag}\left(\frac{1}{\max(G\mathbf{1}, \varepsilon)}\right) G, \tag{3.1}$$

Here, $(\text{diag}(\cdot))$ is a function to construct a diagonal matrix from the vector argument. The maximum is applied element-wise, and $(\varepsilon > 0)$ is a safeguard that prevents division by zero in areas of low coverage or numerically small Gaussian response. Small values such as (10^{-6}) to (10^{-10}) are typically sufficient for this numerical safeguard depending on the scale of the problem; the implementation used throughout this work sets $(\varepsilon = 10^{-8})$.

The rendering model can be written in linear form as

$$\hat{\mathbf{y}} = B\mathbf{x}. \tag{3.2}$$

where the row sums of G are denoted by the vector $G\mathbf{1}$. In this way, all the nonlinearity of the equation can be absorbed by the blending operator $B \in \mathbb{R}^{N\times n}$. $\mathbf{x} \in \mathbb{R}^n$ is the Gaussian

coefficient vector. This leads to $\hat{\mathbf{y}} \in \mathbb{R}^N$ which is the rendered image. **Note that G and B are never built fully, they are built as sparse matrices, if a Kodak-sized image were to have 5-10 thousand Gaussians, this matrix would have billions of entries, and not be suitable. This is why in Section 4.2.2, it is suggested to replace this formulation for higher resolution applications for the faster CUDA path.**

To recap, G is built, containing all Gaussian geometry information (and excluding the opacity and colour coefficients), then it is row-normalized, and renderered into an image using the linear combinination with the opacity/colour coefficients. Note that the heuristic process targets grayscale structure and colour coefficients which are merged with opacity in a separate colour addition phase (after fully realizing fine grayscale structure). Sometimes coefficients in these cases are referred to as 'Gaussian coefficients' in the text, to accomodate the reference to either grayscale or colour coefficients. The rendering formula can be equivalently written per pixel, showing the similarity to the one presented in GaussianImage [45] and ImageGS [46] (who also propose a similar normalized rendering regime, but with Top-K normalization). For an image pixel y_i, with ϵ as a small coefficient to mitigate small divisors:

$$\hat{y}_i = \frac{\sum_{j=1}^{n} x_j \, g_j(\mathbf{p}_i)}{\sum_{j=1}^{n} g_j(\mathbf{p}_i) + \varepsilon}, \qquad i = 1, \ldots, N,$$

The main difference between this formula is the normalization denominator, otherwise it is the same rendering formula as [45].

After the definition of the rendering formula, the equation for the optimization of Gaussians can be outlined. Instead of simply taking ℓ_1 or ℓ_2 error, the ADMM formulation can be utilized to rapidly prototype several image transformation errors into a discrete formulation. For the current pipeline the image features from edge gradients (as the transformations D_x and D_y), and second order image responses from a discrete Laplacian operator L:

$$\begin{aligned} \min_{\mathbf{x}} \; \frac{\lambda_{\text{pix}}}{2} \|\mathbf{w} \odot (B\mathbf{x} - \mathbf{y})\|_2^2 + \frac{\lambda_{\text{grad}}}{2} \Big(\|D_x B\mathbf{x} - D_x \mathbf{y}\|_2^2 \\ + \|D_y B\mathbf{x} - D_y \mathbf{y}\|_2^2 \Big) + \frac{\lambda_{\text{lap}}}{2} \|LB\mathbf{x} - L\mathbf{y}\|_2^2 \\ + \frac{\eta}{2} \|\mathbf{x}_{\mathcal{P}} - \mathbf{x}_{\mathcal{P}}^{\text{prev}}\|_2^2. \end{aligned}$$

The symbol ($\odot$) denotes the Hadamard, or element-wise, product. In the above representation x is the Gaussian coefficient vector, B represents the normalized blending oeprator,

the discrete Laplacian operator is represented by L, the horizontal and edge operators are given by D_x and D_y (where they are simple first order finite differences). Furthermore, a prior term is added for Gaussians from previous completed ADMM stages (in the coarser phases) to mitigate large structural change, Gaussians belonging to $\mathcal{P}$. Lastly the coefficient w represents pixel-weighting derived from the Laplacian and Sobel edge operators, to weight the residual towards edge features:

$$w = vec(1 + \lambda_{str}[0.6\text{norm01}(M_{grad}) + 0.4\text{norm01}(M_{lap})]).$$

A simple function, Norm01, is made to normalize the maps from 0-1. It maps the minimum value to 0, the maximum value to 1, and linearly scales values between them. M_{grad} is resolved using Sobel edge operators, while M_{lap} comes from the Laplacian operator. Then w becomes a pixel-weighting map that focuses on texture and detail. **After B is fixed, the problem is convex, allowing us to use optimization rather than learning techniques like GaussianImage [45].** The stage-dependent weights used were

$$\begin{aligned}
(\lambda_{\text{pix}}, \lambda_{\text{grad}}, \lambda_{\text{lap}}, \eta)_{\text{large}} &= (1.00,\ 0.08,\ 0.02,\ 0),\\
(\lambda_{\text{pix}}, \lambda_{\text{grad}}, \lambda_{\text{lap}}, \eta)_{\text{medium}} &= (1.00,\ 0.15,\ 0.04,\ 2 \times 10^{-4}),\\
(\lambda_{\text{pix}}, \lambda_{\text{grad}}, \lambda_{\text{lap}}, \eta)_{\text{small}} &= (1.00,\ 0.22,\ 0.06,\ 4 \times 10^{-4}).
\end{aligned}$$

Using the formulation in Eq (3.3), the substitution constraint procedure can be applied directly to create an ADMM formulation:

$$\begin{aligned}
\min_{\mathbf{x},\mathbf{z},\mathbf{g}_x,\mathbf{g}_y,\boldsymbol{\ell},\mathbf{v}} \quad & \frac{\lambda_{\text{pix}}}{2}\|\mathbf{w} \odot (\mathbf{z} - \mathbf{y})\|_2^2 + \\
& \frac{\lambda_{\text{grad}}}{2}\left(\|\mathbf{g}_x - D_x\mathbf{y}\|_2^2 + \|\mathbf{g}_y - D_y\mathbf{y}\|_2^2\right) \\
& + \frac{\lambda_{\text{lap}}}{2}\|\boldsymbol{\ell} - L\mathbf{y}\|_2^2 + \frac{\eta}{2}\|\mathbf{x}_{\mathcal{P}} - \mathbf{x}_{\mathcal{P}}^{\text{prev}}\|_2^2
\end{aligned}$$

Subject to:

$$\begin{aligned}
\mathbf{z} &= B\mathbf{x},\\
\mathbf{g}_x &= D_x B\mathbf{x},\\
\mathbf{g}_y &= D_y B\mathbf{x},\\
\boldsymbol{\ell} &= LB\mathbf{x},\\
\mathbf{v} &= \mathbf{x}.
\end{aligned}$$

The augmented Lagrangian can be taken in scaled form:

$$\begin{aligned}\mathcal{L}_\rho = & \frac{\lambda_{\text{pix}}}{2}\|\mathbf{w}\odot(\mathbf{z}-\mathbf{y})\|_2^2 \\ & + \frac{\lambda_{\text{grad}}}{2}\left(\|\mathbf{g}_x - D_x\mathbf{y}\|_2^2 + \|\mathbf{g}_y - D_y\mathbf{y}\|_2^2\right) \\ & + \frac{\lambda_{\text{lap}}}{2}\|\boldsymbol{\ell} - L\mathbf{y}\|_2^2 + \frac{\eta}{2}\|\mathbf{x}_{\mathcal{P}} - \mathbf{x}_{\mathcal{P}}^{\text{prev}}\|_2^2 \\ & + \frac{\rho_z}{2}\|B\mathbf{x} - \mathbf{z} + \mathbf{u}_z\|_2^2 \\ & + \frac{\rho_g}{2}\|D_xB\mathbf{x} - \mathbf{g}_x + \mathbf{u}_{gx}\|_2^2 \\ & + \frac{\rho_g}{2}\|D_yB\mathbf{x} - \mathbf{g}_y + \mathbf{u}_{gy}\|_2^2 \\ & + \frac{\rho_\ell}{2}\|LB\mathbf{x} - \boldsymbol{\ell} + \mathbf{u}_\ell\|_2^2 + \frac{\rho_v}{2}\|\mathbf{x} - \mathbf{v} + \mathbf{u}_v\|_2^2,\end{aligned}$$

Here $\rho_g, \rho_z, \rho_l, \rho_v > 0$, and the penalty parameters, $u_z, u_{gx}, u_{gy}, u_l, u_v$, are the scaled dual variables. Where $\rho_z, \rho_g, \rho_l, \rho_v$ are greater than zero, and the ADMM penalty parameters, and $u_z, u_{gx}, u_{gy}, u_l, u_v$ are the scaled dual variables. Using the formulation in Eq (3.3), the updates can be directly retrieved for the dual and primal variables. First, substitute A:

$$\begin{aligned}A = & \rho_z B^TB + \rho_g B^T D_x^T D_x B \\ & + \rho_g B^T D_y^T D_y B + \rho_\ell B^T L^T L B \\ & + \rho_v I + \eta P^T P,\end{aligned}$$

Then substitute b^K

$$\begin{aligned}\mathbf{b}^k = & \rho_z B^T(\mathbf{z}^k - \mathbf{u}_z^k) + \rho_g B^T D_x^T(\mathbf{g}_x^k - \mathbf{u}_{gx}^k) \\ & + \rho_g B^T D_y^T(\mathbf{g}_y^k - \mathbf{u}_{gy}^k) + \rho_\ell B^T L^T(\boldsymbol{\ell}^k - \mathbf{u}_\ell^k) \\ & + \rho_v(\mathbf{v}^k - \mathbf{u}_v^k) + \eta P^T \mathbf{x}_{\mathcal{P}}^{\text{prev}}.\end{aligned} \tag{3.3}$$

Resolving the x update:

$$\mathbf{x}^{k+1} = A^{-1}\mathbf{b}^k,$$

Which is solved with Cholesky factorization when possible, otherwise with the linear solver in MATLAB (the inverse computation is avoided).

Then the substitution variable updates, for z where division is taken elementwise:

$$\mathbf{z}^{k+1} = \frac{\lambda_{\text{pix}}(\mathbf{w}\odot\mathbf{w})\odot\mathbf{y} + \rho_z(B\mathbf{x}^{k+1} + \mathbf{u}_z^k)}{\lambda_{\text{pix}}(\mathbf{w}\odot\mathbf{w}) + \rho_z},$$

For g_x and g_y:

$$\mathbf{g}_x^{k+1} = \frac{\lambda_{\text{grad}} D_x \mathbf{y} + \rho_g (D_x B \mathbf{x}^{k+1} + \mathbf{u}_{gx}^k)}{\lambda_{\text{grad}} + \rho_g},$$
$$\mathbf{g}_y^{k+1} = \frac{\lambda_{\text{grad}} D_y \mathbf{y} + \rho_g (D_y B \mathbf{x}^{k+1} + \mathbf{u}_{gy}^k)}{\lambda_{\text{grad}} + \rho_g},$$

Then for ℓ

$$\boldsymbol{\ell}^{k+1} = \frac{\lambda_{\text{lap}} L \mathbf{y} + \rho_\ell (LB \mathbf{x}^{k+1} + \mathbf{u}_\ell^k)}{\lambda_{\text{lap}} + \rho_\ell},$$

and for v:

$$\mathbf{v}^{k+1} = \mathbf{x}^{k+1} + \mathbf{u}_v^k,$$

The above constraint is retained due to allowing for easy implementation of non-negativity enforcement, when non-negativity constraints are desired the v-update is instead projected onto the non-negative orthant. When disabled it acts as a damping factor on the updates of x. Then the dual variable updates resolve to

$$\begin{aligned}
\mathbf{u}_z^{k+1} &= \mathbf{u}_z^k + B\mathbf{x}^{k+1} - \mathbf{z}^{k+1}, \\
\mathbf{u}_{gx}^{k+1} &= \mathbf{u}_{gx}^k + D_x B\mathbf{x}^{k+1} - \mathbf{g}_x^{k+1}, \\
\mathbf{u}_{gy}^{k+1} &= \mathbf{u}_{gy}^k + D_y B\mathbf{x}^{k+1} - \mathbf{g}_y^{k+1}, \\
\mathbf{u}_\ell^{k+1} &= \mathbf{u}_\ell^k + LB\mathbf{x}^{k+1} - \boldsymbol{\ell}^{k+1}, \\
\mathbf{u}_v^{k+1} &= \mathbf{u}_v^k + \mathbf{x}^{k+1} - \mathbf{v}^{k+1}.
\end{aligned}$$

At times, coefficients would yield slightly brighter or darker images than the original, so a small corrective gain implementation is added, with a scalar gain:

$$g = \frac{(Bx)^\top y}{(Bx)^\top (Bx) + \varepsilon},$$

Here, (ε) is another numerical safeguard against division by zero, for sparse areas of the image. The implementation uses MATLAB's double-precision machine epsilon here, ($\varepsilon \approx 2.22 \times 10^{-16}$). Then clamping and rescaling with g_{max} to g_{min}, mitigating any changes over 5-10% gain.

$$\begin{aligned}
&g \leftarrow \min\big(g_{\max}, \max(g_{\min}, g)\big), \\
&x \leftarrow g\, x.
\end{aligned}$$

The decoupling of geometry and coefficients, along with fixing geometry with intelligent heuristic proposal allows for an ADMM approach to be applied for the fast optimization of Gaussian coefficients. First the structure is optimized purely in grayscale, with the coarse to fine split as detailed. Then the same formulations are used for each separate colour channel. This formulation allows for multiple fidelity terms for relevant images using different transformation domains, that are efficiently represented by transformations in the resolved variable updates. The ADMM auxiliary variables are all derived in a closed form system of updates, allowing the bypass of machine-learning methods using a convex constrained optimization sub-problem.

3.1.2 Region and Geometry Proposal

Gaussian region proposal is inspired by the common drawing processes used by artists. The process is segregated into a coarse to fine approach of large (level identification), medium (initial shading), and fine (fine details and iteration). The proposal process then attempts to sketch images with Gaussian candidates. Within each separate large, medium, and fine stage are proposal and target images, one is used to create proposals and the other (sometimes the same image) is used as the target image for the operator in the ADMM phase. Each stage utilizes 2 passes to improve the fitting accuracy. The first pass in both the large and medium stages utilizes the proposal images (blurred + quantized for large, blurred for medium), while the second pass is the regular lower resolution image. For the fine stage the original image is used for both passes. The large medium and fine stage are completed in the listed order. The residuals and proposal related image transformations are recalculated after every phase (twice within a large medium or fine branch). First a large quantized version of the image is taken after resizing the original, with subscript L denoting the large image/phase, using $\mathcal{R}$ as a function to resize images with bicubic interpolation by a scale factor (in subscript):

$$Y_L = \mathcal{R}_{1/8}(Y),$$

which can be blurred using a Gaussian stage blur G_{σ_L}, which is a two dimensional Gaussian smoothing kernel with standard deviation σ with $(*)$ denoting two-dimensional convolution.

$$\widetilde{Y}_L = G_{\sigma_L} * Y_L,$$

then quantizing, using $(\mathcal{Q}_{\boldsymbol{\tau}_L}(\cdot))$, which is a intensity quantization function with some threshold $(\boldsymbol{\tau}_L)$, and norm01 linearly scales values between 0-1:

$$T^{(L)}_{\text{prop}} = \text{norm01}\left(\mathcal{Q}_{\boldsymbol{\tau}_L}\left(\widetilde{Y}_L\right)\right),$$

with the large proposal image denoted as $T^{(L)}_{\text{prop}}$, the first large-stage fitting target as $T^{(L)}_{\text{fit},1}$, and the second large-stage fitting target as $T^{(L)}_{\text{fit},2}$. A downsampling factor of 4 is used for the medium stage, and

$$Y_M = \mathcal{R}_{1/4}(Y),$$

then the blurred medium-stage image is

$$\widetilde{Y}_M = G_{\sigma_M} * Y_M.$$

Then the proposal and fitting references for the medium stage are

$$\begin{aligned} T^{(M)}_{\text{prop}} &= \widetilde{Y}_M, \\ T^{(M)}_{\text{fit},1} &= \widetilde{Y}_M, \\ T^{(M)}_{\text{fit},2} &= Y_M. \end{aligned}$$

With the proposal and first and second fitting targets as $T^{(M)}_{\text{prop}}$, $T^{(M)}_{\text{fit},1}, T^{(M)}_{\text{fit},2}$ respectively. In the fine stage, the original image is used in each pass.

$$\begin{aligned} T^{(S)}_{\text{prop}} &= Y, \\ T^{(S)}_{\text{fit},1} &= Y, \\ T^{(S)}_{\text{fit},2} &= Y. \end{aligned}$$

where $T^{(S)}_{\text{prop}}$ are the fine-stage proposal image and $T^{(S)}_{\text{fit},1}, T^{(S)}_{\text{fit},2}$ denote the fine-stage fitting targets.

An important geometric distinction is made at the coarse stages, by separating blurred ellipsoids in disk and pin families. While disks are circular/loosely deformed Gaussians placed in low-frequency/non-edge regions, pins are placed in high-frequency/edge regions. This heuristic split is visualized in Figure 3.1. Before the final stage, the distinction is generally enforced using spatial masks of the edge response, which are thresholded before their use, where the disk mask is the 'not' of the pin mask. The scores for proposal for disks and pins are computed only within their respective regions.

The reconstruction is rendered and taken against the original target image for a residual map at each stage. Proposal maps can then be formed using the edge images, residual map, and the responses in the Laplacian domain. This is also accomplished by setting

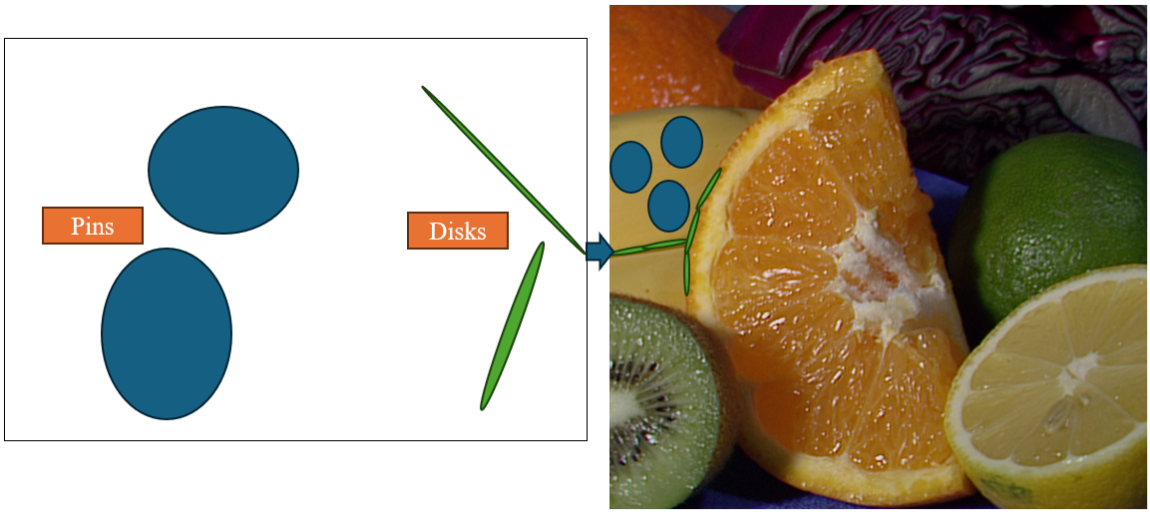


Figure 3.1: Disk and pin geometry splitting for LF (disks) and HF (pin) features

candidate scales which are heuristically proposed for each stage (see compute_geom_maps in Appendix A.3.1). This yields the targeted optimization of detail and structure features of the images.

A residual map is formed as

$$R^{(s)} = \mathrm{norm01}\left(\left|T^{(s)} - \hat{Y}^{(s)}\right|\right),$$

Here $R^{(s)}$ is the normalized residual map at a stage s, and $T^{(s)}$ is the target image at the given stage, while the current rendered reconstruction is $\hat{Y}^{(s)}$.

Proposal scores for large and medium passes are formed with separate disk-like and pin-like candidates. Firstly, a blob-likeness map is generated at each candidate scale with:

$$B^{(k)} = \mathrm{norm01}\left(\mathrm{norm01}\left(L^{(k)}\right) \odot \left(1 - C^{(k)}\right)^{1.08}\right), \tag{3.4}$$

where $L^{(k)}$ is the Laplacian response and $C^{(k)}$ the coherence map with a scale k. Placing blobs in strong second-order response regions, where there is also low coherence. Then a

separate map for edge-likeness for a given scale k is derived with

$$E^{(k)} = \mathrm{norm01}\big(\mathrm{norm01}\big(M^{(k)}\big) \odot \big(0.34 + 0.66\, C^{(k)}\big) + 0.22\ \mathrm{norm01}\big(L^{(k)}\big)\big)\,, \tag{3.5}$$

given $M^{(k)}$, which is derived using the Sobel operator as a gradient magnitude map, and the coherence map is denoted by $C^{(k)}$. Then the final disk proposal score can be written as:

$$\begin{aligned} S_{\mathrm{disk}} = \mathrm{norm01}[(R \odot [0.44(1-E) + 0.22 B_{\mathrm{blob}} \\ +0.18(1-C) + 0.16 E_{\mathrm{soft}}) \odot M_{\mathrm{disk}}], \end{aligned} \tag{3.6}$$

The score map S_{disk} is formed using the current residual, R, the edge-likeness map E, the blob-likeness map B_{blob}, the local coherence map C, the softened edge/ Laplacian response E_{soft}, and M_{disk} is used to restrict the score to disk-like regions. In the large stage, a further reweighting is added to the disk score:

$$S^{(L)}_{\mathrm{disk}} = S_{\mathrm{disk}} \odot (0.35 + 0.65 M_{\mathrm{large}})\,,$$

Given M_{large}, a mask of large quantized regions. The overall coverage of the image is promoted by this addition.

Thereafter, a pin proposal score can be defined with

$$\begin{aligned} S_{\mathrm{pin}} = \mathrm{norm01}(R \odot [0.52E + 0.28 E_{\mathrm{soft}} + 0.20C]) \\ \odot M_{\mathrm{pin}}, \end{aligned} \tag{3.7}$$

With the proposal map S_{pin} being comprised of the residual R, edge-likeness E, softened edge response E_{soft}, the local coherence map C, and the map to restrict scores to pin-like regions M_{pin}. A deduplication check for proposals is used:

$$\|p_i - p_j\|_2 \geq d_{\mathrm{min}}, \qquad i \neq j,$$

For proposed center locations p_j and p_i and minimum allowed spacing d_{min}. This promotes overall coverage, while reducing the complexity and over-tuning in high-frequency regions of proposals. For the finer stage, the allowed distance is made to be sub-pixel. As the proposals are done by tile for speed, the proposal budget is also tied proportionally to the score mass that is contained in each tile according to:

$$\begin{aligned} m_t &= \sum_{p \in \Omega_t} S(p), \\ n_t &\approx N_{\mathrm{tot}} \frac{m_t}{\sum_k m_k}, \end{aligned}$$

given the total proposal mass m_t, the pixel support of the tile Ω_t, the score map S, and the number of total proposals for tile t defined by n_t. Lastly N_{tot} is the total budget for the stage.

A unified proposal score is used for the small stage for disk candidates:

$$S_{\text{small,disk}} = \text{norm01}\Big(0.06 + 0.36R + 0.28E \\ +0.22E_{\text{soft}} + 0.08B_{\text{blob}}\Big), \tag{3.8}$$

where $S_{\text{small,disk}}$ denotes the small-stage score for unified ellipse proposals, R denotes the current residual map, E denotes the edge-likeness map, E_{soft} denotes the softened Laplacian edge response, and B_{blob} denotes the blob-likeness map. The fine-stage pin score is defined separately as

$$S_{\text{small,pin}} = \text{norm01}(R \odot [0.58E + 0.24E_{\text{soft}} + 0.18C]) \\ \odot 1\{E > \tau_{\text{pin}}^{(S)}\}, \tag{3.9}$$

where $S_{\text{small,pin}}$ denotes the fine pin proposal score, C denotes the coherence map, $1\{\cdot\}$ is an indicator mask for the relevant region, and $\tau_{\text{pin}}^{(S)}$ is the fine-stage pin threshold.

The fine stage is analogous to the process of an artist mapping remaining features and highlights iteratively to the target. The first pass of the fine stage is residual guided by an external fine-detail mismatch map from pixel, gradient, and different between the Laplacian responses of the medium reconstruction and the desired target:

$$\begin{aligned} R_{\text{fine}}^{\text{ext}} = 0.52\ &\text{norm01}\Big(|Y - \hat{Y}_{M\to F}|\Big) \\ &+ 0.24\ \text{norm01}|G_x(Y) - G_x(\hat{Y}_{M\to F})| \\ &+ |G_y(Y) - G_y(\hat{Y}_{M\to F})| \\ &+ 0.24\ \text{norm01}\Big(|L(Y) - L(\hat{Y}_{M\to F})|\Big), \end{aligned}$$

The conversion of the medium-stage fit to the fine stage scale is represented with $\hat{Y}_{M\to F}$, G_x and G_y are the Sobel derivative operators, while L is the Laplacian operator. The first pass of the fine stage, external residuals are blended with the usual stage residual, mitigating unresolved high-frequency information. Subsequent passes are then recalculated using the original image and the current stage fit.

This multiscale approach enables targeting intelligible image features at different scales, mimicking conventions and processes in human drawing. This allows for coverage to be addressed partially, and maintains guidance from residual and fine-detail through multiple passes of the small phase. Figure 3.2 illustrates this process on 'fruits.bmp'.

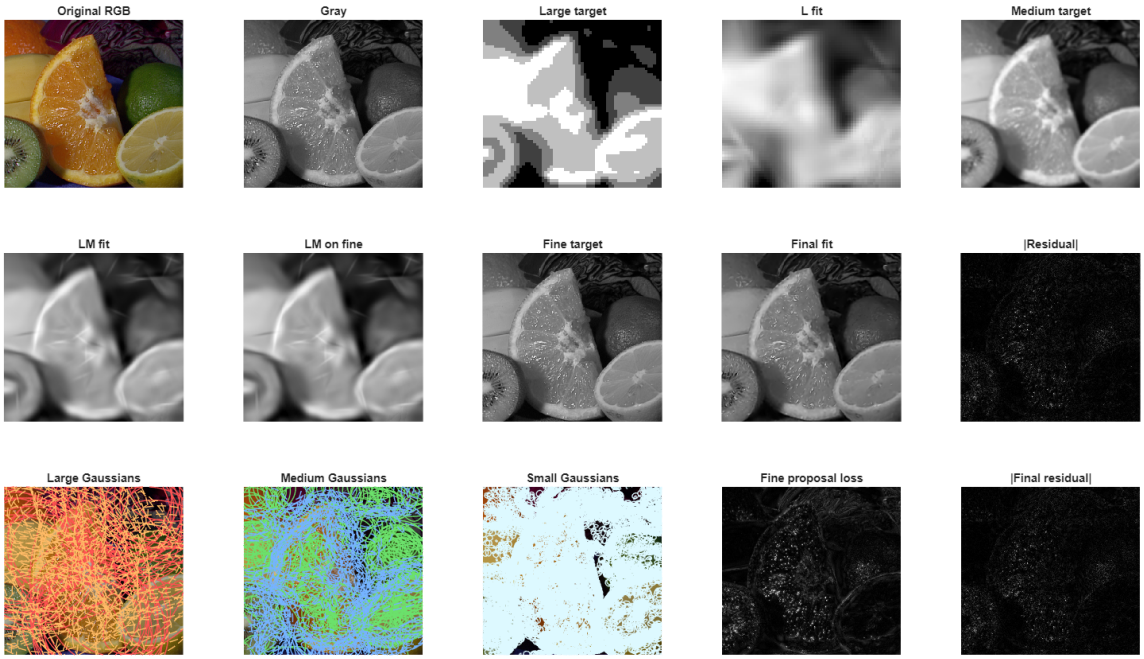


Figure 3.2: Visual flow of hot initialization process, showing fine scale, medium scale, and large scale contributions. Note that targets for first pass only are shown.

Geometry proposal

As aforementioned, the heuristic geometry proposal is accomplished through image convolutions, transformations, and metrics. After centers have been proposed, Gaussian geometry is assigned from multiscale geometric maps. Sets of candidate scales are derived for each stage. For the large stage,

$$\Sigma_{\mathrm{L}} = \{1.0,\ 1.4,\ 1.9,\ 2.5,\ 3.3,\ 4.4,\ 5.8\},$$

for the medium stage,

$$\Sigma_{\mathrm{M}} = \{0.8,\ 1.0,\ 1.3,\ 1.7,\ 2.2,\ 2.9,\ 3.8\},$$

and for the fine stage,

$$\Sigma_{\mathrm{S}} = \{0.75,\ 0.92,\ 1.12,\ 1.40,\ 1.78,\ 2.20,\ 2.75\}.$$

After definition of scale sets for each stage, a structural tensor can be derived using the Sobel derivatives and each smoothing scale σ_k, using the eigen structure to define local coherence in the given direction.

$$J^{(k)} = \begin{bmatrix} J_{11}^{(k)} & J_{12}^{(k)} \\ J_{12}^{(k)} & J_{22}^{(k)} \end{bmatrix},$$

With $J^{(k)}$ as the structure tensor for a scale σ_k. Denoting the eigen values as $\lambda_1^{(k)} \geq \lambda_2^{(k)}$, the coherence is then

$$C^{(k)} = \frac{\lambda_1^{(k)} - \lambda_2^{(k)}}{\lambda_1^{(k)} + \lambda_2^{(k)} + \varepsilon},$$

where $C^{(k)}$ denotes the coherence map at scale σ_k. The coherence attributes can then be used for the dominant local orientation as

$$\Theta^{(k)} = \frac{1}{2}\,\mathrm{atan2}\Big(2J_{12}^{(k)}, J_{11}^{(k)} - J_{22}^{(k)}\Big) + \frac{\pi}{2}, \tag{3.10}$$

Yielding the local orientation map $\Theta^{(k)}$. The winning scale is selected from the set according to the strongest edge-likeness response:

$$k^*(p) = \arg\max_k E^{(k)}(p),$$

given a edge-likeness response, $E^{(k)}$, at a scale σ_k, the selected scale index at a pixel is given with $k^*(p)$ for pixel p. Then the geometry maps for proposal can be formulated as:

$$\begin{aligned} \theta(p) &= \Theta^{(k^*(p))}(p), \\ C(p) &= C^{(k^*(p))}(p), \\ \sigma(p) &= \sigma_{k^*(p)}. \end{aligned}$$

Given the local orientation, $\theta(p)$, the coherence value that was selected $C(p)$, and the base scale $\sigma(p)$ that was selected. Then for a proposed center location p_j, the orientation can be taken directly from the map.

$$\theta_j = \theta(p_j),$$

With θ_j as the in-plane rotation of the j-th primitive. Likewise, the native base scale is initialized as

$$\sigma_j^{\text{base}} = \max(\sigma(p_j), \sigma_{\min}),$$

given $\sigma_{\min}$ as the lower bound on scale for the stage. For the proposal of large and medium disks, the aspect ratio is driven by the areas of low edge strength, with the flipped edge response:

$$\begin{aligned} t_j^{\text{disk}} &= 1 - E(p_j), \\ a_j^{\text{disk}} &= \text{quantize}\big(1 + t_j^{\text{disk}}(a_{\max}^{\text{disk}} - 1).\big). \end{aligned}$$

Given the disk aspect ratio a_j^{disk} and the maximum allowable elongation for smooth regions $a_{\max}^{\text{disk}}$. The minor and major scales are then:

$$\begin{aligned} \sigma_{\min,j} &= \max\big(\sigma_{\min}, g_{\text{disk}}\sigma_j^{\text{base}}\big), \\ \sigma_{\text{maj},j} &= \min\big(a_j^{\text{disk}}\sigma_{\min,j}, \sigma_{\max}\big), \end{aligned}$$

with the disk scale gain g_{disk} and the stage-dependent upper bound $\sigma_{\max}$. Elongation in large and medium pin proposals is controlled directly by coherence:

$$\begin{aligned} t_j^{\text{pin}} &= \text{clip}\left(\frac{C(p_j) - \tau_{\text{fam}}}{1 - \tau_{\text{fam}} + \varepsilon}, 0, 1\right), \\ a_j^{\text{pin}} &= \text{quantize}\Big(a_{\min}^{\text{pin}} + t_j^{\text{pin}}\left(a_{\max}^{\text{pin}} - a_{\min}^{\text{pin}}\right)\Big), \end{aligned} \tag{3.11}$$

with the pin-family coherence threshold set with τ_{fam} and the minimum and maximum aspect ratios for pins defined with $a_{\min}^{\text{pin}}, a_{\max}^{\text{pin}}$. The scales for pins can be retrieved with

$$\begin{aligned} \sigma_{\min,j} &= \max\big(\sigma_{\min}, g_{\text{pin}}\sigma_j^{\text{base}}\big), \\ \sigma_{\text{maj},j} &= \min\Big(a_j^{\text{pin}}\sigma_{\min,j}, \sigma_{\max}\Big), \end{aligned}$$

with a pin scale gain g_{pin}. The edge strength and coherence are both used for small unified ellipses:

$$\begin{aligned} t_j^{\text{small}} &= 0.55\, E(p_j) + 0.45\, C(p_j), \\ a_j^{\text{small}} &= 1 + t_j^{\text{small}}(a_{\max}^{\text{small}} - 1), \end{aligned} \tag{3.12}$$

with the aspect ratio for small unified proposals denoted with a_j^{small}. Scales can then be retrieved with

$$\begin{aligned}\sigma_{\min,j} &= \max\big(\sigma_{\min}, g_{\text{small}}\sigma_j^{\text{base}}\big)\,,\\ \sigma_{\text{maj},j} &= \min\big(a_j^{\text{small}}\sigma_{\min,j}, \sigma_{\max}\big)\,.\end{aligned}$$

For the pin proposals at the fine stage, the aspect ratio computation is determined from the edge strength and coherence qualities:

$$\begin{aligned}t_j^{\text{small,pin}} &= \text{clip}(0.6\,E(p_j) + 0.4\,C(p_j), 0, 1)\,,\\ a_j^{\text{small,pin}} &= \text{quantize}\Big(a_{\min}^{\text{small,pin}} + t_j^{\text{small,pin}}\left(a_{\max}^{\text{small,pin}} - a_{\min}^{\text{small,pin}}\right)\Big)\,,\end{aligned} \tag{3.13}$$

where $a_j^{\text{small,pin}}$ denotes the aspect ratio of fine pin Gaussians. The quantization operator maps a given continuous aspect ratio to the nearest value in a permitted discrete aspect ratio set. Once $\sigma_{maj,j}$, $\sigma_{\min,j}$, and θ_j are assigned, the covariance matrix of the j-th Gaussian is

$$\Sigma_j = R(\theta_j)\begin{bmatrix}\sigma_{maj,j}^2 & 0\\ 0 & \sigma_{\min,j}^2\end{bmatrix}R(\theta_j)^T,$$

where Σ_j denotes the covariance matrix of the j-th Gaussian and $R(\theta_j)$ is the standard 2×2 rotation matrix.

3.1.3 Hole-Patching

At lower Gaussian counts, initializations may face sparsity problems, especially after ADMM training in the normalized formula. To mitigate the affected regions, a simple coverage patch is implemented that attempts to target uncovered areas with large disk-like Gaussians. This step occurs after fine-stage fitting, as the final tuning stage. Let $Y(p)$ denote the target grayscale image, $\hat{Y}(p)$ the current fine-stage reconstruction, and $C(p)$ denote the raw Gaussian coverage at pixel p, where $C(p)$ can be found by

$$\begin{aligned}G_{pj} &= g_j(p),\\ C(p) &= \sum_{j=1}^{N} G_{pj},\end{aligned}$$

Where p denotes a pixel location, j a Gaussian index, $g_j(p)$, the raw value of a Gaussian at p, and $C(p)$ the Gaussian coverage. The target non-black mask is

$$M_T(p) = \mathbf{1}[Y(p) > \tau_T].$$

The reconstruction-dark mask is

$$M_R(p) = \mathbf{1}[\hat{Y}(p) \leq \tau_R].$$

The low-coverage mask is

$$M_C(p) = \mathbf{1}[C(p) \leq \tau_C].$$

Where

$$\begin{aligned} \tau_T &= 0.035, \\ \tau_R &= 0.010, \\ \tau_C &= 10^{-8}. \end{aligned}$$

The initial hole mask is then

$$H_0(p) = M_T(p)[M_R(p) \vee M_C(p)].$$

Equivalently,

$$H_0(p) = \mathbf{1}[Y(p) > \tau_T][\mathbf{1}[\hat{Y}(p) \leq \tau_R] \vee \mathbf{1}[C(p) \leq \tau_C]].$$

Small isolated regions are removed using a minimum-area threshold, then small morphological closing is applied to the nearby connected pixels, and thereafter, we can separate the hole mask into connected components, and process each hole according to its area. Areas under 12 pixels are neglected. Given P, the desired number of pixels per added Gaussian (90), then

$$n_k = \max(1, \lceil A_k/P \rceil).$$

A global patch budget (600) dictates the overall distribution. The final patch budget for a given patch is either the maximum remaining Gaussians or n_k. To decide the Gaussian placement candidate pixels are sorted using target brightness and random jitter.

$$\begin{aligned} s(p) &= Y(p) + 0.05\xi_p, \\ \xi_p &\sim U(0, 1). \end{aligned}$$

Which is further modulated with the aforementioned $d_{\min}$ to mitigate nearby centers.

Thereafter, the scaling coefficients of the Gaussians can be approximated using an equivalent circular radius, r_k of the hole area.

$$r_k = \sqrt{A_k/\pi}.$$

The initial pass scale σ_k can be defined as

$$\sigma_k^0 = 0.35 r_k.$$

The scale is clamped from $\sigma_{min} = 0.85$ to $\sigma_{max} = 5$. Then each Gaussian can be a circular disk with

$$\begin{aligned} \sigma_{k,\text{maj}} &= \sigma_k, \\ \sigma_{k,\text{min}} &= \sigma_k, \\ \theta_k &= 0. \end{aligned}$$

Given each new Gaussian from a patch, the initial coefficient can be set by using a Gaussian-weighted local mean of the target, where Ω denotes the local support of each Gaussian.

$$a_q = \frac{\sum_{p\in\Omega} Y(p) g_q(p)}{\max(\sum_{p\in\Omega} g_q(p), \epsilon)}.$$

It is then clipped to a valid intensity range (0-1, so filled atoms do not include negative coefficients). Here ϵ is the MATLAB default machine eps. The model with the newly appended hole-patching Gaussians is refit for 100 additional iterations, before proceeding with colour fitting.

3.1.4 Training Integration

To integrate the MATLAB implementation of the ADMM pipeline as a hot initialization, the MATLAB Gaussians must be converted to the GaussianImage repository [45] format. As such, the MATLAB initialization represents each Gaussian in image space coordinates, with centers in pixel coordinates, orientation as an angle in radians and the RGB coefficient. To use this initialization inside GaussianImage, physical parameters are first converted to learnable tensors expected by the PyTorch model. Pixel centers are normalized to the [-1,1] range, and mapped through inverse hyperbolic tangent, allowing the model's tanh

position activation to recover the intended normalized location. Gaussian scales are stored in the raw scale parameterization deployed by the model. The rotation is first subtracted form π then wrapped to 0-2π and then normalized from 0-1. Afterwards the rotation is converted with the inverse sigmoid transformation to match the expected sigmoid-based rotation parameter. The RGB coefficients are then directly copied into the feature tensor. To aid with sparsity issues, a densification and pruning pipeline is added to the structured pipelines.

Densification and pruning

Akin to the original Gaussian Splatting, to deal with areas of low detail and hole-like artifacts which are mitigated (but not removed by 3.1.3), a densification and pruning procedure is proposed. This method is applied on the structured initialization pipelines especially at low initial Gaussian amounts to promote reallocation in efficient areas. This adds a further layer of detail allocation in the process, to be compared with the original fully random initialization process.

Firstly, at a refinement stage a pruning score is defined from using the absolute colour-coefficient magnitude:

$$s_j = \frac{1}{3} \sum_{c \in \{r,g,b\}} |x_{j,c}|.$$

$$|\mathcal{P}| = \text{round}(\rho_p N_0)$$

Where N_0 is the initial Gaussian count for the fit, and $\mathcal{P}$ represents the set of Gaussians which are identified as pruning candidates. That is to say $\mathcal{P}$ contains the indices of the lowest scoring round$(\rho_p N_0)$ Gaussians.

Instead of immediately replacing pruned Gaussians, first, the Gaussians are made dormant by turning their opacities and colour coefficients to zero.

$$\begin{aligned} \mathbf{x}_j &\leftarrow \mathbf{0}, \qquad j \in \mathcal{P}, \\ \alpha_j &\leftarrow 0, \qquad j \in \mathcal{P}, \\ \sigma_{j,active} &\leftarrow \sigma_{dormant}, \qquad \sigma_{dormant} = 10^{-3}. \end{aligned}$$

This allows the model to be trained in this semi-dormant state, with dormant Gaussians ignored in training. The model is trained for 450 iterations (of the total 30000) before extra Gaussians are added (and dormant Gaussians are replaced). A modest growth rate is allotted for coverage, training parameters are:

$$\begin{aligned} \rho_p &= 0.06, \\ \rho_g &= 0.20, \\ \Delta t_{densify} &= 450. \end{aligned}$$

Where $\Delta t_{densify}$ is the delay between dormant training and densification, ρ_g is the total allowed proportional growth, and ρ_p is the fraction of Gaussians pruned per event. To compute the added Gaussians, the current reconstruction after dormant training is used to compute a residual map:

$$R(\mathbf{p}) = \text{norm01}\left(\sum_{c\in\{r,g,b\}} |Y_c(\mathbf{p}) - \hat{Y}_c(\mathbf{p})|\right).$$

This residual map is first sharpened before usage in densification.

$$R_0(\mathbf{p}) = \max(R(\mathbf{p}), 0)^{\gamma_R}.$$

Where

$$\gamma_R = 3.00.$$

An edge map is also computed, using Y_L as the target luminance image

$$E(\mathbf{p}) = \text{norm01}\left(\sqrt{[G_x Y_L(\mathbf{p})]^2 + [G_y Y_L(\mathbf{p})]^2 + \epsilon}\right).$$

Where Gx and Gy are Sobel edge kernels for horizontals and verticals respectively. Likewise, a similar term is found from the residual-gradient, to target saliency related proposals

$$R_g(\mathbf{p}) = \text{norm01}\left(\sqrt{[G_x R(\mathbf{p})]^2 + [G_y R(\mathbf{p})]^2 + \epsilon}\right).$$

Other maps from 3.1.2, particularly the edge map $E(\mathbf{p})$, coherence map $C(\mathbf{p})$, blob-likeness map $B(\mathbf{p})$, and local orientation map $\theta(\mathbf{p})$ are computed as before. Then an overall scoring map can be computed by:

$$A_{\text{sal}}(\mathbf{p}) = \Big[1 + \lambda_E E(\mathbf{p}) + \lambda_R R_g(\mathbf{p}) + \lambda_K K(\mathbf{p}) + \lambda_D D(\mathbf{p})\Big]^{\gamma_A},$$

Where $K(\mathbf{p})$ and $D(\mathbf{p})$ are defined below. $K(\mathbf{p})$ measures local contrast with the target luminance image Y_{lum}, and the blurred version of the same image $\overline{Y}_{\text{lum}}$. The quantity $d(\mathbf{p})$ represents the current normalized renderer denominator, and $\widetilde{d}$ is its spatial median. Allowing $D(\mathbf{p})$ to emphasize regions with below-median Gaussian coverage.

$$K(\mathbf{p}) = \text{norm01}\left(\left|Y_{\text{lum}}(\mathbf{p}) - \overline{Y}_{\text{lum}}(\mathbf{p})\right|\right),$$
$$D(\mathbf{p}) = \text{norm01}\left(\max\left(\widetilde{d} - d(\mathbf{p}), 0\right)\right).$$

For the final experiments, the saliency weights were

$$\lambda_E = 0.35, \qquad \lambda_R = 0.50, \qquad \lambda_K = 0.25, \qquad \lambda_D = 0.00, \qquad \gamma_A = 1.00.$$

The pin and disk proposal maps are then calculated with the augmented formula:

$$S^{ref}_{pin}(\mathbf{p}) = R_0(\mathbf{p}) A_{sal}(\mathbf{p}) \left[\lambda_{p,E} E(\mathbf{p}) + \lambda_{p,C} C(\mathbf{p}) + \epsilon\right],$$
$$S^{ref}_{disk}(\mathbf{p}) = R_0(\mathbf{p}) A_{sal}(\mathbf{p}) \left[\lambda_{d,B} B(\mathbf{p}) + \lambda_{d,S}(1 - E(\mathbf{p})) + \epsilon\right].$$

The pin and disk proposal weights used in the final experiment are

$$\lambda_{p,E} = 0.35, \qquad \lambda_{p,C} = 0.25, \qquad \lambda_{d,B} = 0.35, \qquad \lambda_{d,S} = 0.25.$$

Then Gaussian placement is decided by normalizing the maps and sampling them as distributions.

$$P^{ref}_{pin}(\mathbf{p}) = \frac{S^{ref}_{pin}(\mathbf{p})}{\sum_{\mathbf{q}} S^{ref}_{pin}(\mathbf{q}) + \epsilon},$$
$$P^{ref}_{disk}(\mathbf{p}) = \frac{S^{ref}_{disk}(\mathbf{p})}{\sum_{\mathbf{q}} S^{ref}_{disk}(\mathbf{q}) + \epsilon}.$$

The geometry assignment used during training-time densification differs slightly from the original MATLAB hot initialization. The python implementation during training utilizes residual and structure maps to determine the location of new Gaussians, and for pin like proposals the orientation (for disks orientation is random). The scale of new Gaussians is chosen utilizing modulation with the image density and current median scales from existing Gaussians, allowing for predictable behaviour after initialization.

Given a proposal center sampled from we first add a random jitter, then map it to the normalized coordinate domain, yielding $\tilde{\boldsymbol{\mu}}_j$.

Thereafter, the scale is initialized using the square-root of the average image area of each Gaussian, h. Then preliminary scaled are assigned for each of the pin and disk proposals

$$\begin{aligned}
\sigma^{pin}_{maj,j} &= m_{pin,maj}h, \\
\sigma^{pin}_{min,j} &= m_{pin,min}h, \\
\sigma^{disk}_{maj,j} &= m_{disk}h, \\
\sigma^{disk}_{min,j} &= m_{disk}h.
\end{aligned} \tag{3.14}$$

Incorporation of normalized rendering

The implementation is built on top of Gaussian Image [45], which incorporates the gsplat [43] library for Gaussian rendering. The rendering formulation can therefore use the same sum-rendering formulation as Gaussian Image [45], which is analogous to the previously proposed $\sum_{j=1}^{n} x_j g_j(\mathbf{p}_i)$ if we define some $x_{j,denom}$ as a column of ones, then the same rendering pipeline can be used after appending $x_{j,denom}$ (per channel), denoting the Gaussian rendering formula as

$$\mathcal{R}(X)_i = \sum_{j=1}^{n} g_j(\mathbf{p}_i) X_j,$$

where $\mathbf{p}_i$ is the pixel location, $g_j(\mathbf{p}_i)$ is the response of Gaussian j at that pixel, and X_j is the coefficient associated with Gaussian j. For RGB rendering, $X_j \in \mathbb{R}^3$, giving

$$\hat{Y}^{sum}_i = \mathcal{R}(X)_i = \sum_{j=1}^{n} g_j(\mathbf{p}_i)\mathbf{x}_j.$$

Then the CUDA rendering operation for summed Gaussians can simply be applied twice by replacing the RGB coefficient vector with the reshaped ones.

$$x_{j,\text{denom}} = 1, \qquad j = 1, \ldots, n.$$

The denominator is then

$$d_i = \mathcal{R}(x_{\text{denom}})_i = \sum_{j=1}^{n} g_j(\mathbf{p}_i) x_{j,\text{denom}} = \sum_{j=1}^{n} g_j(\mathbf{p}_i).$$

The normalized renderer is then

$$\hat{Y}_i^{norm} = \frac{\mathcal{R}(X)_i}{\mathcal{R}(x_{\text{denom}})_i + \epsilon} = \frac{\sum_{j=1}^{n} g_j(\mathbf{p}_i)\mathbf{x}_j}{\sum_{j=1}^{n} g_j(\mathbf{p}_i) + \epsilon}.$$

Thus, normalized rendering can be easily incorporated using the same sum rendering operation twice, by simply appending $x_{j,\text{denom}} = 1$ in place of the learned coefficients. Where ϵ is is typically between 10e-8 and 10e-10.

Sum-based initialization

To study the affect of the initialization process and the normalized formula choice separately, a separate sum-based formulation for initialization is conducted. While the processes are identical, the formulation simplifies slightly. For fixed Gaussian geometry, let

$$G_{ij} = g_j(\mathbf{p}_i),$$

where $g_j(\mathbf{p}_i)$ is the response of Gaussian j at pixel $\mathbf{p}_i$. In the normalized formulation, the blending matrix is defined previously in Eq (3.1). Recall that the rendered image is formed with:

$$\hat{\mathbf{y}} = B\mathbf{x}.$$

For the sum-based initialization, the operator B is simply replaced by G. so that the rendered image becomes

$$\hat{\mathbf{y}}^{sum} = G\mathbf{x}. \tag{3.15}$$

The ADMM objective is otherwise unchanged, except that every occurrence of B is replaced by G. Therefore, the coefficient update uses the same auxiliary variables and fidelity terms, but solves for coefficients under the original sum-rendering model:

$$\min_{\mathbf{x}} \frac{\lambda_{pix}}{2}\|\mathbf{w} \odot (G\mathbf{x} - \mathbf{y})\|_2^2 + \frac{\lambda_{grad}}{2}\left(\|D_x G\mathbf{x} - D_x\mathbf{y}\|_2^2 + \|D_y G\mathbf{x} - D_y\mathbf{y}\|_2^2\right) + \frac{\lambda_{lap}}{2}\|LG\mathbf{x} - L\mathbf{y}\|_2^2.$$

3.1.5 Quantization Contest

Gaussian Image provides a library extension to quantize Gaussians further using learned entropy encoding. In this process, Gaussians are allowed to train for several thousand

iterations within the quantized environment. Although this produces results with low bpp, image metrics and full pipeline time for first-time encoding may be impractical for some applications. Inspired by methods such as SOGS [27], an image-based sorted representation was developed. However, instead of adding further burden to training, the approach is done with a contest against discrete sorting techniques .

To efficiently represent Gaussians, image representations such as the png [41] or WebP [12] format can be used as the final storage medium. These mediums can greatly reduce file sizes [12, 19, 41]. The current quantization method for GaussianImage utilizes learned quantization techniques [45]. Although this yields effective compression results, on consumer grade resources the training times for such compression networks (which are per image) may be taxing. Likewise, especially for the normalized rendering formula, small accumulated Gaussian weights may amplify the effects of quantization errors for coefficients, especially in sparse regions, so parameter distributions may cause more destruction at harsh quantization scales. In general small pin like Gaussians and structured initialization Gaussians form differing distributions from the random initialization results as shown in Figure 3.13.

To deal with the instability of the normalized rendering, structured initialization effects, and the overall speed of the pipeline, an image based sorting method is developed. The method first sorts Gaussians in common and fast discretized patterns, then does a bpp comparison in the final saved Gaussian form, as well as a PSNR comparison between sorted- and unsorted (but still compressed) representations. If sorting does not greatly reduce the PSNR of the compressed image representation, then the identity or unsorted image is taken.

The Gaussian parameters start as row-wise tensors, where each row-tensor i, represents all parameters of a Gaussian, i. **The goal is to permute these tensors so that adjacent rows have smoother parameter variations that are efficiently compressed with the PNG and WebP formats.**

Overall sorting regime

As aforementioned, each Gaussian can be represented as a row tensor of the form:

$$\mathbf{a}_i = \{\boldsymbol{\mu}_i, \mathbf{s}_i, \theta_i, \mathbf{c}_i, \alpha_i\} .$$

Then the entire state of Gaussians can be:

$$\mathcal{A} = \{\mathbf{a}_i\}_{i=1}^{N} .$$

A sorting method can be described with a descriptor vector for each Gaussian, z_i:

$$z_i = (z_i^{(1)}, z_i^{(2)}, \ldots, z_i^{(m)}).$$

The row permutation is obtained by lexicographic sorting:

$$\pi = \text{sort}_{z^{(1)}, z^{(2)}, \ldots, z^{(m)}}(i). \tag{3.16}$$

In this statement for the sort π, each descriptor, z_i^k, is a sorting key, where the next descriptor z_i^{k+1} is used as a tie breaker, going down the chain.

After sorting every parameter tensor is sorted

$$G_\pi = \{g_{\pi_j}\}_{j=1}^N.$$

The reordered set can also be equivalently written as

$$g_j^\pi = g_{\pi_j}.$$

This is done by applying the same permutation to every row-aligned Gaussian parameter tensor. Noting that a model stores one Gaussian per tensor row, and its first dimension is N, the permutation can be described with A as:

$$A_\pi^{(k)}[j] = A^{(k)}[\pi_j], \qquad \text{shape}(A^{(k)})_1 = N.$$

Where the shape formulation simply describes each row-aligned parameter tensor $A^{(k)}$ has N rows.

The challenge in the discrete sorting methods is that sorting one parameter is almost definitely not the optimal sort for another. **Instead of learning a representation which optimizes all parameters, a contest is taken between several discrete techniques centered around one or more of the parameters.**

Morton sorting

Morton ordering converts the 2D Gaussian centers to coordinates in a 1D order. It is a transformation for the higher dimensional data that preserves the 2D spatial locality and converts to a 1D order. This ordering method is useful as nearby Gaussians often naturally have similar features in position, scale, and colour. The Gaussian centers are normalized and quantized into integer coordinates, after which interleaving the coordinate bits form a

Morton code, which can then be used to sort, and finally pack parameters into images.

Firstly, Gaussian centers are converted to a normalized unit square:

$$x_i' = \frac{x_i - x_{\min}}{x_{\max} - x_{\min} + \epsilon}, \qquad y_i' = \frac{y_i - y_{\min}}{y_{\max} - y_{\min} + \epsilon}.$$

After quantization this yields integer grid coordinates:

$$u_i = \lfloor (2^b - 1) x_i' \rfloor, \qquad v_i = \lfloor (2^b - 1) y_i' \rfloor.$$

The Morton code [28] is obtained by interleaving the bits of u_i and v_i, such that v=[u_1 v_1 u_2 v_2]. Yielding the order:

$$m_i = \text{bitinterleave}(u_i, v_i).$$

Snake spatial ordering

Snake ordering is a simpler method of spatial ordering. It groups Gaussians by their coarse row or column, and reverses the direction on alternating rows/columns. This reduces the jump between scanlines (at the end of one scanline and beginning of the next).

Given the numbers of coarse rows and columns H_g and W_g for horizontal coordinates a course row index is first defined as ρ_i:

$$\rho_i = \lfloor H_g y_i' \rfloor.$$

Then the horizontal coordinate is reversed on alternating rows:

$$\bar{x}_i = \begin{cases} x_i', & \rho_i \text{ even}, \\ -x_i', & \rho_i \text{ odd}. \end{cases}$$

The ordering is then:

$$\pi_{\text{xy-snake}} = \text{sort}_{\rho, \bar{x}}(i).$$

Then similarly for vertical ordering we define κ_i:

$$\kappa_i = \lfloor W_g x_i' \rfloor.$$

Then we can define π using $\bar{y}_i$:

$$\bar{y}_i = \begin{cases} y_i', & \kappa_i \text{ even}, \\ -y_i', & \kappa_i \text{ odd}. \end{cases}$$

$$\pi_{\text{yx-snake}} = \text{sort}_{\kappa, \bar{y}}(i).$$

Feature/colour sorting

While spatial sorting methods target the spatial locality synergizing with other parameters, colour based sorting methods focus on the sorting of Gaussian features (colours), which may dominate encoded size in some cases. Every colour sorting method follows from computing a scalar or vector descriptor from c_i then sorting the rows lexicographically.

Given a colour vector:

$$c_i = (r_i, g_i, b_i)$$

We can form a luminance descriptor with:

$$\ell_i = 0.299r_i + 0.587g_i + 0.114b_i.$$

Then we can form the ordering, π, with:

$$\pi_{\text{luma}} = \text{sort}_{\ell,y',x'}(i).$$

Sorting primarily by brightness, then using spatial coordinates as tie breakers.

We can also sort by the descriptor being the colour channels directly:

$$z_i = (r_i, g_i, b_i, y'_i, x'_i).$$

Resulting in an ordering:

$$\pi_{\text{RGB}} = \text{sort}_{r,g,b,y',x'}(i).$$

In this method we instead sort by using the RGB/colour vector, sorting by the value of the first channel, and then using second and third colour channels as secondary keys, and then the spatial coordinates as the final tie breakers.

Furthermore, instead of biasing brightness or any one colour channel, a hue-luminance method can be used. This method allows us to encode colour into hue, saturation, and luminance, and allows similar colours to be targeted by using hue as the primary descriptor. First defining:

$$\begin{aligned} M_i &= \max(r_i, g_i, b_i), \\ m_i &= \min(r_i, g_i, b_i) \\ d_i &= M_i - m_i. \end{aligned}$$

The saturation can be approximated with:

$$s_i^c = \frac{d_i}{M_i + \epsilon}.$$

The luminance as:

$$\ell_i = 0.299 r_i + 0.587 g_i + 0.114 b_i.$$

Then the hue can be approximated using the dominant channel:

$$h_i = \begin{cases} \frac{g_i - b_i}{d_i + \epsilon}, & M_i = r_i, \\ 2 + \frac{b_i - r_i}{d_i + \epsilon}, & M_i = g_i, \\ 4 + \frac{r_i - g_i}{d_i + \epsilon}, & M_i = b_i. \end{cases}$$

Thus the hue-luma ordering is:

$$\pi_{\text{hue-luma}} = \text{sort}_{h,\ell,s^c,y',x'}(i)$$

where we first convert colour vectors to hue luminance and saturation, then sort primarily by hue, use luminance and saturation as the secondary keys, and then spatial coordinates as tie breakers.

Principal-component-axis (PCA) sorting can also be used. This attempts sorting through the projection of the colour vector onto its dominant variation direction. Finding the mean colour vector as:

$$\bar{c} = \frac{1}{N} \sum_{i=1}^{N} c_i.$$

Then v_1 can be defined as the first principal component of the centered colour matrix and the PCA score can be defined as:

$$p_i = (c_i - \bar{c})^T v_1.$$

Thus the PCA ordering is:

$$\pi_{\text{PCA}} = \text{sort}_{p,y',x'}(i).$$

This ordering is done through collecting colour vectors, subtracting the mean colour vector, computing the dominant PCA direction, and projecting each colour vector onto that

direction. Afterwards, sorting is done by the scalar projection, and spatial coordinates are used as tie breakers.

Lastly, some orderings combine spatial and colour descriptors, a spatial-primary ordering is

$$\pi_{\text{xy-feature}} = \text{sort}_{y',x',\ell}(i).$$

Primarily using spatial locality, and luminance as the secondary key. While a colour primary ordering is:

$$\pi_{\text{feature-xy}} = \text{sort}_{h,\ell,s^c,y',x'}(i).$$

Primarily grouping similar colours and using spatial qualities as tie breakers. Spatial primary tends to preserve smoothness in the position maps, while colour primary sorting tends to reduce entropy in the corresponding maps.

Contest specifications

All sorting methods described above are applied to the Gaussian parameters, and rated and chosen by bpp drop. The visualization of the effects of these sorts can be seen in Figure 3.3. For safety an extra guard must be placed. A pure mathematical Gaussian evaluation should be invariant to row order; however, the implemented renderer can be sensitive to some fitted parameter configurations. Thus, each proposed order is first tested by rendering the sorted model, and evaluating any degradation between the pre-sorted and post-sorted models. Let a candidate order be:

$$G_{\pi_o} = \{g_{\pi_{o,j}}\}_{j=1}^{N}.$$

The original fitted render can be represented as

$$I_{\text{fit}} = R(G),$$

and the render after applying order, o, is

$$I_o = R(G_{\pi_o}).$$

The candidate order is accepted only if the sorted render order does not break the rendering regime:

$$\text{PSNR}(I_o, I_{\text{fit}}) \geq \eta,$$

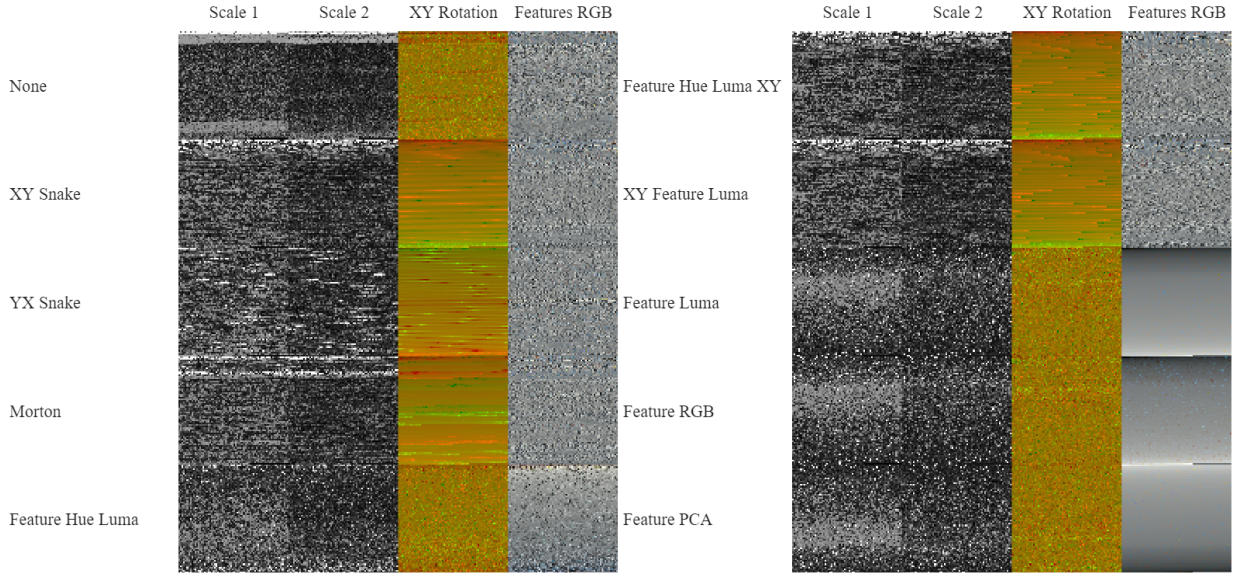


Figure 3.3: Visualization of the several sorting methods outlined for the contest. Features are saved as RGB, x-y-rotation is combined as RGB, and scale channels are separated.

There is also an optional check to ensure that the sorted order does not degrade from the original image:

$$\mathrm{PSNR}(I_{\mathrm{fit}}, I) - \mathrm{PSNR}(I_o, I) \leq \tau.$$

In the stricter experiments:

$$\eta = 100 \text{ dB}.$$
$$\tau = 0.10 \text{ dB}.$$

Note that the sorted I_o is computed without saving the codec or any quantization, it is the result of rendering after permutation of the raw fitted result. These thresholds reject candidate orderings that would measurably affect or artifact the fitted render, the valid candidate orderings are defined by:

$$\mathcal{O}_{\mathrm{valid}} = \{o \in \mathcal{O} : \mathrm{PSNR}(I_o, I_{\mathrm{fit}}) \geq \eta \quad \text{and} \quad \mathrm{PSNR}(I_{\mathrm{fit}}, I) - \mathrm{PSNR}(I_o, I) \leq \tau\}. \quad (3.17)$$

The selected ordering is chosen as the lowest bitrate among accepted candidates:

$$o^* = \arg \min_{o \in \mathcal{O}_{\mathrm{valid}}} \mathrm{bpp}(o).$$

Likewise, if no sorted order satisfies constraints, then the original candidate order is preserved:

$$o^* = \text{id}.$$

Where here 'id', is simply the original order. The total bitrate is measured as

$$\text{bpp} = \frac{8B_{\text{total}}}{HW},$$

Where B_{total} includes all stored parameter images and metadata.

PNG and WebP storage

The final storage medium of the Gaussian parameters is either a PNG or WebP image, where either of the mediums benefit from smooth storage. While WebP has a longer encoding time than PNG, it generally outperforms in the storage of smooth images [12], as such WebP is expected to win the bpp contest; however, the faster encode time of PNG may yield practical use-cases.

For experiments, the codec tests both PNG and lossless WebP storage using quantized parameter images. First, Gaussians are quantized to constant bit depths for each parameter. 16 bits are used for position, 10 for scale, 12 for rotation, and 8 for colour. For each scalar parameter channel, x, the values are linearly mapped to an integer range using the minimum and maximum. Later runs use differing bit-depths for parameters particularly medium quality and low quality, utilizing better quantization transformations from further experiments in Section 3.3.3. Table 3.1 reflects the bit ranges and transformations utilized in the experiments.

Quantized values are stored in image arrays. For a channel containing only 8 bit data:

$$I_q[i] = q_i.$$

For 16-bit integer values,

$$q_i \in \{0, \ldots, 65535\},$$

It is stored in a low-high byte split with:

$$q_i^{\text{lo}} = q_i \bmod 256$$
$$q_i^{\text{hi}} = \lfloor q_i/256 \rfloor.$$

Table 3.1: Quantization settings for parameters before image-codec saving at different regimes, high, medium, and low. Note resultant PSNR values differ from high and medium/low ranges disproportionately due to testing and optimization of quantization shown in Section 3.3.3.

Regime	Parameter group	Transform	Bits
High	Position (x, y)	Linear	16
	Scale (s_1, s_2)	Linear	10
	Rotation θ	Linear	12
	Colour/features (r, g, b)	Linear	8
Medium	Position (x, y)	Linear	16
	Scale (s_1, s_2)	Log1p Med	8
	Rotation θ	Linear	10
	Colour/features (r, g, b)	Asinh P90	8
Low	Position (x, y)	Linear	14
	Scale (s_1, s_2)	Log1p Med	8
	Rotation θ	Linear	8
	Colour/features (r, g, b)	Asinh P90	8

The value can then be reconstructed as

$$q_i = q_i^{\text{lo}} + 256 q_i^{\text{hi}}.$$

For the WebP storage, channels that contain more than 8 bits are stored as separate lossless WebP images:

$$I_q^{\text{lo}} \quad \text{and} \quad I_q^{\text{hi}}.$$

PNG simply uses a 16-bit PNG for these cases.

3.1.6 Experiment Details

The python-end experiment is built on top of the GaussianImage library [45], which utilizes the gsplat [43] library. The experiments were run on the Kodak dataset [11], unlike other methods [46, 4, 15], I test natural images particularly at under 1 bpp and 15000 Gaussians.

The experiment is conducted as a three-stage pipeline, first starting with the Gaussian initialization in MATLAB, which is then converted to the Gaussian Image format. The tested systems are:

- The proposed full initialization and refinement pipeline, paired with optimization using the normalized rendering formula. (GS-Sketch)
- The sum-based initialization described in Section 3.1.4 and refinement pipeline paired with regular summed Gaussian optimization. (GS-Sketch(sum))
- The GaussianImage [45] baseline with randomized sum initialization. (GI)

The three system choices permit a partial isolation of the design choices, examining specifically the effect of normalized and additive rendering in the structured pipelines, and the comparison between structured initialization and refinement against random initialization in the sum pipeline. As the full pipelines include a late fitting stage, as well as refinement, hole repair, and other strategies, the resulting comparisons should be interpreted as against full pipelines, and not evidence for any single pipeline addition. Likewise, to commonly refer to the proposed methods, the terms 'structured initialization and refinement' and 'structured pipeline(s)' may both be used, referring to the holistic structured pipeline, and not any individual element.

The GI baseline also uses one deterministic case-indexed random seed for each Kodak image. This provides reproducible results which are similar to results in GaussianImage [45], but fully estimating uncertainty of random initializations is not within the scope of the project.

The original MATLAB code utilizes heuristic and manually tuned proposal counts to target approximately 10,000 Gaussians. Therefore, a modulation strategy to target different numbers of Gaussians is developed.

The GI [45] baseline uses the same seeded-random initialization style as the original GaussianImage implementation [45]; however, the experiment is not an exact recreation due to differing quantization iterations and training protocol iterations (early stop) and different random draws. In the presented pipeline, the random baseline is deterministic and case-indexed so that each Kodak image has reproducible random initializations. Likewise, final results maintain similarity with original GaussianImage [45] reported results.

Modulating initialization and trained Gaussian budgets

The reported main experiments utilize a sweeping version of the code, that uses the original results to scale the requested budget. Here K defines the requested budget. Given some

K requested (say 3000), then using the original $K_{ref} = 10000$

$$K_{\text{ref}} = 10000,$$
$$s(K) = \frac{K}{K_{\text{ref}}}.$$

The original stage proposal counts can be defined with:

$$N_{L,0} = N_{L,d,0} + N_{L,p,0},$$
$$N_{M,0} = N_{M,d,0} + N_{M,p,0},$$
$$N_{S,0} = N_{S,d,0} + N_{S,p,0},$$

Where subscripts d and p, represent disk and pin, and N represents the total proposal counts. Due to the fine stage having 3 passes, the proposal budget is:

$$N_{\text{eff},0} = N_{L,0} + N_{M,0} + P_S N_{S,0}.$$

Where P_S is the number of fine-stage passes by default 3. Given a desired budget for K:

$$N_{\text{eff}}(K) = \text{round}[s(K) N_{\text{eff},0}].$$

We can then scale stage budgets using the original structure:

$$N_L = \text{round}[s(K) N_{L,0}],$$
$$N_M = \text{round}[s(K) N_{M,0}]$$
$$N_S = \text{round}\left[\frac{N_{\text{eff}}(K) - N_L - N_M}{P_S}\right].$$

Due to the screening, deduplication, local spacing constraints, and the hole-patch procedure, the final number of exported Gaussians is image-dependent. The tentative sweep ranges were scaled as referenced by the original run to 1000, 3000, 7000, and default (≈10000). The final run uses hybrid refinement events rather than a fixed-only schedule. A drop-triggered event is allowed when the recent PSNR has decreased by more than the tolerance. The ranges for the labeled runs (label N03000, for example for the intended 3000+ run), are shown in Table 3.2. During joint training a **densification growth rate of 0.2** is allotted, adding further Gaussians, as shown in Table 3.3.

The GI baseline is trained after structured pipelines, and for every compared image, the final count of the GS-Sketch system (normalized rendering) is used to modulate growth from

Table 3.2: MATLAB hot-initialization Gaussian count ranges before training and count matching.

Budget regime	GS-Sketch	GS-Sketch(sum)	Combined ranges
N01000	1719-2005	1741-2008	1719-2008
N03000	3364-4173	3307-4154	3307-4173
N07000	6366-8423	6127-8087	6127-8423
NDEFAULT	8284-11514	7994-11055	7994-11514

Table 3.3: Final fitted/evaluated Gaussian count ranges for each budget regime after densification growth.

Budget regime	Final Gaussian count range
N01000	2063-2406
N03000	4037-5008
N07000	7639-10108
NDEFAULT	9941-13817

densification to reach the same Gaussian count in GS-Sketch(sum), and the GI baseline is initialized with the same amount of Gaussians as both. This ensures that **per image comparison** the **same number of Gaussians is used**, results are still grouped by these broad ranges and budget regimes.

Additionally, some further modulation of disk-pin splits is conducted to promote a larger number of pin Gaussians at lower primitive budgets, this ensures high frequency detail is captured, but may cause bias towards such areas in constrained Gaussian counts. For each stage (a), let $(r_{a,0})$ be the default pin share and $(r_{a,\text{low}})$ be the pin share used in the low-budget setting. The interpolation weight is

$$
\begin{aligned}
s(K) &= \frac{K}{K_{\text{ref}}}, \\
s_{\text{low}} &= \frac{K_{\text{pin}}}{K_{\text{ref}}}, \\
\beta(K) &= \text{clip}\left(\frac{1 - s(K)}{1 - s_{\text{low}}}, 0, 1\right)^{\gamma}.
\end{aligned}
$$

In the reported experiments, ($K_{\text{pin}} = 3000$) and ($\gamma = 1.0$). Therefore, ($\beta(K) = 0$) at the reference budget (K=10000), ($\beta(K) = 1$) for budgets at or below (K=3000), and intermediate budgets use a proportional interpolation between the two settings.

The pin share for stage (a) is then

$$r_a(K) = [1 - \beta(K)]r_{a,0} + \beta(K)r_{a,\text{low}}.$$

For a total number of proposals ($N_a(K)$) for stage (a), pin and disk proposal counts can be shown with

$$N_{a,p}(K) = \text{round}[r_a(K)N_a(K)],$$
$$N_{a,d}(K) = N_a(K) - N_{a,p}(K).$$

Training system

The three aforementioned systems (GI, GS-Sketch, GS-Sketch(sum)) are trained with ℓ_2 reconstruction loss:

$$\frac{1}{HW}\sum_p |\hat{I}(p) - I(p)|_2^2.$$

Where H and W are the dimensions of the image, p represents a pixel, and $\hat{I}$ the fit/reconstruction while I is the original image. As the structured pipelines struggle with coverage, late fitting is conducted with an ℓ_1+MS-SSIM loss.

$$\text{Late loss} = (1 - \lambda)[1 - \text{MS-SSIM}(\hat{I}, I)] + \lambda\frac{1}{HW}\sum_p |\hat{I}(p) - I(p)|, \tag{3.18}$$

With $\lambda = 0.7$.

Training is conducted with 30000 iterations for the structured pipelines, while the baseline GI is allotted 50000 iterations, with a stagnation early-exit. The early exit is defined when a 2000 iteration window does not drift beyond a stagnation tolerance of 0.02 PSNR.

The prune fraction and growth fraction for densification and pruning is 0.06 and 0.2 respectively. Prune score is based on the absolute colour coefficient magnitude:

$$\frac{1}{3}\sum_{c\in r,g,b} |x_{j,c}|.$$

The number of pruned Gaussians can be defined given the current number of Gaussians N_0:

$$N_{\text{prune}} = \text{round}(\rho_p N_0),$$
$$\rho_p = 0.06.$$

The other relevant heuristic decisions are a maximum respawn count of 12, 450 iterations of delay between pruning and respawn, and densification and refinement are disabled for the final 2000 iterations. The MS-SSIM related loss is used for the final 1500 iterations. Timing metrics are reported for post-realized iterations like crossing iterations by taking the proportional time for that iteration compared to the entire run.

Quantization testing

To test quantization, the fit results of the N03000 set are used. The GI-baseline learned entropy-encoding is allotted 25000 iterations. The quantization run for entropy-encoded representations is run separately from tests of the WebP and PNG formats due to rapid development procedures. After initial trials, linear quantization is replaced with asinh and log based functions which are described in later Section 3.3.3.

The tested sorting systems are outlined in Section 3.1.5. Likewise, the list of participating sorting methods is seen below:

- (xy)-snake,
- (yx)-snake,
- Morton,
- feature hue/luma,
- feature hue/luma with spatial grouping,
- spatial plus feature luma,
- feature luma,
- feature RGB,
- feature PCA.

Quantization testing contains experiments from initial then adjusted runs. Based on the results of the winning sorts, and storage formats further testing is done.

Evaluation metrics

Image quality is assessed by two common image quality metrics, and a proposed task-specific Edge-PSNR measured from Sobel-gradient magnitudes. The PSNR [18] metric is used to discern general quality. Thereafter an edge-PSNR metric is derived from Sobel gradient magnitudes, where S_x and S_y are Sobel filters:

$$E(I)(p) = \sqrt{[S_x I(p)]^2 + [S_y I(p)]^2},$$

Then

$$\text{MSE}_E = \frac{1}{HW} \sum_p [E(\hat{I})(p) - E(I)(p)]^2,$$

$$\text{EdgePSNR} = 10 \log 10 \frac{1}{\text{MSE}_E}.$$

Edge-PSNR is interpreted as a gradient-domain measure of reconstruction validity, and not claimed to be a predictor of human perceptual preference. Lastly for texture the MS-SSIM [40] metric is used. Timing is also measured on an RTX3060 GPU and AMD Ryzen 5 5600G CPU, **note that all experiments are kept on consumer grade hardware.**

3.2 Results

The following sections detail results compiled from performance on the Kodak dataset [11], for both Gaussian fits, and final compression scenarios.

3.2.1 Low-Primitive Representation

The results of the final fit image quality metrics (PSNR [18], Edge-PSNR, MS-SSIM [40]) can be found in Table 3.4 and for the peak fits Table 3.5. It can be seen that the proposed structured initialization and refinement systems outperform the baseline in Figure 3.6 in terms of PSNR and edge PSNR at every primitive range. Peak iterations also follow the same trend, where the difference becomes most prevalent in the 8-15k Gaussian range, where over 1dB of PSNR difference is noticed between GS-Sketch and the baseline. The peak iterations consistently occur before the baseline peak. The proposed GS-Sketch and GS-Sketch(Sum) pipelines outperform the baseline in PSNR and Edge-PSNR, while

trailing slightly behind in texture scores (MS-SSIM). The gap in MS-SSIM is reduced as budgets increase, at 0.0208 at 2063-2406, to 0.0032 at 9941-13817.

Figure 3.4 reflects the initialization and peak fit results for kodim10, where both structured initialization and refinement systems are seen to achieve PSNR values higher than the baseline with random initializations. Likewise, the GS-Sketch(sum) variant is outperformed by the GS-Sketch variant, in terms of both initialization and peak fit quality, in all ranges but the initial approximately 2000 Gaussian range. The effect of the structured initialization and refinement systems can be visualized by Figure 3.5, where the structured initializations and refinement provide much more directed coverage near important areas of the image, resulting in less noticeable reconstruction artifacts, as seen in the eyes of the birds in Kodim23. Likewise, it can be seen that the normalized formula allows for smoother blending of large Gaussians, where ellipsoidal artifacts are noticed in the same region of the GS-Sketch(Sum) variant, but are blended smoothly (as in the target image) by the GS-Sketch variant with normalized rendering.

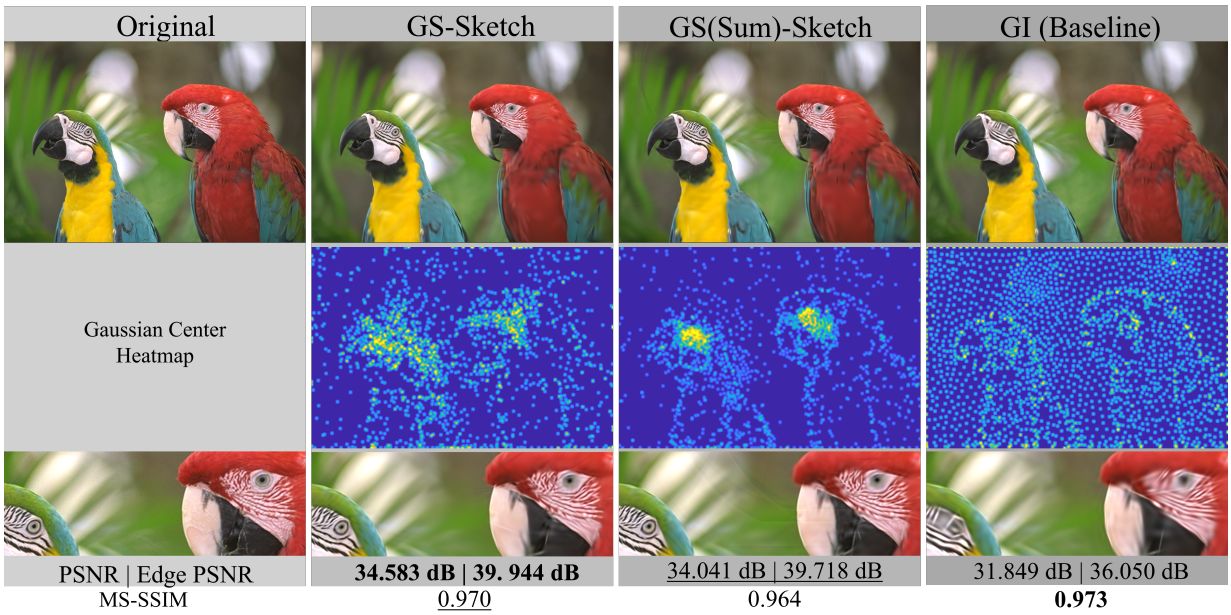


Figure 3.4: GS-Sketch, GS-Sketch(sum), and baseline GI systems at various Gaussian counts on Kodak Image 10. The quality of both initializations and final fits increases drastically with increasing Gaussian counts. At higher Gaussian counts, GS-Sketch consistently produces higher PSNR and edge PSNR over the other tested methods.

Table 3.4: Average final-fit reconstruction quality and training time by budget regime and system.

Budget	System	Edge-PSNR	PSNR	MS-SSIM	Time (s)
2063-2406	GS − Sketch	32.20	27.49	0.9039	180.3
2063-2406	GS − Sketch(sum)	**32.65**	**28.13**	0.9125	151.0
2063-2406	GI(baseline)	31.47	27.47	**0.9247**	131.2
4037-5008	GS − Sketch	**35.45**	**30.62**	0.9496	236.2
4037-5008	GS − Sketch(sum)	35.44	30.51	0.9480	172.4
4037-5008	GI(baseline)	34.12	29.56	**0.9592**	133.4
7639-10108	GS − Sketch	**39.07**	**33.46**	0.9759	281.1
7639-10108	GS − Sketch(sum)	38.56	32.92	0.9722	191.4
7639-10108	GI(baseline)	37.53	32.04	**0.9801**	137.1
9941-13817	GS − Sketch	**40.70**	**34.62**	0.9830	299.8
9941-13817	GS − Sketch(sum)	40.12	34.06	0.9800	200.0
9941-13817	GI(baseline)	39.49	33.40	**0.9862**	143.1

Table 3.5: Average peak-fit PSNR and peak iteration by budget regime and system.

Budget	System	Peak PSNR	Peak iteration
2063-2406	GS − Sketch	28.09	**20808**
2063-2406	GS − Sketch(sum)	**28.31**	26527
2063-2406	GI(baseline)	27.47	33136
4037-5008	GS − Sketch	**30.81**	**25631**
4037-5008	GS − Sketch(sum)	30.70	26399
4037-5008	GI(baseline)	29.56	33348
7639-10108	GS − Sketch	**33.49**	26735
7639-10108	GS − Sketch(sum)	33.16	**24469**
7639-10108	GI(baseline)	32.04	33918
9941-13817	GS − Sketch	**34.67**	27538
9941-13817	GS − Sketch(sum)	34.35	**23874**
9941-13817	GI(baseline)	33.40	34567

Lastly, Table 3.6 shows the average crossing iteration in each budget regime for the structured initialization and refinement methods to overtake the final GI baseline fit. It can be seen that crossing times are generally about 1/3 to 1/2 the training time of the original method, while also producing more interpretable, structured, and higher PSNR results.

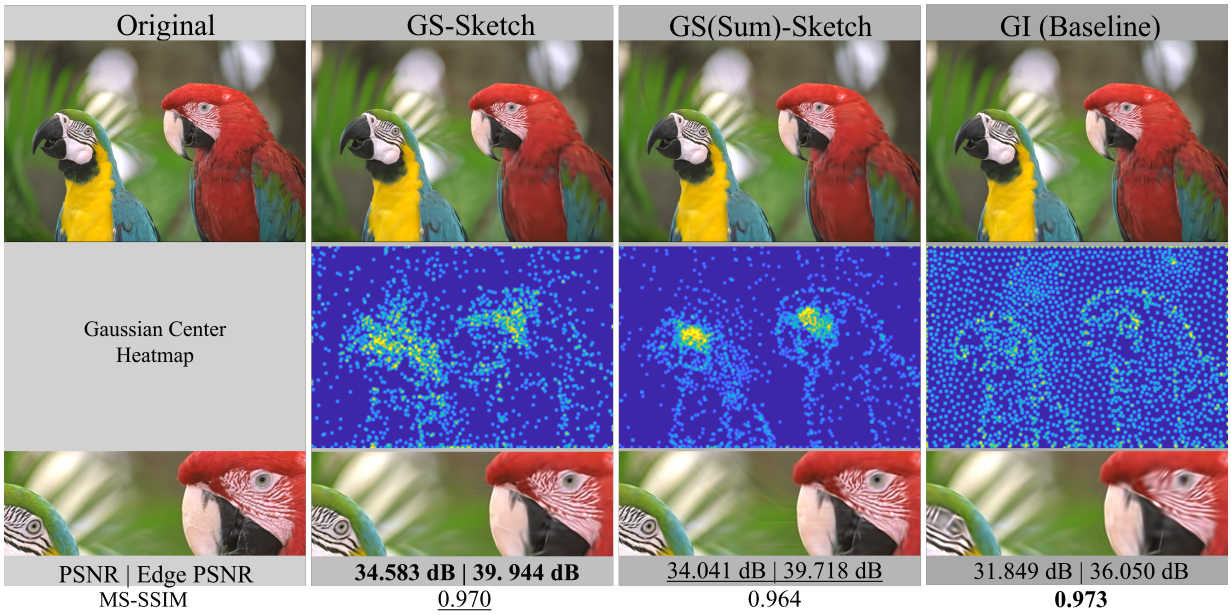


Figure 3.5: Effects of final fit from different systems on Kodim23.png with 2206 Gaussians. The proposed method provides emphasis on important high-frequency information, the normalized rendering is seen to adequately map smooth surface features with large Gaussians without artifacts.

3.2.2 Compression Approaches

The proposed compression method with linear quantization of parameters can be seen in Table 3.7. It can be seen in general that the native quantization (learned codebook technique) achieves much lower bpp, but causes significantly more degradation, especially in the initialized variants, which have differing parameter distributions to the random initialization as discussed in Section 3.3.3.

The bpp ranges are generally slightly higher in the proposed WebP/PNG based codec; however, are particularly competitive with the results reported in Figure 4 of the rate-distortion curves in GaussianImage [45], this estimate shows that the native quantizer reaches 30PSNR at $\geq$ 1 bpp while their bounded estimate achieves it approximately 0.9 bpp. The high WebP variant for the normalized GS-Sketch system reaches this benchmark at around 0.846 bpp. Likewise, further variants exceed this 30 dB PSNR value at 0.8 bpp for the same system as shown in Table 3.9. While the codec is outperformed by native

Table 3.6: Average crossing-fit PSNR and proportional training time by budget regime and system.

Budget	System	Crossing PSNR	Crossing iter.	Time frac.	Time (s)
2063-2406	GS − Sketch	27.76	6764	0.225	40.0
2063-2406	GS − Sketch(sum)	27.47	7368	0.246	37.0
2063-2406	GI(baseline)				131.2
4037-5008	GS − Sketch	29.56	5132	0.171	39.6
4037-5008	GS − Sketch(sum)	29.56	5640	0.188	32.4
4037-5008	GI(baseline)				133.4
7639-10108	GS − Sketch	32.04	5585	0.186	51.9
7639-10108	GS − Sketch(sum)	32.04	6975	0.232	44.8
7639-10108	GI(baseline)				137.1
9941-13817	GS − Sketch	33.40	7126	0.238	70.4
9941-13817	GS − Sketch(sum)	33.55	8368	0.279	56.2
9941-13817	GI(baseline)				143.1

quantization in terms of bpp reduction, the learned codebook approach suffers with the differing distributions in the proposed structured pipelines. The proposed codec has a slightly increased performance at the medium to high bpp ranges, while uncompetitive in providing minimal bpp (about 0.08bpp increase for the regular GS-Sketch system), it is competitive when compared to the rate distortion curves in [45] at medium to high bpp ranges (0.7-1 bpp), with the GI baseline for natural images. Additionally, there is a significant time reduction in total encoding time, although the learned codebook representation has a lower post-training encoding time. In a practical sense, since each codebook has to be learned separately, per image, in many applications first-time encode time will be relevant. In this sense we benchmark that total time reduces from more than 5 minutes, to under 20 seconds, and under 5 seconds in the PNG variant as shown in Tables 3.8 and 3.10.

The visual results for the proposed image codec method with the WebP track are reflected in Figure 3.7. The native learned codebook quantization of the same image results in PSNR for GS-Sketch, GS-Sketch(Sum), and GI(baseline) as 29.62 dB, 28.41 dB, 30.17 dB. It can be seen that the GI baseline outperforms (as stated previously) in the native quantization method, but in some cases the WebP compression outperforms native quantization, while retaining higher quality. Likewise, the gaps from improved fits are better retained by the proposed compression pipeline for the GS-Sketch pipeline after compression; however, at lower ranges the sum variant begins to be outperformed by GI baseline. This illustrates

the competitiveness of our proposed codec, while maintaining the per-image practicality to avoid further codebook optimization. Figure 3.7 also shows that small details are better retained by our proposed methods, adding sharpness and representation in visually important regions of the image.

Table 3.7: Average compression quality for the high-rate image-codec setting. Native GaussianImage quantization is repeated as a reference; the improved transform quantization results are discussed in the following low/medium-rate comparison.

Setting	System	bpp	PSNR	Edge PSNR	MS-SSIM	PSNR Drop
Native Quant.	GS-Sketch	0.653	26.811	31.950	0.8937	3.808
WebP High	GS-Sketch	0.846	30.001	35.373	0.9442	0.617
PNG High	GS-Sketch	1.027	30.001	35.373	0.9442	0.617
Native Quant.	GS-Sketch(sum)	0.653	26.581	32.077	0.8914	3.926
WebP High	GS-Sketch(sum)	0.850	28.931	35.351	0.9384	1.576
PNG High	GS-Sketch(sum)	1.034	28.931	35.351	0.9384	1.576
Native Quant.	GI (baseline)	0.653	27.634	32.209	0.9323	1.929
WebP High	GI (baseline)	0.800	29.145	34.087	0.9474	0.418
PNG High	GI (baseline)	0.974	29.145	34.087	0.9474	0.418

Table 3.8: Average timing for the high-rate image-codec setting. Native quantization includes codebook learning/quantization for total time, while winner encode reports only the final selected compression step.

Setting	System	Total Time	Winner Enc.	Decode
Native Quant.	GS-Sketch	369.79	0.004	0.002
WebP High	GS-Sketch	24.51	1.581	0.015
PNG High	GS-Sketch	3.95	0.101	0.013
Native Quant.	GS-Sketch(sum)	311.72	0.003	0.001
WebP High	GS-Sketch(sum)	27.85	1.532	0.017
PNG High	GS-Sketch(sum)	4.55	0.106	0.014
Native Quant.	GI (baseline)	315.27	0.004	0.001
WebP High	GI (baseline)	18.78	1.580	0.016
PNG High	GI (baseline)	3.08	0.105	0.013

Table 3.9: Average compression quality for the low- and medium-rate transform image-codec settings, with native GaussianImage quantization repeated for comparison.

Setting	System	bpp	PSNR	Edge PSNR	MS-SSIM	PSNR Drop
Native Quant.	GS-Sketch	0.653	26.811	31.950	0.8937	3.808
WebP Low	GS-Sketch	0.731	28.496	34.044	0.9310	2.123
PNG Low	GS-Sketch	0.856	28.496	34.044	0.9310	2.123
WebP Medium	GS-Sketch	**0.800**	**30.115**	35.247	0.9460	0.503
PNG Medium	GS-Sketch	0.927	30.115	35.247	0.9460	0.503
Native Quant.	GS-Sketch(sum)	0.653	26.581	32.077	0.8914	3.926
WebP Low	GS-Sketch(sum)	0.739	27.772	34.017	0.9256	2.735
PNG Low	GS-Sketch(sum)	0.877	27.772	34.017	0.9256	2.735
WebP Medium	GS-Sketch(sum)	0.812	29.432	35.232	0.9432	1.076
PNG Medium	GS-Sketch(sum)	0.944	29.432	35.232	0.9432	1.076
Native Quant.	GI (baseline)	0.653	27.634	32.209	0.9323	1.929
WebP Low	GI (baseline)	0.692	28.192	32.882	0.9485	1.371
PNG Low	GI (baseline)	0.781	28.192	32.882	0.9485	1.371
WebP Medium	GI (baseline)	0.758	29.324	33.943	0.9567	0.239
PNG Medium	GI (baseline)	0.855	29.324	33.943	0.9567	0.239

3.3 Discussion

The following sections will detail the various technical and qualitative observations of the method, that lead towards the interpretation of the results, limitations and issues in the current implementation, and defines the paths for further research directions. The various limitations revealed will be reiterated in the dedicated Section 4.2.1. Note that a primary limitation is the lack of full ablations for each design decision individually; however, limited ablations are done between design choices (sum and normalized rendering, with and without densification for normalized rendering). As such, most claims are relative to the entire structured initialization and refinement pipeline, and full length ablations are out of the current project scope and deferred to future work.

3.3.1 Sharpness-Coverage Tradeoff

The results suggest a tradeoff between sharpness and coverage when tuning initializations and refinement towards image details. Random initializations generally provide better coverage, mapping subtle texture details more accurately, whereas the high-frequency bias in the proposed methods generally causes these regions to be represented with large-smooth

Table 3.10: Average timing for the low- and medium-rate transform image-codec settings, with native GaussianImage quantization repeated for comparison. For image codecs, total time includes the full adaptive contest and winner encode is only the selected packing/order.

Setting	System	Total Time	Winner Enc.	Decode
Native Quant.	GS-Sketch	369.79	0.004	0.002
WebP Low	GS-Sketch	17.26	1.083	0.014
PNG Low	GS-Sketch	4.37	0.117	0.014
WebP Medium	GS-Sketch	17.92	1.135	0.012
PNG Medium	GS-Sketch	4.38	0.110	0.013
Native Quant.	GS-Sketch(sum)	311.72	0.003	0.001
WebP Low	GS-Sketch(sum)	19.62	1.008	0.013
PNG Low	GS-Sketch(sum)	4.96	0.119	0.014
WebP Medium	GS-Sketch(sum)	20.29	1.087	0.012
PNG Medium	GS-Sketch(sum)	4.90	0.119	0.012
Native Quant.	GI (baseline)	315.27	0.004	0.001
WebP Low	GI (baseline)	12.62	1.037	0.012
PNG Low	GI (baseline)	3.42	0.115	0.013
WebP Medium	GI (baseline)	13.30	1.043	0.012
PNG Medium	GI (baseline)	3.36	0.108	0.013

primitives. At higher Gaussian counts, the structured initialization and refinement based methods narrows the texture gap in terms of MS-SSIM compared to the baseline method. This can have interesting effects, especially in human-subject natural images. As seen in Figure 3.8, skin qualities which are commonly visually/aesthetically low-frequency, but may contain subtle textures are reduced and smoothed in the initialized approaches. In the baseline approach, the coverage of random initializations ensures such textures are represented more uniformly. As a tradeoff, the sharp features like those around the eyes and mouth are enhanced with the initialized approaches.

The efficiency of the normalized rendering formula and late fit loss mitigate this weakness, which is more evident from GS-Sketch outperforming GS-Sketch(Sum) in this metric, while both are outperformed by the baseline GI with random initializations. Moreover, particularly with the normalized rendering formula, which allows for more complex interactions, and generally larger distributions for feature coefficients, black NaN/zero artifacts are noticed, especially in regions of highly anisotropic or large Gaussians. The structured initialization and refinement pipelines produce larger and thinner Gaussians, that may cause unexpected interactions with the optimized rendering pipelines,

also evident in Section 3.3.4. Such artifacts, and the tradeoff between subtle texture and high-frequency information can be seen in Figure 3.9.

3.3.2 Normalized Rendering

The benefits to the normalized rendering equation come primarily with how denominator interactions allow for easier single-tone filling of sparse areas as shown in Figure 3.10. Another benefit beyond smooth colour representation with minimal Gaussians is allowing for different behaviour for edge interactions. For the normalized renderer, an edge Gaussian can simply be overlayed or adjacent to a region Gaussian, although sparse space will be filled from denominator interactions, given a higher opacity (or weight in our case) for the edge Gaussian, it will remain visible at its intended colour. In the same case for a sum renderer, more Gaussians may be used due to this simple blending interaction not being as viable. The sum renderer in this case will struggle as the center edge Gaussian, if placed over top or adjacent to a larger structure Gaussian, will be significantly brightened during summation, without any normalization. This effect is shown in Figure 3.11.

However, the normalized formula has some tradeoffs against the simple sum-based formulation. Firstly, the space of normalized solutions has a higher dimensional equivalent class of solutions (synonymously more, at least one proven, additional null directions) when compared to the baseline sum formula. Defining the sum renderer as:

$$I_{\text{sum}}(p) = \sum_{i=1}^{N} g_i(p;\theta_i)c_i.$$

Where $g_i(p;\theta_i) \geq 0$ is the spatial weight of the Gaussian i at the pixel p, $c_i \in \mathbb{R}^3$ represents the colour coefficient. The normalized renderer can be written as:

$$I_{\text{norm}}(p) = \frac{\sum_{i=1}^{N} g_i(p;\theta_i)c_i}{\sum_{i=1}^{N} g_i(p;\theta_i) + \epsilon}.$$

If we momentarily ignore ϵ, then consider rescaling all the weights by a factor $\lambda > 0$, then the sum renderer becomes:

$$\tilde{I}_{\text{sum}}(p) = \sum_{i} \lambda g_i(p)c_i = \lambda I_{\text{sum}}(p),$$

This clearly generally changes the image; however in the normalized formulation:

$$\begin{aligned}\tilde{I}_{\text{norm}}(p) &= \frac{\sum_i \lambda g_i(p) c_i}{\sum_i \lambda g_i(p)} \\ &= \frac{\lambda \sum_i g_i(p) c_i}{\lambda \sum_i g_i(p)} \\ &= I_{\text{norm}}(p).\end{aligned}$$

The rescaling of weights does not affect the solution, i.e., there is a continuous scale equivalence in the weights which the additive renderer does not have. Equivalently, defining $F_{\text{norm}}(\Theta)$ as the vectorized normalized render and $\Theta(\lambda)$ as the parameter path from common weight rescaling. Since

$$F_{\text{norm}}(\Theta(\lambda)) = F_{\text{norm}}(\Theta(1)),$$

differentiating with respect to λ at $\lambda = 1$ gives

$$J_{\text{norm}}(\Theta) v = 0, \qquad v = \left.\frac{d\Theta(\lambda)}{d\lambda}\right|_{\lambda=1}.$$

Thus v is in the nullspace of the normalized renderer Jacobian. For a least-squares objective the curvature from Gauss-Newton is roughly $J^\top J$, so this null direction corresponds to a flat or nearly flat direction of the local loss landscape. This suggests that the normalized renderer does not just perform better with the structured initialization and refinement (as shown in PSNR results in Table 3.5), but that the nature of initialization and/or refinement technique may strongly affect the convergence of training. This can be seen in Figure 3.12, where random initializations are attempted with the normalized model, yielding much worse results compared to random sum initializations, and clearly the structured systems.

3.3.3 Sorting Quantization Approaches

The method developed for fast image-codec compression in WebP and PNG final formats avoids encoder training times, however, the quantization approach for actual parameters is not nearly as robust. In fact, the image-parameter codec representation could be uniquely applied to any prior quantization method that is codebook based. Likewise, the results of Table 3.9 show that although fit results are higher, the native quantization more significantly degrades the structured pipelines. The effect is primarily a result of the differing

distributions in scales and feature values. The initialized approaches contain Gaussians that are significant outliers in scale, resulting from the multiscale proposal approach. Although these Gaussians have a good impact in image representation efficiency, they cause minimum and maximum range, and linear quantization distributions to be greatly affected.

As shown in Figure 3.13, the structured pipelines have significant low-total bins in the large regions, especially evident in the major scale distribution. The distinct peaks for large coarse, and medium-large detail features can be seen in the proposed approaches, while the baseline random initializations produce more expected and quantization friendly distributions.

To aid in gaining competitive quantization results, the initial results from Table 3.7 utilized linear quantization; however, later runs utilized nonlinear transformations based on natural logarithm and hyperbolic-arcsin formulations.

Asinh P90. The Asinh P90 transform is utilized for the Gaussian colour coefficients. For a parameter value x, a robust scale s is resolved from the 90th percentile of the collection of values

$$s = P_{90}(|x|)$$

Then the stored value becomes:

$$u = \operatorname{asinh}((\frac{x}{s}))$$

and can be decoded (as $\hat{x}$) from the inverse transform (from the retrieved coefficients $\hat{u}$).

$$\hat{x} = s \sinh(\hat{u})$$

Log1p Med. This transformation is the same idea, using the median scale value:

$$s = \operatorname{median}(x)$$

Then the transformed value, u, is

$$u = \log\left(1 + \frac{x}{s}\right),$$

Where decoding to $\hat{x}$ applies:

$$\hat{x} = s(\exp(\hat{u}) - 1)$$

The results of Table 3.11 are provided to show the benefit of the chosen quantization approaches over linear quantization in terms of RMSE and Max Error Reduction % (between quantized and non quantized parameters).

Table 3.11: Average transform quantization diagnostics for the low codec setting.

System	Parameter	Transform	RMSE	Linear RMSE	RMSE Red. %	Max Error Red. %
GS-Sketch	Feature B	Asinh P90	0.01572	0.02458	36.05	57.36
GS-Sketch	Feature G	Asinh P90	0.01701	0.02624	35.19	78.26
GS-Sketch	Feature R	Asinh P90	0.0168	0.02588	35.08	27.71
GS-Sketch	Scale 1	Log1p Med	0.05247	0.07571	30.69	75.33
GS-Sketch	Scale 2	Log1p Med	0.03417	0.06196	44.86	84.98
GS-Sketch(sum)	Feature B	Asinh P90	0.005558	0.01048	46.95	58.32
GS-Sketch(sum)	Feature G	Asinh P90	0.006037	0.01141	47.07	91.45
GS-Sketch(sum)	Feature R	Asinh P90	0.005918	0.01097	46.05	87.53
GS-Sketch(sum)	Scale 1	Log1p Med	0.05117	0.07284	29.76	74.78
GS-Sketch(sum)	Scale 2	Log1p Med	0.03583	0.06273	42.88	82.47
GI (Baseline)	Feature B	Asinh P90	0.00422	0.01004	57.97	68.09
GI (Baseline)	Feature G	Asinh P90	0.004992	0.01146	56.44	65.20
GI (Baseline)	Feature R	Asinh P90	0.004928	0.01108	55.53	66.87
GI (Baseline)	Scale 1	Log1p Med	0.01944	0.0232	16.19	58.97
GI (Baseline)	Scale 2	Log1p Med	0.01927	0.023	16.20	59.18

3.3.4 Sorting Artifact and Behaviour Analysis

It is seen that either the yx_snake or xy_snake win every sort contest for the GS-Sketch pipeline, likewise, it is also seen that when one sort fails most tend to fail. Table 3.12 reflects the average bpp of each sorting method, aggregated by category. Timing for each sort is relatively similar, while bpp is drastically lower for some sorts for each system. For GS-Sketch the top winners are yx-snake (0.705626 bpp), xy-snake (0.705892 bpp), hue/luma+xy (0.709553 bpp), for GS-Sketch(sum) yx-snake (0.697488 bpp), xy-snake (0.698292 bpp), hue/luma+xy (0.703490 bpp), and for GI (baseline) xy+luma (0.693825 bpp), hue/luma+xy (0.693874 bpp), xy-snake (0.694430 bpp). This suggests that the easiest and most natural sorting method is to group Gaussians in similar spatial positions together, this automatically will group Gaussians in similar image regions, and therefore smooth qualities in the resultant parameter images are enhanced leading to higher bpp reductions overall.

Likewise, as noted in earlier bpp tests and sort contest results some images in the structured pipelines containing artifact errors when sorting. To diagnose this issue, a test is run

Table 3.12: Average bitrate of sorting methods on WebP regime on low settings. Time is direct-sort safety test plus encoding time; for no sort it is encoding time only.

Sort method	GS-Sketch		GS-Sketch(sum)		GI (baseline)	
	bpp	Time (s)	bpp	Time (s)	bpp	Time (s)
no sort	0.772	1.067	0.764	1.004	0.767	0.983
yx-snake	**0.706**	1.079	**0.697**	1.036	0.697	1.043
xy-snake	0.706	1.082	0.698	1.041	**0.694**	1.043
hue/luma+xy	0.710	1.093	0.703	1.066	0.694	1.036
xy+luma	0.710	1.088	0.704	1.046	0.694	1.034
Morton	0.711	1.081	0.704	1.048	0.701	1.023
RGB	0.718	1.038	0.716	1.021	0.723	1.019
PCA	0.729	1.025	0.725	1.009	0.729	0.990
luma	0.733	1.051	0.730	1.021	0.732	0.995
hue/luma	0.759	1.090	0.756	1.055	0.748	1.060

that neglects the compression codec entirely, and simply tests the sorting of Gaussians. Table 3.13 reflects the results of the full sorting regime on the WebP low-bitrate setting, while Figure 3.14 below reflects examples of failed cases. **Note that sorting errors are defined by PSNR drops between the sorted and unsorted images before saving, irrespective of codec functionality and thus only row sorting artifacts are revealed.** It can be seen that tile like artifacts result when sorting the structured representations, while the random representation may promote redundancy and be more resilient to anisotropic conditions, causing no failed cases. The potential bpp reduction is also shown. It can be seen that although the baseline random initializations provide a generally slightly lower bitrate, the majority of the bpp reduction between GI and the proposed methods comes from sorting failures (rejecting otherwise compressive sorts for renderer-safety) in the codec. As seen in Table 3.13 much of the potential benefits of the sorting formulation is not realized in the structured pipelines, especially in the sum-based initialization. The gap between realized bpp for GS-Sketch and baseline (0.039bpp), is reduced to 0.012 bpp in the ideal case where all sorts are valid. The sum variant fails the most permutation test before sorting occurs, rejecting most sorts. This is likely due to tile rendering limits as discussed in Appendix Section A.2. If these cases were realized the gaps between the baseline and structured pipeline methods due to differing distributions would be more adequately addressed. The visual effect of sorting/tiling artifacts are displayed in Figure 3.14.

Table 3.13: Renderer-safe sorting counts and bitrate bounds on Kodak N03000 using WebP. Successful and failed sorts count direct no-restore sort attempts across all images and candidate orderings.

Method	Success	Failure	Actual bpp	No-fail bpp	All-fail bpp
GS-Sketch	135	81	0.731	0.704	0.772
GS-Sketch(sum)	81	135	0.739	0.697	0.764
GI (baseline)	216	0	0.692	0.692	0.767

The ideal solution is ensuring the trained model is sorting safe, by preordering Gaussians before/during training, or isolating the tiling artifacts in rendering. Furthermore, if sorting safety is enforced elsewhere in the pipeline significant time reductions can be made, as referenced by the results of the winning sorts in Table 3.12, compared to the total times of WebP sort contests in Table 3.10, the use of sorting safety enforcement during training and intelligent sorts based on only the top spatial sorting winners can increase both bpp reduction and time savings. Times can be significantly reduced (estimated from per sort results) to under 5-10s for WebP, and similar reductions to roughly one-half to one-quarter of the original time in PNG, greatly improving the proportional speed, and overall bpp reductions due to safe sorting in the methods.

3.3.5 Refinement with Densification and Pruning

It is relevant to justify the use of the proposed refinement process, and highlight the value of this contribution. Firstly, Figure 3.15 reflects the value at various iterations of densification and pruning. Due to the complexities of the normalized rendering formula outlined in Section 3.3.2, optimization can even degrade fit performance metrics when sparsity artifacts begin to be introduced. The two ablation experiments on kodim01 illustrate this effect. Two separate experiments, allowing for 0.2 growth, and a no-growth schedule are tested for comparison. Methods that allow for densification and pruning refinement achieve higher overall PSNR scores as shown in Table 3.14 and without growth in Table 3.15. It can be seen that, especially at the lower budget regimes, densification and pruning refinement yields better results, both visually and numerically in terms of hole artifacts and degradation of quality.

At smaller Gaussian counts, the differences in PSNR values are observed to diverge as training iterations increase, due to unstable training of the more complex normalized rendering formula, and sparsity interactions. Overall, results are improved by using the suggested

Table 3.14: Training snapshots for the practical densification ablation. The refinement run uses prune/train/densify with growth enabled.

Case	Iter.	No-ref. PSNR	Ref. PSNR	ΔPSNR	No-ref. MS-SSIM	Ref. MS-SSIM	No-ref. time	Ref. time
N1000	5000	15.80	23.00	7.19	0.6422	0.8008	20.6	26.1
N1000	10000	14.07	20.15	6.07	0.5796	0.7388	41.1	52.2
N1000	15000	13.29	23.81	10.53	0.5411	0.8362	61.7	78.3
N1000	20000	12.46	24.03	11.57	0.5007	0.8545	82.2	104.4
N1000	25000	12.10	23.67	11.57	0.4821	0.8398	102.8	130.5
N1000	30000	11.84	24.20	12.36	0.4666	0.8742	123.3	156.6
N7000	5000	27.24	27.85	0.61	0.9470	0.9555	33.7	39.0
N7000	10000	27.15	28.41	1.26	0.9504	0.9628	67.4	78.0
N7000	15000	26.14	28.57	2.42	0.9500	0.9644	101.1	116.9
N7000	20000	25.53	28.59	3.06	0.9500	0.9647	134.8	155.9
N7000	25000	25.21	28.59	3.37	0.9494	0.9646	168.5	194.9
N7000	30000	24.73	28.75	4.02	0.9488	0.9667	202.2	233.9

Table 3.15: Training snapshots for the fixed-budget densification ablation. The refinement run uses prune/train/densify with zero net growth.

Case	Iter.	No-ref. PSNR	Ref. PSNR	ΔPSNR	No-ref. MS-SSIM	Ref. MS-SSIM	No-ref. time	Ref. time
N1000	5000	15.80	22.52	6.71	0.6422	0.7881	20.6	24.8
N1000	10000	14.07	19.75	5.67	0.5796	0.7119	41.1	49.7
N1000	15000	13.29	23.62	10.34	0.5411	0.8255	61.7	74.5
N1000	20000	12.46	23.65	11.18	0.5007	0.8385	82.2	99.3
N1000	25000	12.10	23.74	11.64	0.4821	0.8352	102.8	124.2
N1000	30000	11.84	22.60	10.76	0.4666	0.8311	123.3	149.0
N7000	5000	27.24	27.44	0.20	0.9470	0.9458	33.7	34.5
N7000	10000	27.15	27.78	0.64	0.9504	0.9525	67.4	69.0
N7000	15000	26.14	27.72	1.58	0.9500	0.9505	101.1	103.4
N7000	20000	25.53	27.88	2.35	0.9500	0.9541	134.8	137.9
N7000	25000	25.21	27.90	2.68	0.9494	0.9540	168.5	172.4
N7000	30000	24.73	27.71	2.97	0.9488	0.9544	202.2	206.9

refinement process, especially at lower Gaussian counts. The refinement process deals with sparsity artifacts, and continuously reallocates Gaussians into more optimal regions based on residual scores throughout pruning and densification.

3.3.6 Practical Relevance of Gaussian Representations

This section outlines some short demonstrations related to practical application of Gaussians as image representations. **These are not meant to be interpreted as full projects.** Gaussians have been used as latent spaces in diffusion tasks [7], show significant super-resolution capability [15] due to the resolution invariance of Gaussian representations (i.e., they can be rendered at any resolution), and provide an interpretable, explicit, and editable primitive. To elaborate, Gaussian primitives allow for local manipulation, and also

transformative manipulation of Gaussians. For example, if we take each Gaussian feature vector:

$$\mathbf{c}_j = \begin{bmatrix} r_j & g_j & b_j \end{bmatrix}^T.$$

Since stored coefficients are in an asinh type transform, we can first invert it:

$$\mathbf{c}_j = \mathbf{s} \odot \sinh(\mathbf{u}_j),$$

Given $\mathbf{u}_j$, the stored feature vector and $\mathbf{s}$ the per-channel scale, after editing, the modified colour is mapped back as

$$\mathbf{u}'_j = \operatorname{asinh}\left(\frac{\mathbf{c}'_j}{\mathbf{s}}\right).$$

Then global edits can be applied directly to the Gaussian colour vectors through simple matrix transforms. For example for cold colour enhancement:

$$\mathbf{c}'_j = M_{\text{cool}}\mathbf{c}_j, \qquad M_{\text{cool}} = \begin{bmatrix} 0.86 & 0 & 0 \\ 0 & 1.02 & 0.02 \\ 0 & 0.08 & 1.22 \end{bmatrix}.$$

For warm:

$$\mathbf{c}'_j = M_{\text{warm}}\mathbf{c}_j, \qquad M_{\text{warm}} = \begin{bmatrix} 1.18 & 0.05 & 0 \\ 0.02 & 1.03 & 0 \\ 0 & 0 & 0.82 \end{bmatrix}.$$

Gaussian scales can also be manipulated for interesting effects, for example blurring, these can be directly multiplied. Furthermore, in the structured pipelines, major and minor scales are meaningful due to disk-pin splits. If edge structure is to be identified, we can simply use heuristic cues like primitive scale statistics. Let each Gaussian have minor and major scale

$$\sigma_{\min,j} = \min(\sigma_{1,j}, \sigma_{2,j}), \qquad \sigma_{\max,j} = \max(\sigma_{1,j}, \sigma_{2,j}).$$

An approximate edge-primitive set can then be selected using scale quantiles:

$$\mathcal{E} = \{j : \sigma_{\min,j} \leq Q_{\tau_{\min}}(\sigma_{\min}) \quad \text{and} \quad \sigma_{\max,j} \leq Q_{\tau_{\max}}(\sigma_{\max})\}\,.$$

Here $Q_\tau(\cdot)$ denotes the empirical τ-quantile of the corresponding Gaussian scale distribution, while τ_{min} and τ_{max} denote minor and major scale selection thresholds respectively. Modulating major scales to take around the bottom half, and minor scales around the bottom quarter generally yields edge structures. The effects of these transformations can be seen in Figure 3.16. Section 3.3.6 reflects a small demo of the Gaussian latent space.

Proof-of-Concept Gaussian latent vector demonstration

A potential application of fast and structured Gaussian proposal is the aspect of treating the Gaussians as a latent vector. In recent diffusion work, wavelets have been used as a structured transform-domain representation to speed up and improve accuracy of diffusion methods [35]. However, Gaussians can be simultaneously viewed as a probabilistic distribution of image structure. Therefore, they may equivalently be used as a latent representation space for images. To illustrate this a simple diffusion study is done by training an MLP in a simple single image proof-of-concept diffusion process. First, the Gaussian parameters and coefficients are packed to a latent vector z_0, which is the normalized latent space configuring orientation, scale, position, and intensity of the primitives. We corrupt the latent with Gaussian noise across T=50 timesteps, which produces noisy latent samples at each timestep. Thereafter, a simple MLP is trained on the single-image latent dataset to predict the clean latent from the noisy one.

$$z_t = \sqrt{\bar{\alpha}_t}\, z_0 + \sqrt{1-\bar{\alpha}_t}\, \epsilon, \qquad \epsilon \sim \mathcal{N}(0, I),$$

In the reverse process, the noisy latent is iteratively updated by the learned denoiser, using an aggressive update rule that pulls a generated noisy sample towards the predicted clean sample at each step.

$$z_{t-1} = (1-\lambda_t) z_t + \lambda_t\, \hat{z}_0^{(t)} + \sigma_t \xi_t, \qquad \xi_t \sim \mathcal{N}(0, I),$$

where z_0 is the original latent vector, z_t the noisy latent at t, $\bar{\alpha}_t$ denotes the cumulative noise-retention coefficient, and $\epsilon \sim \mathcal{N}(0, I)$ denotes standard Gaussian noise. During the reverse, z_{t-1} denotes the next step update, $\hat{z}_0^{(t)}$ is the denoiser prediction of the clean model from z_t, λ_t is the blending weight, σ_t is the noise scale, $\xi_t \sim \mathcal{N}(0, I)$ denotes Gaussian perturbation, and T is the total number of diffusion steps. λ_t is defined as:

$$\lambda_t = 0.45 + 0.50 \left(1 - \frac{t-1}{T}\right)$$

The results of the diffusion process can be seen in Figures 3.17 and 3.18 using 8629 Gaussians and noise steps T=50. The recovered latent is seen to have PSNR 21.8 dB, and 0.0066 MSE to the fit. **This demonstration is of a simple one-image MLP case, so broad claims cannot be made; however, with reviewed literature in the use of 3D Gaussians [47], and specifically Gaussian image models [7] as diffusion latents, this section serves to provide a visual demonstration of the Gaussian latent representation.** This is relevant because the Gaussians are essentially probabilistic primitives that represent the statistical structure of an image. Further work looks towards synergistic latent representations in fields like medical image processing, where inherent noise in images from physical processes are often modeled with (although usually other underlying distributions dictate the spread, like shot-noise) probabilistic distributions.

3.3.7 Established Image Compression Regimes

The main benefit of the presented methodology is not the outright replacement of pure image compression regimes. Methods like JPEG [19], and WebP [12] generally outperform Gaussian representation methods on non-super resolution tasks, and especially on natural images [46]. In such super resolution tasks, bpp ranges are often much lower for relevant results, where as they are much more stringent for lower resolutions like the Kodak dataset [11]. This is a primary reason why [46] simply saves all parameter values at 16 bit, as Gaussian counts at 10000 and higher are still relevant for 2k images yielding under 0.4 bpp [46]. **The particular framing of this work is to take the more difficult case for this primitive representation, for natural lower resolution images, and improve the GI baseline at the lower Gaussian counts and bpp ranges for these cases, that other methods struggle with [46].**

As such, the presented GS-Sketch prototype results at the WebP medium setting achieves 30 dB PSNR at an average Kodak bpp of approximately 0.8, which occurs before the presented data in [45] for the GI baseline (at around 1 bpp according to their rate-distortion curves).

The experimental results suggest that the medium WebP setting has the best tradeoff in performance and performs the best on the GS-Sketch proposed variant. Thus, the range between the listed N1000 and N7000 regimes as shown in Table 3.3, is used to form rate distortion curves. The fit results at various Gaussian ranges in the thousands-scale show consistently higher PSNR for the proposed initialized methods. Comparison against JPEG at matched bpp ranges is shown in Figure 3.19. It is shown that although JPEG still

maintains its sizeable $\tilde{2}$ PSNR lead in these low-medium resolution natural image cases, the edge PSNR is significantly higher for the GS-Sketch method compared to JPEG, indicating more accurate reconstruction of Sobel-gradient responses. Likewise, the MS-SSIM also shows that the method begins to catch up at higher Gaussian counts.

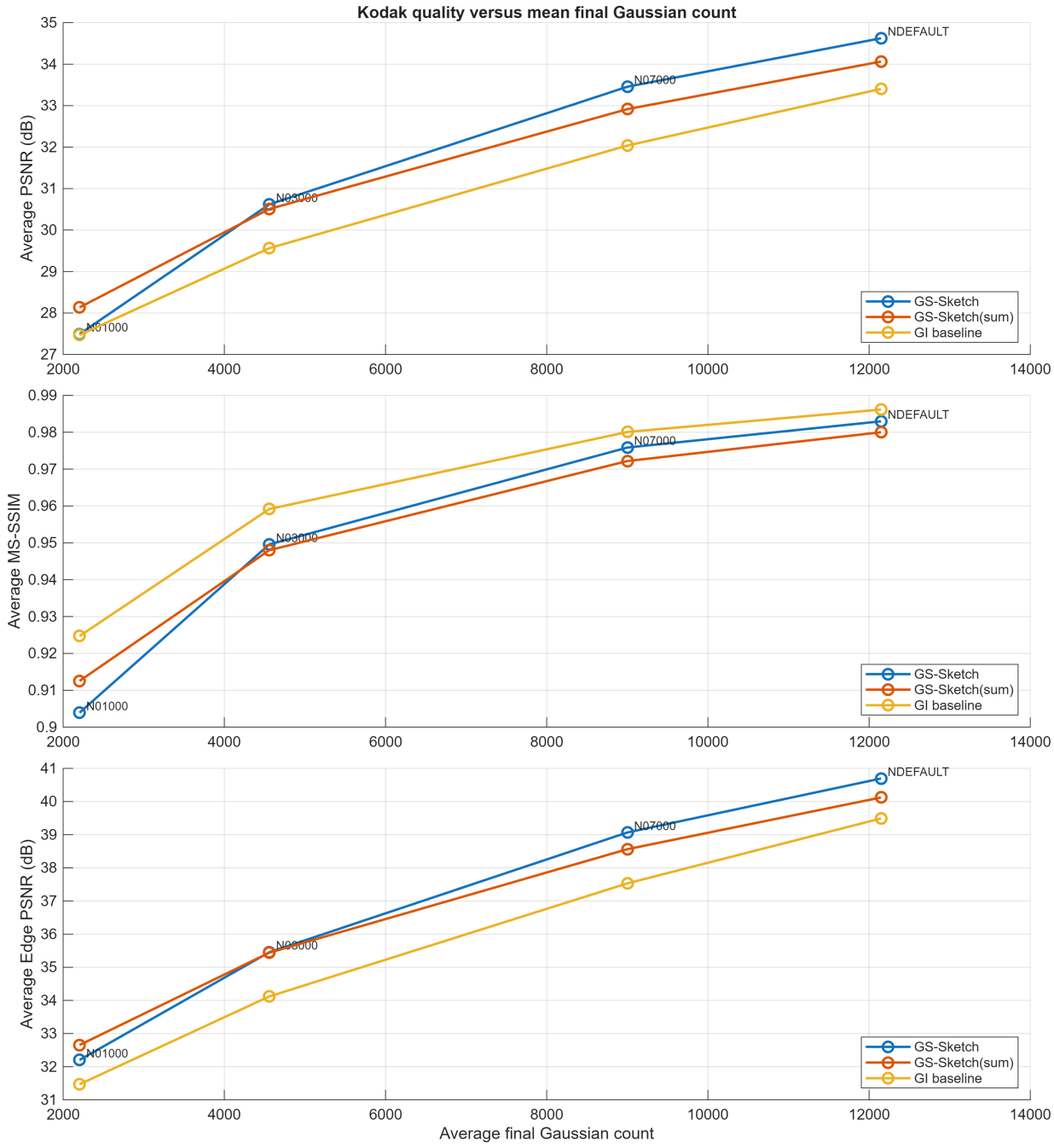


Figure 3.6: Budget-Quality Curves for proposed GS-Sketch methods and baseline GI.

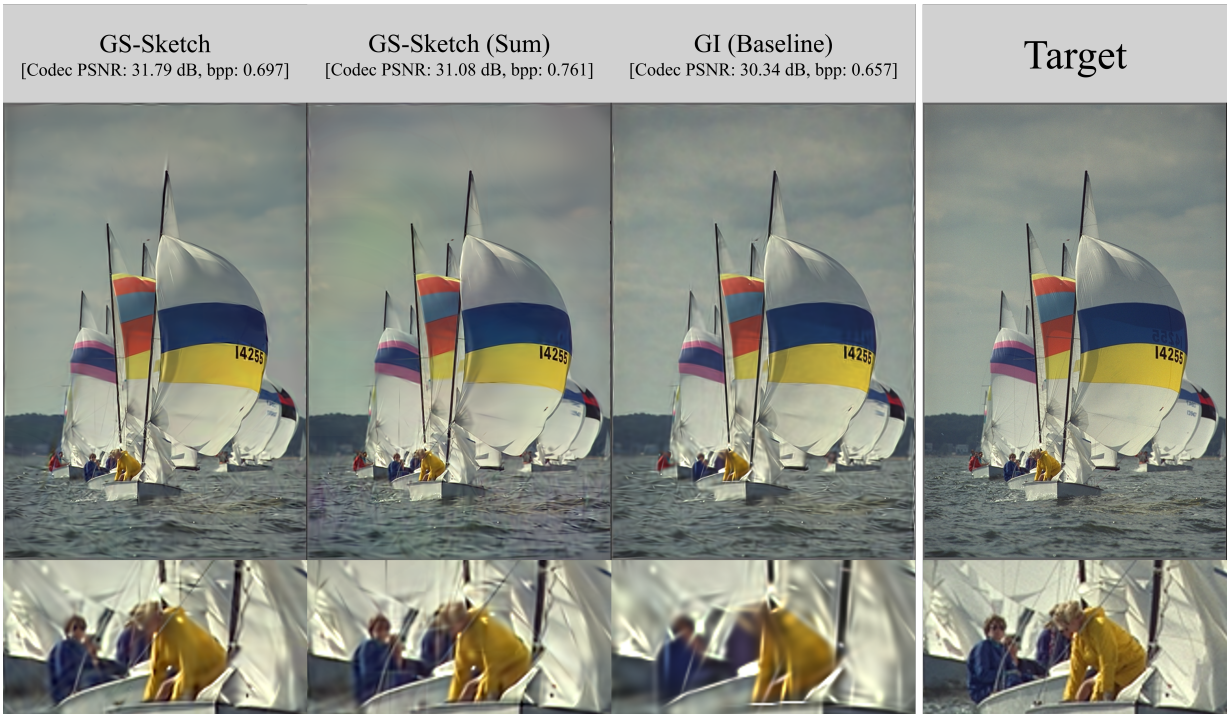


Figure 3.7: Post-compression codec renders for kodim09.png, showing our method retains features in visually impactful regions of the image, after compression. Bpp benchmarks show that the method can be competitive with learned-codebook approaches, while providing less PSNR drop against the implemented native quantization approach.

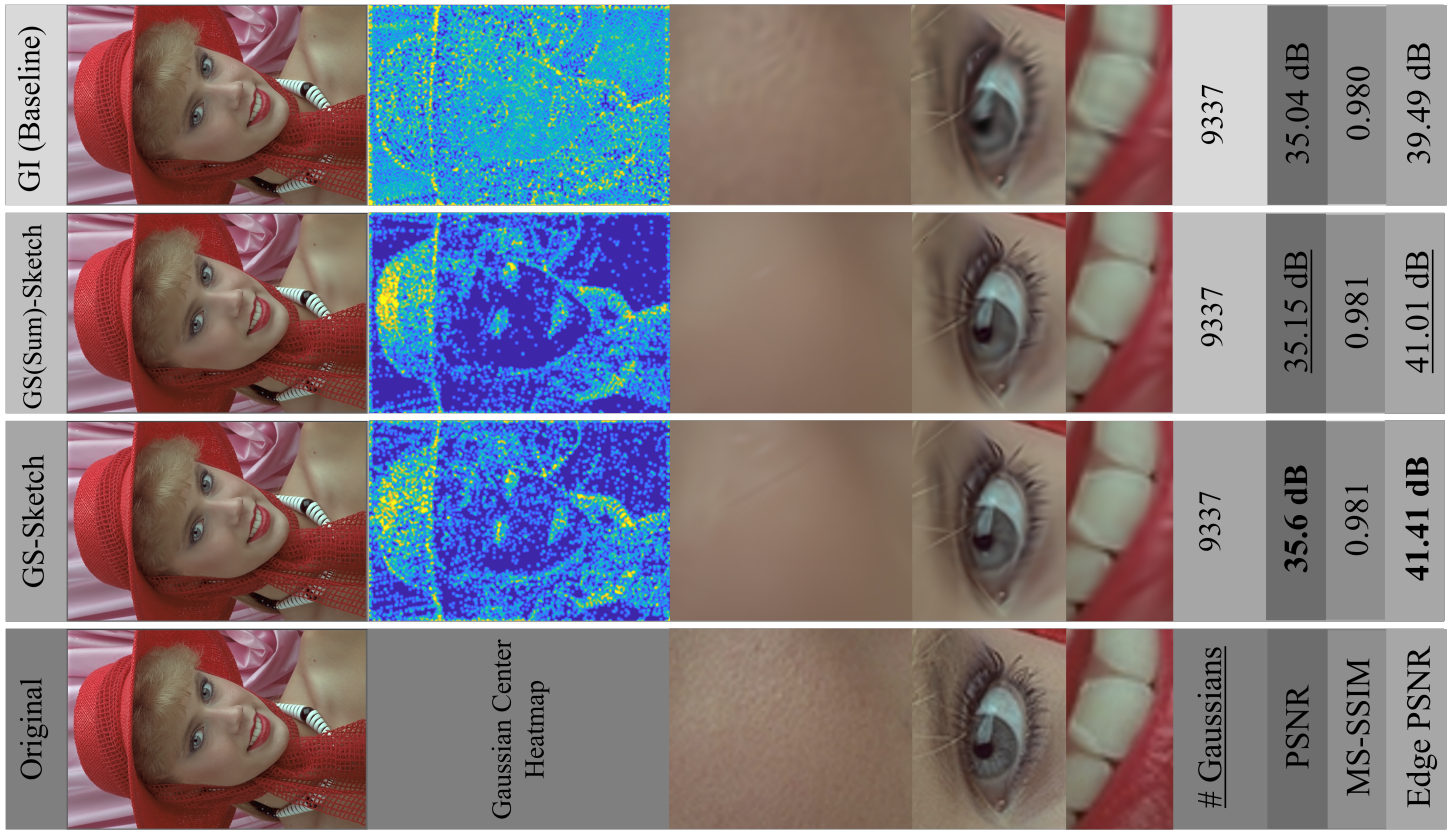


Figure 3.8: Visual tradeoff in subtle texture and visual sharpness of prominent features in the human case of Kodim04.png, where our models achieve slightly lower MS-SSIM, particularly due to skin-texture representation as large smooth Gaussians, or uniform small Gaussians in random initializations. This shows our method trades smoothness in skin-like textures for sharpness in other facial feature aspects.

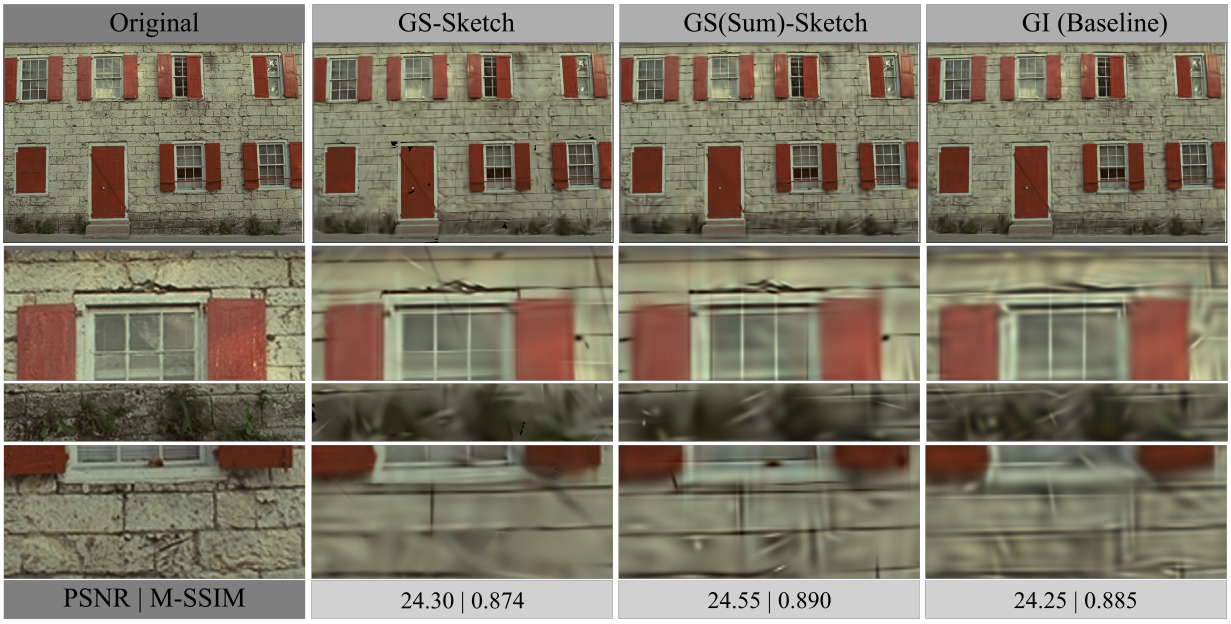


Figure 3.9: Texture tradeoff in low/stringent Gaussian counts, sparsity may result in black artifacts from lack of Gaussians in certain regions, however small textures such as the window pane features may be better represented. Random initializations generally have more Gaussians to provide more uniform texture coverage.

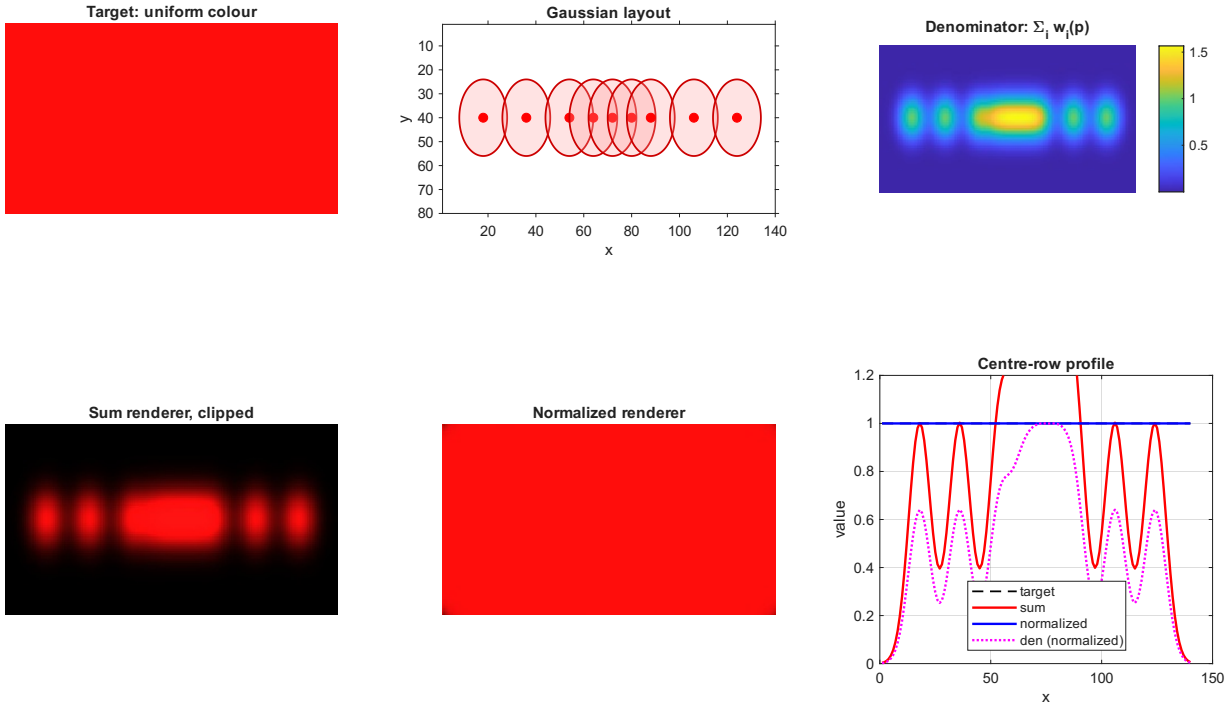


Figure 3.10: Benefit of normalized rendering in representation of uniform colour with sparse Gaussians. Note that in the actual rendering implementation, Gaussians have a maximum radius of effect, so they appear as ellipsoids for normalized rendering, but still can represent smooth colour in their regions.

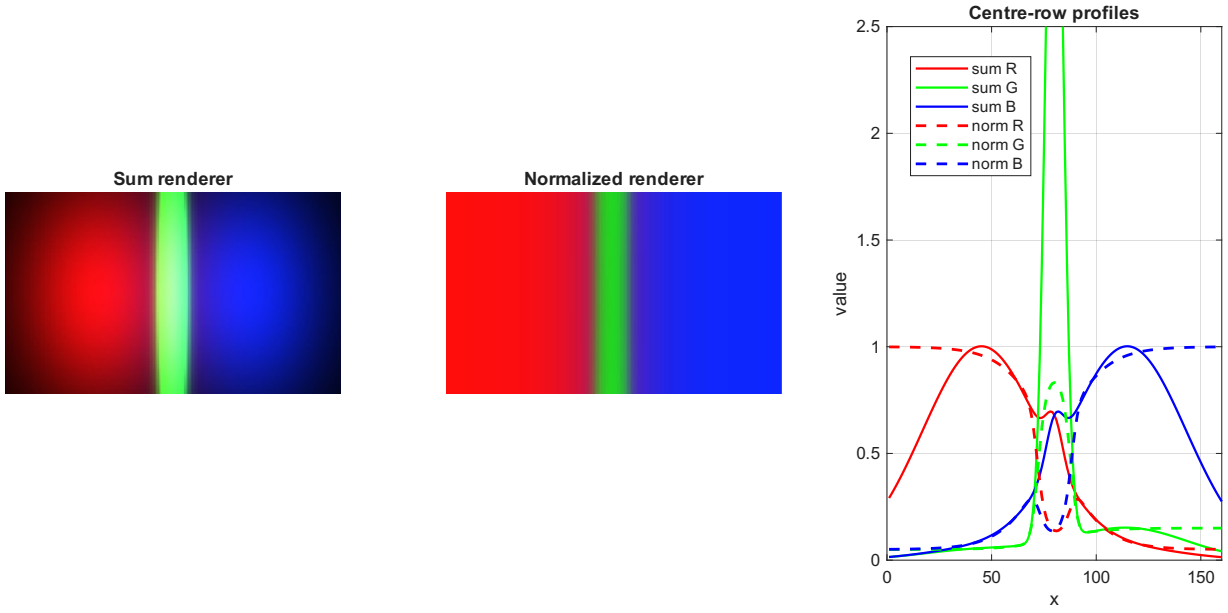


Figure 3.11: Benefit of normalized rendering in representation of edges along with large structured Gaussians, where summation causes colour brightening, normalization allows for a smooth transition to the intended brightness without complexity.

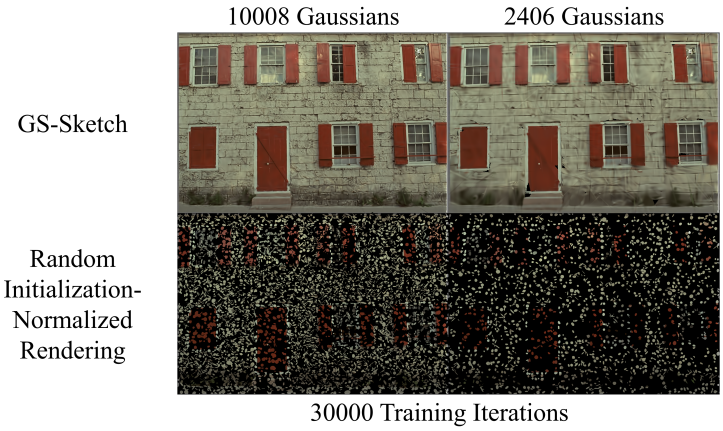


Figure 3.12: Training for 30000 iterations with random initializations, achieving final PSNR <12 dB for random initializations. This illustrates the convergence has a strong dependence on initialization.

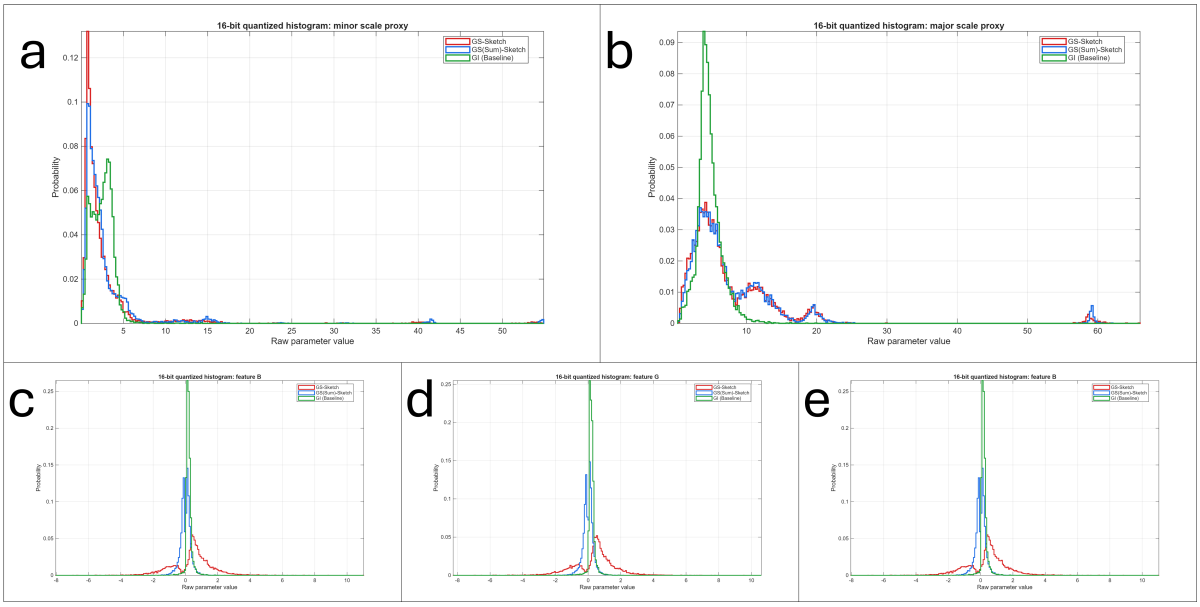


Figure 3.13: Distribution of parameter values in a 16-bit linear quantized histogram for (a) minor scales (b) major scales, (c) (d) (e) R, G, B feature channels of GS-Sketch, GS-Sketch(Sum) and GI (baseline).

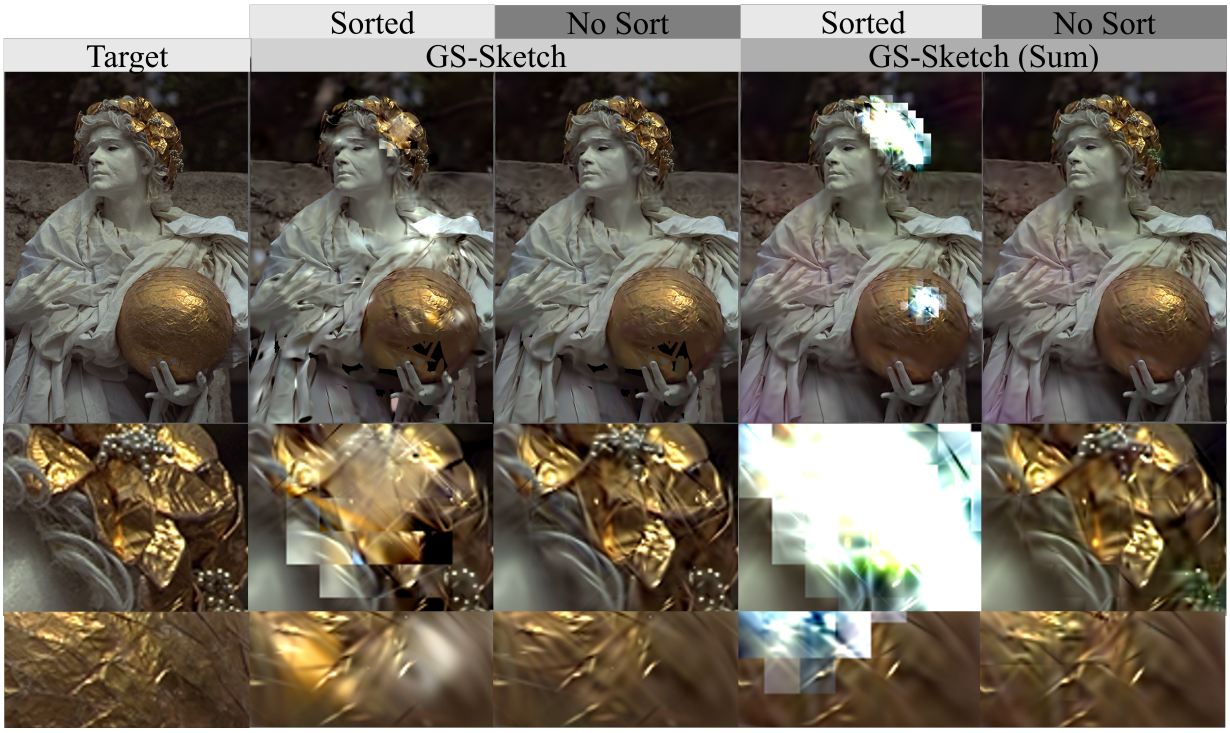


Figure 3.14: Sorting artifacts in kodim17.png at 4633 Gaussians.



Figure 3.15: Visualization of training effect of densification, with and without growth, on kodim01, using 2005 (2406 for growth allotted) Gaussians. Final and peak PSNR values are displayed. This illustrates the value of refinement in dealing with black artifacts common with normalized rendering in sparse scenarios. (Final fit values for 30000 iterations, while visuals for up to 20000 iterations are displayed).



Figure 3.16: Visualization of structure identification, colour transformations, and tied scale-structural edits. Edge brightening increases the identified edge Gaussians colour coefficients, Non-Edge Blur increases scales of the Gaussians not identified as edges, while Edge Blur blurs the identified edge Gaussians only.



Figure 3.17: Denoising (backward) steps with training MLP from initialization on fruits for 50 denoising steps on a randomized start

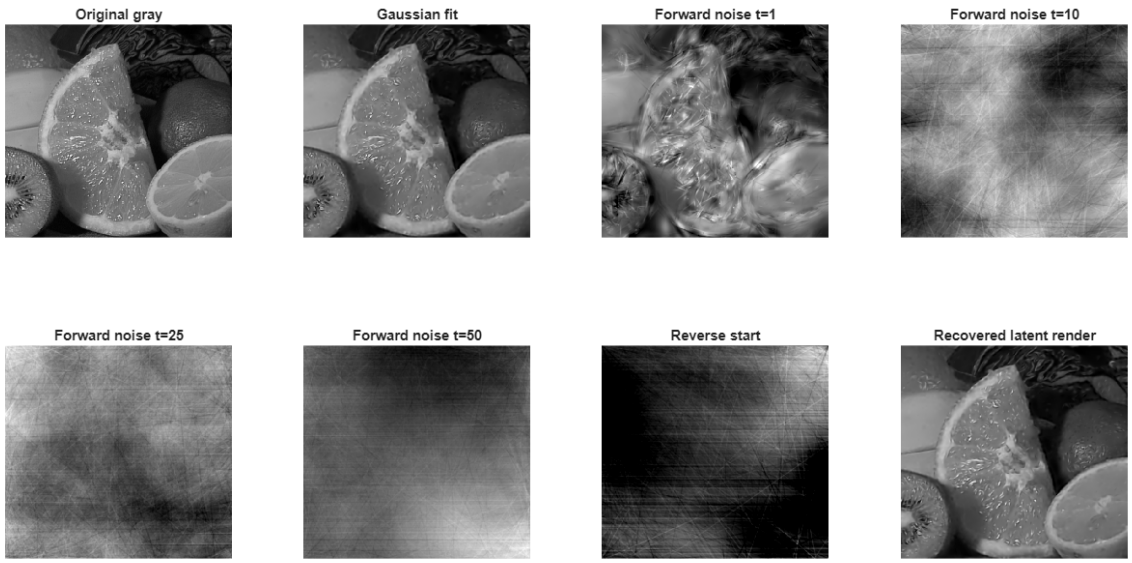


Figure 3.18: Visualization of forward steps, original image, reverse start, and output image

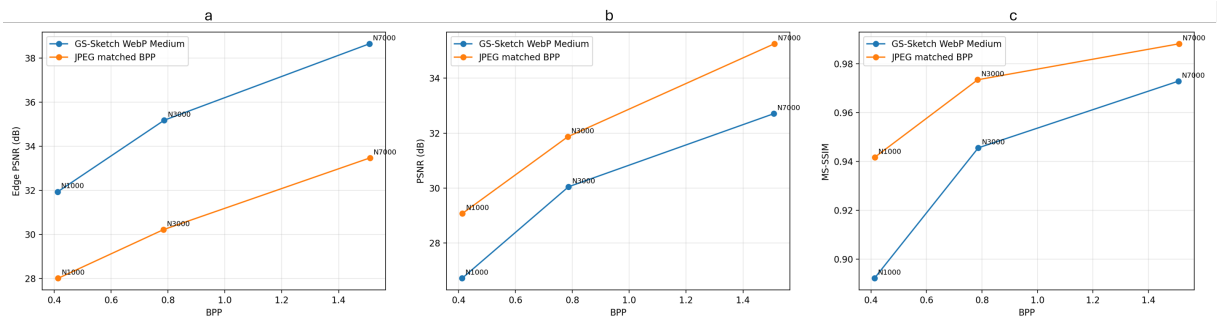


Figure 3.19: Rate distortion plots for the presented image metrics (c) MS-SSIM [40], (a) Edge-PSNR defined in Section 3.1.6, and (b) PSNR [18]. Taken at the regimes from N1000 to N7000, as defined in Table 3.3

Chapter 4

Conclusions

This chapter outlines the summary of the contributions of this work, then outlines the future work, limitations, and implementation improvements planned before publication.

4.1 Summary of Method and Results

This work has outlined the GS-Sketch method, that proposes a structured Gaussian image representation. GS-Sketch is implemented by proposing geometry using image structural elements (coherence, Laplacian, edge behaviours), locking it, and treating the problem as a linear convex optimization problem, optimizing coefficients with an ADMM implementation. This allows Gaussian optimization in the Gaussian Image[45] library, utilizing gsplat [43]. These measures allow for structured initializations and refinement to carry prior information into high-frequency regions.

4.1.1 Primary Contributions

Multi-scale sketch-inspired Gaussian proposal. I design a novel proposal method based on the sketching process using large, medium, fine stages, and targeting specific image qualities like the image levels at lower resolutions. The method utilizes residual, edge, coherence, and local scale maps to place Gaussians. Likewise, I propose a heuristic split between disk-like Gaussians in smoother regions, and pin-like Gaussians in edge/detail regions.

Fixed-geometry ADMM coefficient optimization. Once all geometric parameters of Gaussians are fixed, the image formulation can be written as Eq (3.2) and Eq (3.1). This allows for a linear system when all geometry is fixed, and a convex optimization problem that can be formulated into ADMM updates through Eq 3.3, and the dual variable updates. The main novelty is proposing a completely heuristic geometry proposal, and separating it entirely from coefficient solving.

Densification, pruning, and hole mitigation. Several practical additions, in terms of densification and pruning refinement during training as detailed in Section 3.1.4 and hole patching as detailed in Section 3.1.3. These mitigate common artifacts noticed in the normalized rendering formula, particularly black artifacts in starved/low Gaussian regimes. The densification and pruning refinement serves to continually redistribute Gaussians to residual regions during training.

Adaptive sorting contest for parameter-image compression. Gaussian parameters have their final representation packed into image-like arrays and compressed with common image compression architectures, particularly PNG [41] and WebP [12]. A contest is conducted between several discrete intuitive sorts, where generally spatial coherence places similar Gaussian parameters in smooth arrays and therefore improves compression. This method replaces costly first-time training of learned vector quantization approaches.

4.1.2 Primary Results

Improved low-count structured Gaussian pipeline. Structured initializations and refinement pipelines generally improve edge and high-frequency structures compared to random initializations and no refinement, especially in coherent edge regions where pins align to the image structure. In general across all regimes, the proposed methods outperform random initialization in PSNR [18] and edge PSNR (defined in Section 3.1.6), while trailing slightly behind in multi-scale texture scores (MS-SSIM [40]) due to lack of coverage.

Competitive WebP medium result. The WebP medium final compression regime with the GS-Sketch pipeline reaches roughly 30 dB PSNR at around 0.8 bpp, ahead of the reported rate distortion curve in Gaussian Image [45], where the baseline GaussianImage-RS reaches 30dB around 1 bpp, although the baseline does not fully recreate the original random draws, this result contextualizes performance increases at higher bpps.

Better than reproduced GI baseline in most direct-codec comparisons. In the outlined experiments, aside from the native learned codebook pipeline, where GaussianImage is generally stronger due to better distributions as shown in Figure 3.13, both the fit and codec result for the structured initialization and refinement methods (for the WebP medium regimes) beat the GaussianImage [45] baseline in terms of PSNR and Edge-PSNR.

Learned quantization is stronger but may not provide practical encode times. Although the learned codebook provides a faster post-training encode time, the total time when including the full training of the codebook for *every* image in the pipeline, the discrete WebP and PNG saving methods show their advantage. These methods reduce total encode times when comparing total proposed codec stage to native quantization, yielding approximately 10–100× lower (depending on PNG or WebP) first-time runtime. Note that this result concerns only the codec-stage compression (quantization contest).

Normalized rendering is superior in most structured pipeline regimes. Although GS-Sketch (Sum) performed better at the lowest Gaussian counts, for the rest of the range GS-Sketch outperforms both methods (including baseline [45]) as evident from Table 3.4. However, it struggles at lower Gaussian counts due to starved coverage. Although it produces generally smoother colour palettes, and higher edge fidelity, lower Gaussian counts can produce black artifacts in sparse areas, and general sparsity artifacts in the result as shown in Figure 3.9.

Lastly, a large benefit to the method is proportional time savings when considering crossing fit iterations. Natural structure is also increased, but raw quality can be seen to overtake the random baseline at relatively early training iterations. A summary figure is created for the experimented pipelines based on the retrieved results, a timing summary figure is created in Figure 4.1. The time reduction and fit quality increase illustrate the benefits of the method, which does not achieve comparable bpp at the lowest ranges, but offers a middle ground for improved quality at slightly higher bpp ranges(as aforementioned meeting the 30dB PSNR mark before the GI baseline [45]).

4.1.3 Conceptual Conclusions

Structured initialization and refinement are useful and provide differing results. Structured initializations and refinement allow for geometry and placement to be

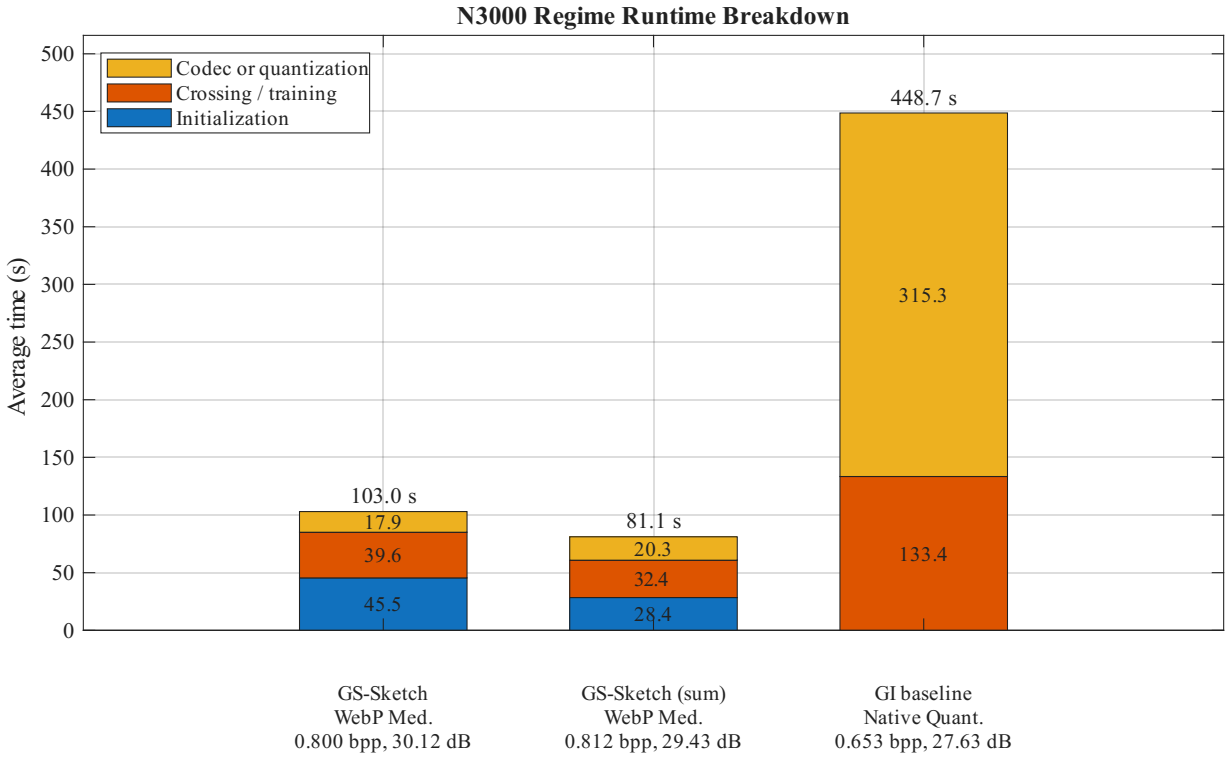


Figure 4.1: Comparison between timing of the proposed GS-Sketch and GS-Sketch(Sum) variants compared to the original GaussianImage-RS [45] pipeline at N3000 (refer to Table 3.3) on the Kodak dataset [11]. Encoding time for baseline considers the per-image codebook learning time. Crossing times are taken from proportional training times of crossing iterations relative to final iterations. PSNR and bpp listed.

suggested as prior information to the Gaussian optimization process, and iteratively redistributed during training. The results of this work provide evidence that different initializations and refinement strategies do not optimize to the same result, and that the solution space for Gaussians is initialization sensitive, especially under the normalized rendering formula detailed in Section 3.3.2. Image structure and heuristic intuition can be used to provide useful initializations for centers, scales, and rotations of Gaussians.

Sharpness-Coverage Tradeoff. At low Gaussian counts, the tradeoff between coverage and sharpness of adaptive allocation techniques are amplified. The general results show that although the structured initializations and refinement produce higher PSNR and edge consistency, they represent subtle texture and irregular regions worse than random initializations. This suggests structured initializations and refinement should have coverage-oriented/random components to find an optimal balance.

Normalized rendering changes the solution space. Normalized rendering does not just change the visual smoothness of Gaussian primitives. The denominator interactions create a more complex solution space that requires good initialization and refinement qualities to function properly under the experiment regime.

Gaussian representations are not only compression. At low-mid resolution natural image ranges, Gaussian representations, including the one proposed here, are not generally better [45] than mature codecs like WebP [12], or JPEG [19]. The main benefit is a conjunction between representations remaining editable, interpretable, and quickly rendered. Specifically providing rendering speeds that are in magnitudes 100-1000 fps+ [45].

Closing the codec gap is still worthwhile. Although established compression regimes remain stronger, improving Gaussian compression, especially in the difficult case of medium-low resolution natural images, is beneficial. It can be seen that the performance of the method when compared to JPEG [19] is still inferior, if research improvements continue, then the closing of this gap will make explicit representations more practical. JPEG remains superior at matched bitrate for PSNR and MS-SSIM metrics. This would be beneficial in terms of interactive representations (as Gaussian representations allow primitive-level editing) geometric structure, interpretability, latent-space representations, and compatibility with Gaussian rendering pipelines.

Best future regimes may be higher-resolution or task-specific. Part of the reason Kodak benchmarks at reasonable bpp with these methods is difficult is because super-

resolution is a much more efficient use of the representation. At higher-resolutions, the cost of a single Gaussian in terms of bpp is significantly reduced, which is why many flagship Gaussian image methods like Image-GS [46] that target high resolution datasets do not require any further compression/quantization approach. Conversely, reducing the cost and performance degradation for smaller resolution and natural images is beneficial towards making the model universal.

4.2 Further Work

The results suggest several directions for the improvement of the work and its performance, while others are directions for the use of Gaussian image representations or extensions in practical applications.

High-resolution/ImageGS [46] benchmarking. High resolution regimes are an obvious extension, given the suitability of Gaussian representations and success of other structured methods [46] in high resolution or super resolution tasks [49, 15, 4]. A key extension of this project, pending improvement of the ADMM implementation as referred to in Section 4.2.2, is to benchmark against such super resolution targets, particularly in non-natural images.

Applications using Gaussian primitive representations. Currently, a single-latent experiment was conducted to provide discussion material for the use of Gaussians in latent spaces; however, other work has already begun to apply Gaussian image representations for real dataset diffusion approaches [7]. Likewise, there are potential applications in fields where images are collected from sensor data, like medical image processing, where Gaussian/statistical primitives may provide valuable representations for images.

3D Gaussian prior densification. Previous pipelines I have worked on regarding Gaussian splatting have also targeted the initialization; however, in 3D [6, 26]. A useful extension of this project would be to utilize depth maps and camera views from COLMAP [34, 33], and provide a rich initialization using Gaussians-Image representations, projected back in 3D space using the COLMAP anchor points, and depth estimates from a network such as MiDaS [31]. Such densification procedures are particularly helpful when heuristics are applied to conform to expected practical applications. This allows trained Gaussians to be encoded with intelligent prior information suited to downstream tasks.

Image-codec storage of learned codebooks. An alluring extension is to replace our linear/simple quantization approaches before image saving with the learned vector codebook approach [45]. Saving resultant vector quantized values as images may produce savings with minimal metadata.

Hybrid Gaussian–wavelet compression. As seen in results, a primary weakness of the structured methods is the difficult representation of low-amplitude texture regions (waves, subtle skin features, wall textures). Gaussian allocation counts, especially in starved regimes have a difficulty in mapping such features. Wavelet compression [35, 23, 32]; however, maps well to repetitive patterns and textures, and could be joined in a hybrid form to identify and address different areas with different methods.

Canonical Gaussian compression architectures. In 3D, Gaussians still maintain a hefty computational burden over other latent representations. In Gaussian diffusion experiments, other means of formulating the working latent space, particularly autoencoders [47] are used. A latent space for Gaussians in these cases may be useful, especially when canonical (i.e indices or positions of values in vectors have some level of consistent meaning in the space). Autoencoders accomplish this goal internally; however, normal Gaussian representations are sort invariant. Using the same final-image representation method, a wavelet transform can be performed on Gaussian splats before saving in the suggested image codec WebP or PNG forms. Wavelet transforms for these pseudo-canonical properties have shown great success in improving diffusion models in images [35]. This allowed for a discrete and canonical transform for latent spaces, and allows for diffusion approaches in 3D to be reduced to 2D problems. The same approach may be viable for Gaussians in 2D, but requires development and testing.

4.2.1 Limitations

Although results show that GS-Sketch provides a useful direction in improving quality and speed at low-medium ranges in natural images, there are still important limitations that should be discussed. These do not necessarily invalidate the method, but describe the current boundary of the prototype, and where further work is relevant.

Comparison to established image codecs. GS-Sketch is not meant to be a replacement for JPEG/WebP/etc., nor are most Gaussian formats [45, 44] that target natural images. Although high-resolution compression [46, 49], and super resolution tasks show some success [15, 4], especially with normalized rendering formulae and

structured initializations [46], this work specifically targets the more difficult case for these representations which are natural images like the Kodak dataset [11]. Benchmarks show at these lower Gaussian counts, where other methods are not typically aimed or studied, our method outperforms the baseline GI [45]. A comparison with JPEG [19] can be found in Section 3.3.7.

CPU-based MATLAB solving. The proposal and ADMM coefficient solving are currently a MATLAB/CPU implementation, which can be greatly optimized from this prototype stage. This can be slower for higher Gaussian counts and higher resolution images. Although the current scope of this project, on the natural Kodak dataset [11], allowed for the current implementation decisions, improvements should be made before larger scale evaluation. These improvements are detailed in Section 4.2.2. Such improvements are necessary before comparison with super-resolution pipelines as detailed in 4.2.

Sparse coverage in normalized rendering. Rendering with normalized formulae can produce black artifacts from sparsity effects in under-defined regions. Hole patching and refinement during training (the proposed densification and pruning process) mitigate this as seen in Figure 3.15; however, small artifact regions still remain as seen in Figure 3.9.

Heuristic Gaussian proposal. Structures for Gaussians are ultimately proposed from residual and image transformation maps. Although this creates interpretable and useful representations, missed structures especially in subtle texture regions (which are seen to greatly affect perceptual quality metrics like MS-SSIM) can cause imperfect estimations that carry forward. Likewise, there is limited correction of geometry elements after proposal before final optimization. There are arguments for a joint optimization phase to replace the ADMM fitting phase, or be modulated with it at every low-medium-high scale.

Low-count capacity limits. Low count sparsity greatly affects the pipeline, especially the proposed normalized rendering variant. The method has a tradeoff in sharpness and coverage as established in Section 3.3.1. Some artifacts are prevalent due to capacity/sparsity limitations, and not just optimizer failures.

Codec metadata overhead. Reported bpp includes parameter images and overhead information, a self contained decoder requires the transforms, shapes, and packing data. A more optimized binary format may be able to reduce this overhead.

Preliminary latent-vector and interpretability demonstrations. The examples of editing and latent representation support the practical relevance, especially when put in conjunction with other work in the field [7, 47], that has utilized Gaussians for diffusion like tasks. However, larger datasets and stronger denoisers are relevant for a full experiment, likewise, full breadth of the usefulness of interactivity would require its own project.

Limited baseline reproduction ablations. The present evaluation measures dataset-level variation using the 24 Kodak natural images. This is used in place of repeated-seed uncertainty for each given image. The baseline was fixed with a deterministic seed for controlled comparison, mimicking the seeded-random initialization style of GaussianImage [45]. The original GaussianImage [45] work and related reproductions do not consider multi-seed uncertainty analyses in their scopes [46]. In practice, random GaussianImage initializations use broad uniform position coverage, so different seeds produce similar global coverage statistics, but local placements may still affect individual images. Full multi-seed variance studies for methods are left to future work. Likewise, the use of refinement with random initializations in the sum category may be relevant for further work.

The limitations suggest that this compression method is not an outright image-codec replacement, but drives a fine balance between representation interpretability, editability, explicit representation, and compression. The method is most usefully framed as a step towards improving interpretable Gaussian-based image representations for use in interactive, or learning-based methods, especially improved towards natural images and medium-low resolution. Future work should then focus on the implementation improvements in terms of ADMM solver speed, enforcing sort safety upstream in the pipeline, or replacement of the ADMM optimization stages with faster CUDA optimization of Splat coefficients. Metadata and sorting decisions can also be improved. These are discussed in further detail in Section 4.2.2.

4.2.2 Implementation Improvements

This section outlines implementation improvements that are achievable in the scope of this project before publication.

MATLAB-to-Python/GPU Acceleration. The results for timing the MATLAB implementation may be improved with conversion to python/GPU acceleration, or a

port to CUDA optimization with fast image rendering. In the CUDA rendering case, for example with the gsplat [43] library, the ADMM formulation would be obsolete because of Eq 3.3, containing linear combinations of the elements of B, which would not be accessible if we rasterized images using the Gaussian optimization CUDA process.

ImageGS-Based Implementation and Benchmarking. As discussed in the introductory chapters, a popular research direction where Gaussian representations succeed is high resolution/super-resolution representations. To this end, and benchmark against the state of the art in this niche, the implementation should be moved into the ImageGS library, allowing for direct comparison, and utilization of some optimizations for normalized rendering [46].

Sorting Enforcement and Stability. Sorting enforcement should be addressed, either through the renderer swap, or improvement of stability by pre-sorting before optimization, and/or during refinement. Other implementation approaches to reduce sorting errors are to limit the amount of Gaussians per tile, as discussed in Appendix Section A.2.

Normalized Denominator Ablation. For stringent cases, the denominator listed in Eq 3.1 may change behaviour if the denominator was $+\epsilon$ versus clamping it. The prior implementation is used in the python port for training, which is slightly faster, but ablation should be done to see whether noticeable differences occur, and the sensitivity to ϵ. The two forms are essentially equivalent when Gaussian coverage is well above ϵ, which is the case for the evaluated set. Likewise, ablation for allowing negative coefficients is beneficial. No evaluated set has more than 1% relative denominator difference (per pixel) between the clamped form $\max(G, \epsilon)$ and the additive form $G + \epsilon$. Thus although not algebraically identical, their difference is numerically negligible, nonetheless, they should be made consistent in the future published pipeline.

Reduction of Sorting Regimes for Quantization. Significant time savings can be had from reducing the number of sorts (and clearly removing the intensive sort safety check), as previously discussed in Section 3.3.4. The results suggest that the strongest sorts, simple spatial ordering with yx and xy snakes, are the pivotal sorts to try for the contests. As such, further implementations should take the simple contest between these two sorts, along with ensuring sort safety prior to the call. This results in shortened time with evidence of this found in the reruns for JPEG comparison in

Appendix Table A.1, where time reduction between previous runs is observed for the WebP format, when using only xy and yx snake sorting.

References

[1] Johannes Ball'e, Valero Laparra, and Eero P. Simoncelli. End-to-end optimized image compression. In *International Conference on Learning Representations*, 2017.

[2] Johannes Ball'e, David Minnen, Saurabh Singh, Sung Jin Hwang, and Nick Johnston. Variational image compression with a scale hyperprior. In *International Conference on Learning Representations*, 2018.

[3] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, 3(1):1–122, 2011.

[4] Du Chen, Liyi Chen, Zhengqiang Zhang, and Lei Zhang. Generalized and efficient 2d gaussian splatting for arbitrary-scale super-resolution. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2025.

[5] Zhengxue Cheng, Heming Sun, Masaru Takeuchi, and Jiro Katto. Learned image compression with discretized gaussian mixture likelihoods and attention modules. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7939–7948, 2020.

[6] SoumyaDeep Chowdhury, Misbaudeen Aderemi Adesanya, Noushad Ahamed Chittoor Mohammed, Kanwarpartap Singh Gill, and Kuljeet Singh Grewal. Video-driven gaussian splatting for as-built building geometry with energy simulation. *Automation in Construction*, 188:106981, 2026.

[7] Simon Coessens, Arijit Samal, Akash Malhotra, and Nacéra Bennacer Seghouani. Gaussiandiffusion: Learning image generation process in gaussian representation space. HAL preprint, 2025. HAL Id: hal-05522830.

[8] L. Peter Deutsch. Deflate compressed data format specification version 1.3. Request for Comments RFC 1951, Internet Engineering Task Force, May 1996.

[9] Emilien Dupont, Adam Goliński, Milad Alizadeh, Yee Whye Teh, and Arnaud Doucet. Coin: Compression with implicit neural representations, 2021.

[10] Emilien Dupont, Hrushikesh Loya, Milad Alizadeh, Adam Goliński, Yee Whye Teh, and Arnaud Doucet. COIN++: Neural compression across modalities. *Transactions on Machine Learning Research*, 2022.

[11] Eastman Kodak Company. Kodak lossless true color image suite. https://r0k.us/graphics/kodak/, 1993. PhotoCD PCD0992; 24 lossless true-colour images, accessed July 20, 2026.

[12] Google. Webp compression techniques. https://developers.google.com/speed/webp/docs/compression, 2025. Accessed: 2026-07-14.

[13] Dailan He, Ziming Yang, Weikun Peng, Rui Ma, Hongwei Qin, and Yan Wang. ELIC: Efficient learned image compression with unevenly grouped space-channel contextual adaptive coding. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5718–5727, 2022.

[14] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[15] Jintong Hu, Bin Xia, Bin Chen, Wenming Yang, and Lei Zhang. Gaussiansr: High fidelity 2d gaussian splatting for arbitrary-scale image super-resolution, 2024.

[16] Binbin Huang, Zehao Yu, Anpei Chen, Andreas Geiger, and Shenghua Gao. 2d gaussian splatting for geometrically accurate radiance fields. In *SIGGRAPH 2024 Conference Papers*. Association for Computing Machinery, 2024.

[17] Yi-Hua Huang, Yang-Tian Sun, Ziyi Yang, Xiaoyang Lyu, Yan-Pei Cao, and Xiaojuan Qi. SC-GS: Sparse-controlled gaussian splatting for editable dynamic scenes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4220–4230, 2024.

[18] Quan Huynh-Thu and Mohammed Ghanbari. Scope of validity of PSNR in image/video quality assessment. *Electronics Letters*, 44(13):800–801, 2008.

[19] International Telecommunication Union. Information technology—digital compression and coding of continuous-tone still images: Requirements and guidelines, 1992.

[20] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics*, 42(4), 2023.

[21] Théo Ladune, Pierrick Philippe, Félix Henry, Gordon Clare, and Thomas Leguay. COOL-CHIC: Coordinate-based low complexity hierarchical image codec. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 13515–13522, 2023.

[22] Jinming Liu, Heming Sun, and Jiro Katto. Learned image compression with mixed transformer-cnn architectures. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14388–14397, 2023.

[23] Stéphane G. Mallat. A theory for multiresolution signal decomposition: The wavelet representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(7):674–693, July 1989.

[24] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *European Conference on Computer Vision*, 2020.

[25] David Minnen, Johannes Ballé, and George D. Toderici. Joint autoregressive and hierarchical priors for learned image compression. In *Advances in Neural Information Processing Systems*, volume 31, 2018.

[26] Noushad Ahamed Chittoor Mohammed, Misbaudeen Aderemi Adesanya, SoumyaDeep Chowdhury, Sudipta Debnath, Andrew Halliday, Gurjit S. Randhawa, Aitazaz A. Farooque, and Kuljeet Singh Grewal. Drones and community-powered rapid neighborhood energy modeling: Demonstrated in a real-world case study. *Sustainable Cities and Society*, 136:107089, 2026.

[27] Wieland Morgenstern, Florian Barthel, Anna Hilsmann, and Peter Eisert. Compact 3d scene representation via self-organizing gaussian grids. In *Computer Vision – ECCV 2024*, pages 18–34, Cham, 2025. Springer Nature Switzerland.

[28] Guy M. Morton. A computer oriented geodetic data base and a new technique in file sequencing. Technical report, International Business Machines Company, 1966.

[29] Haokai Pang, Heming Zhu, Adam Kortylewski, Christian Theobalt, and Marc Habermann. ASH: Animatable gaussian splats for efficient and photoreal human rendering.

In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1165–1175, 2024.

[30] Minghan Qin, Wanhua Li, Jiawei Zhou, Haoqian Wang, and Hanspeter Pfister. Langsplat: 3d language gaussian splatting. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 20051–20060, 2024.

[31] René Ranftl, Katrin Lasinger, David Hafner, Konrad Schindler, and Vladlen Koltun. Towards robust monocular depth estimation: Mixing datasets for zero-shot cross-dataset transfer. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(3):1623–1637, 2022.

[32] V. J. Rehna and M. K. Jeya Kumar. Wavelet based image coding schemes: A recent survey. *International Journal on Soft Computing*, 3(3):101–118, August 2012.

[33] Johannes Lutz Schönberger and Jan-Michael Frahm. Structure-from-motion revisited. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4104–4113, 2016.

[34] Johannes Lutz Schönberger, Enliang Zheng, Jan-Michael Frahm, and Marc Pollefeys. Pixelwise view selection for unstructured multi-view stereo. In *Computer Vision–ECCV 2016*, volume 9907 of *Lecture Notes in Computer Science*, pages 501–518. Springer, 2016.

[35] Luigi Sigillo, Shengfeng He, and Danilo Comminiello. Latent wavelet diffusion: Enabling 4k image synthesis for free, 2025.

[36] Vincent Sitzmann, Julien N. P. Martel, Alexander W. Bergman, David B. Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. In *Advances in Neural Information Processing Systems*, volume 33, 2020.

[37] Matthew Tancik, Pratul P. Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan T. Barron, and Ren Ng. Fourier features let networks learn high frequency functions in low dimensional domains. In *Advances in Neural Information Processing Systems*, volume 33, 2020.

[38] Joanna Waczyńska, Tomasz Szczepanik, Piotr Borycki, Sławomir Tadeja, Thomas Bohné, and Przemysław Spurek. Mirage: Editable 2d images using gaussian splatting, 2024.

[39] Junjie Wang, Jiemin Fang, Xiaopeng Zhang, Lingxi Xie, and Qi Tian. Gaussianeditor: Editing 3d gaussians delicately with text instructions. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 20902–20911, 2024.

[40] Zhou Wang, Eero P. Simoncelli, and Alan C. Bovik. Multi-scale structural similarity for image quality assessment. In *Proceedings of the 37th Asilomar Conference on Signals, Systems and Computers*, volume 2, pages 1398–1402, 2003.

[41] World Wide Web Consortium. Portable network graphics (png) specification, third edition, June 2025.

[42] Guanjun Wu, Taoran Yi, Jiemin Fang, Lingxi Xie, Xiaopeng Zhang, Wei Wei, Wenyu Liu, Qi Tian, and Xinggang Wang. 4d gaussian splatting for real-time dynamic scene rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 20310–20320, 2024.

[43] Vickie Ye, Ruilong Li, Justin Kerr, Matias Turkulainen, Brent Yi, Zhuoyang Pan, Otto Seiskari, Jianbo Ye, Jeffrey Hu, Matthew Tancik, and Angjoo Kanazawa. gsplat: An open-source library for gaussian splatting. *Journal of Machine Learning Research*, 26(34):1–17, 2025.

[44] Zhaojie Zeng, Yuesong Wang, Chao Yang, Tao Guan, and Lili Ju. Instant gaussianimage: A generalizable and self-adaptive image representation via 2d gaussian splatting. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2025.

[45] Xinjie Zhang, Xingtong Ge, Tongda Xu, Dailan He, Yan Wang, Hongwei Qin, Guo Lu, Jing Geng, and Jun Zhang. Gaussianimage: 1000 fps image representation and compression by 2d gaussian splatting. In *European Conference on Computer Vision*, 2024.

[46] Yunxiang Zhang, Bingxuan Li, Alexandr Kuznetsov, Akshay Jindal, Stavros Diolatzis, Kenneth Chen, Anton Sochenov, Anton Kaplanyan, and Qi Sun. Image-gs: Content-adaptive image representation via 2d gaussians. In *Proceedings of the Special Interest Group on Computer Graphics and Interactive Techniques Conference Papers*, pages 1–11, 2025.

[47] Junsheng Zhou, Weiqi Zhang, and Yu-Shen Liu. Diffgs: Functional gaussian splatting diffusion, 2024.

[48] Shijie Zhou, Haoran Chang, Sicheng Jiang, Zhiwen Fan, Zehao Zhu, Dejia Xu, Pradyumna Chari, Suya You, Zhangyang Wang, and Achuta Kadambi. Feature 3dgs: Supercharging 3d gaussian splatting to enable distilled feature fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 21676–21685, 2024.

[49] Lingting Zhu, Guying Lin, Jinnan Chen, Xinjie Zhang, Zhenchao Jin, Zhao Wang, and Lequan Yu. Large images are gaussians: High-quality large image representation with levels of 2d gaussian splatting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2025.

APPENDICES

Appendix A

Supplementary Material

A.1 Comparison Tables

This section contains various Tables that elaborate on results, that are not pivotal to the body of the paper.

Table A.2 reflects the fit fps of the different formulations. Note that a lower speed is

Table A.1: Matched-bpp comparison between GS-Sketch WebP Medium and JPEG on Kodak. JPEG quality is the average quality setting selected to match the GS-Sketch bitrate.

Budget	Method	BPP	PSNR	Edge PSNR	MS-SSIM	Enc. Time (s)	JPEG Q
N01000	GS-Sketch WebP Medium	0.412	26.72	31.93	0.8921	2.52	–
N01000	JPEG matched BPP	0.414	29.08	28.01	0.9417	–	20.8
N03000	GS-Sketch WebP Medium	0.787	30.05	35.18	0.9455	4.83	–
N03000	JPEG matched BPP	0.784	31.87	30.22	0.9734	–	47.7
N07000	GS-Sketch WebP Medium	1.508	32.71	38.65	0.9729	10.13	–
N07000	JPEG matched BPP	1.510	35.25	33.47	0.9882	–	77.8

expected from the normalization formula essentially calling the renderer twice. Likewise, allocation of per tile densities cause the structured initializations to also have slightly slower rendering times. This is due to more tiles with high Gaussian densities, and higher general peak Gaussian densities in each tile, which translates to the tile-based CUDA rendering.

Table A.2: Fit fps aggregated by system and budget. Values are the mean of the fit fps column across images.

Budget	GS-Sketch	GS-Sketch (sum)	GI (baseline)	Images
N1000	495.2	935.2	1096.5	24
N3000	408.1	769.7	1056.6	24
N7000	356.2	680.7	1045.1	24
Default	334.6	644.7	997.8	24

Table A.3: Per budget PSNR results across the 24 Kodak images. Strict wins are every image that system got the best score of all 3 tested systems, while "beats GI" refers to when a structured system beats the GI baseline.

Budget	System	Mean Gaussians	Mean ± Std. PSNR (dB)	Median PSNR (dB)	Maximum PSNR (dB)	Strict Wins	Beats GI
N01000	GS-Sketch	2203.5	27.49 ± 3.45	28.03	34.58	3	11/24
	GS-Sketch(sum)	2203.5	$\mathbf{28.13 \pm 3.39}$	**28.68**	33.94	**21**	**24/24**
	GI baseline	2203.5	27.47 ± 3.12	28.33	31.94	0	N/A
N03000	GS-Sketch	4557.3	$\mathbf{30.62 \pm 3.68}$	**30.98**	**37.33**	**16**	22/24
	GS-Sketch(sum)	4557.3	30.51 ± 3.59	30.89	36.60	7	**23/24**
	GI baseline	4557.3	29.56 ± 3.17	30.08	34.54	1	N/A
N07000	GS-Sketch	9002.2	$\mathbf{33.46 \pm 3.68}$	**33.87**	**39.09**	**24**	**24/24**
	GS-Sketch(sum)	9002.2	32.92 ± 3.67	33.31	38.52	0	22/24
	GI baseline	9002.2	32.04 ± 3.15	32.29	36.61	0	N/A
Default	GS-Sketch	12148.5	$\mathbf{34.62 \pm 3.55}$	**34.94**	**39.96**	**23**	**24/24**
	GS-Sketch(sum)	12148.5	34.06 ± 3.58	34.06	39.62	1	20/24
	GI baseline	12148.5	33.40 ± 3.19	33.38	38.60	0	N/A

Table A.3 shows that as budgets go up, the normalized method as the budget grows more consistently beats the baseline, while the variability in the structured methods is also higher, they still consistently perform better than the baseline.

A.2 Further Discussion on Sorting Artifacts

Although the reason for sorting artifacts may seem nebulous, the error can be diagnosed. Firstly, refer to Figure A.1 for a useful visual aid. This shows that sorting artifacts occur in regions with high Gaussian density (roughly estimated with occupancy of tiles over 256 Gaussians highlighted in red). This and the tile-like nature of the artifacts on non-spatial related sorting as shown point towards renderer errors. This pattern suggests that there are some built in heuristic limits that are applying a pseudo Top-K method for Gaussian rendering. By looking at the GaussianImage[45] rendering pipeline this decides the rendering procedure. For colour images as tested the gsplat fork library with rasterize_sum_forward

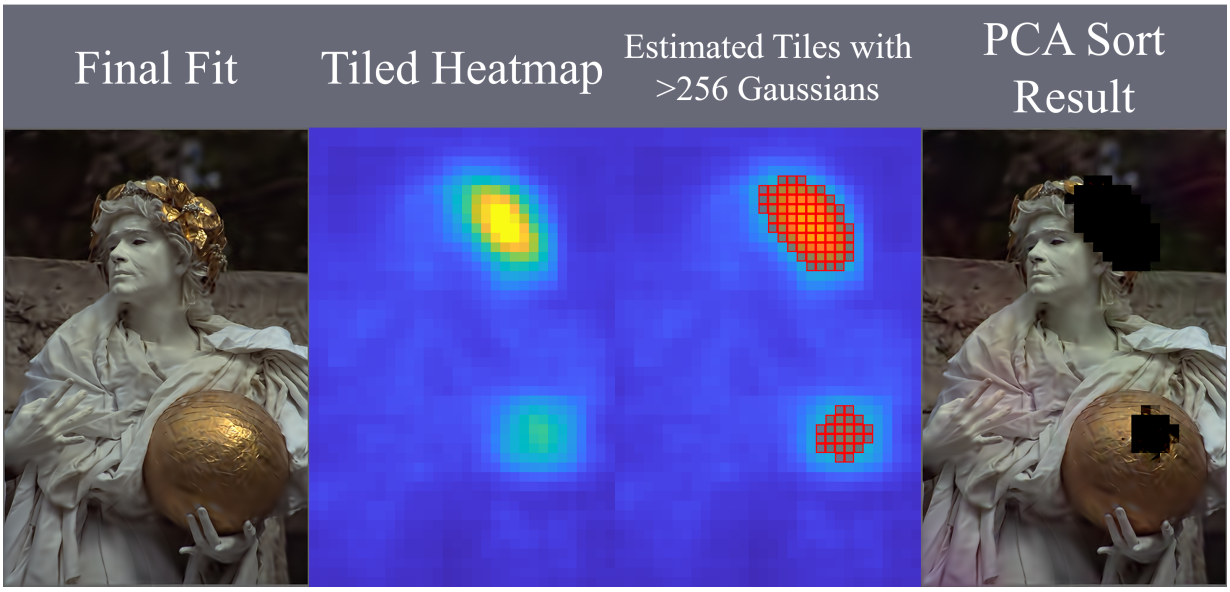


Figure A.1: Pattern of sorting artifacts in high Gaussian count regions, tiled density estimation heatmap shown, and the final artifacts from Feature PCA (non-spatial) sorting. It can be seen that artifacts are specifically in regions with 256+ Gaussians contributing.

in forward.cu is used.

```
if colors.shape[-1] == 3:
    rasterize_fn = _C.rasterize_sum_forward
else:
    rasterize_fn = _C.nd_rasterize_sum_forward
```

Furthermore, in that code it begins with

```
int num_batches = (range.y - range.x + BLOCK_SIZE - 1) / BLOCK_SIZE;

....
for (int b = 0; b < num_batches; ++b) {
```

With 16x16 block sizes (commonly) this would result in each BLOCK_SIZE being 256. Meaning 256 Gaussians are processed per batch in num_batches. If we look to just the end of the loop it says:

```
    done = true;
    }
```

Meaning after the first batch, it simply finishes computation for that tile. This means that the implementation has a large conceptual bottleneck not previously revealed. Either enforcing no more than 256 Gaussians in a tile, or removing this limiter would improve results for compression or quality respectively. As it stands, in such tiles (to where Table 3.13 shows there are many cases in the hot initialized approaches) there are many redundant Gaussians that do not get the opportunity to interact with the quality, but are still training regardless. This also means that technically, the normalization formula is pseudo-topK, as only 256 Gaussians can ever be processed in a tile. Although not intentional, this aligns with other work [46] that incorporates Top-K normalization.

The sorting failures and subsequent study reveal this implementation-level limitation of the current pipeline. Although it was intended that the Gaussian representations are permutation invariant, the inspected renderer processes the Gaussians in tile batches terminated after 256 Gaussians are ran. The headline fit results are not produced using any post-fitting permutations, and instead use the native row order. Sorting is also only accepted when such artifacts are not noticed. Failed sorting attempts are likewise *always* rejected

compression attempts, than ever reported as improvements. Under this renderer, ignored Gaussians that fall above this 256 threshold in a tile represent a bitrate overhead rather than any additional rendering quality, as they are non-contributing to the result. The issue mainly limits the compression efficiency and motivates the further work of mitigation using one of the suggested strategies.

A.3 Relevant Code Snippets

Full code release is pending implementation improvements for consecutive publication, as discussed in Section 4.2.2. Portions are displayed here with original code and casual comments for supplement.

A.3.1 Computing geometry proposals

```
function maps=compute_geom_maps(Y,sigmaSet)
%the inputs are Y the image we are using for proposal,
% and then sigmaSet (see default_cfg) for the details.


h=size(Y,1);
w=size(Y,2);
nSig=numel(sigmaSet);
%we basically make 5 maps here
% one for theta, one for coherence, one for edges
% one for sigmas (gaussian shapes) and one for blobs
edgeStack=zeros(h,w,nSig);
thetaStack=zeros(h,w,nSig);
cohStack=zeros(h,w,nSig);
sigmaStack=zeros(h,w,nSig);
blobStack=zeros(h,w,nSig);

%for each sigma in the sigma set
for k=1:nSig
    sig=sigmaSet(k);
    %first blur the image using that sigma
```

```
Ys=imgaussfilt(Y,sig,'FilterSize',fsz_from_sigma(sig));
%then take the gradients (sobel) magintudes
[gx,gy]=imgradientxy(Ys,'sobel');
gradMag=sqrt(gx.^2+gy.^2);
%then we do the coherence structure forming
Js=max(1,1.35*sig);
J11=imgaussfilt(gx.^2,Js,'FilterSize',fsz_from_sigma(Js));
J22=imgaussfilt(gy.^2,Js,'FilterSize',fsz_from_sigma(Js));
J12=imgaussfilt(gx.*gy,Js,'FilterSize',fsz_from_sigma(Js))
   ;

tmp=sqrt((J11-J22).^2+4*J12.^2);
l1=0.5*(J11+J22+tmp);
l2=0.5*(J11+J22-tmp);
%this l1 and l2 above are eigen value like quantities,
   they are
%basically like the dominant local direction
coh=(l1-l2)./(l1+l2+eps);
%theta is formed using the same J maps
theta=0.5*atan2(2*J12,J11-J22)+pi/2;

% the scale normalized laplacian response here
logResp=abs(sig^2*del2(Ys));
% then we make the edge like areas, where coherence and
   gradient are
% both high and the laplacian
edgeLike=norm01(gradMag).*(0.34+0.66*coh)+0.22*norm01(
   logResp);
edgeLike=norm01(edgeLike);
%then we do the blobs, bnasically in regions where where
   coherence is low
% but laplacian is high
blobLike=norm01(logResp).*(1-coh).^1.08;
blobLike=norm01(blobLike);

edgeStack(:,:,k)=edgeLike;
thetaStack(:,:,k)=theta;
cohStack(:,:,k)=coh;
```

```
    sigmaStack(:,:,k)=sig*ones(h,w);
    blobStack(:,:,k)=blobLike;
end
% so then we find the bestIdx based on the edge stacks, so
   where the
% maximum values occur and which sigma gave the bset response.
    We find the
% winning sigma at each pixel
[~,bestIdx]=max(edgeStack,[],3);

thetaMap=zeros(h,w);
cohMap=zeros(h,w);
sigmaMap=zeros(h,w);
edgeMap=zeros(h,w);
blobMap=zeros(h,w);

% this is like winner takes all after we set it up
% thetaMap comes from the winning scale cohMap comes from the
   winning scale
% sigmaMap is the winning sigma edgeMap is the winning edge
   score
% blobMap is the blob score at that same winning scale
for k=1:nSig
    mk=(bestIdx==k);
    t=thetaStack(:,:,k); thetaMap(mk)=t(mk);
    c=cohStack(:,:,k); cohMap(mk)=c(mk);
    s=sigmaStack(:,:,k); sigmaMap(mk)=s(mk);
    e=edgeStack(:,:,k); edgeMap(mk)=e(mk);
    b=blobStack(:,:,k); blobMap(mk)=b(mk);
end

maps.theta=thetaMap;
maps.coh=cohMap;
maps.sigma=sigmaMap;
maps.edgeLike=edgeMap;
maps.blobLike=blobMap;

end
```

A.3.2 ADMM Application

```
function [x,fit,info]=admm_multispace_stage(B,target,Gaussians
   ,op,prior,idxPrior,semanticMap,xInit)
if nargin<5, prior=[]; end
if nargin<6, idxPrior=[]; end
if nargin<7, semanticMap=[]; end
if nargin<8, xInit=[]; end
[ht,wt]=size(target); y=target(:); n=size(B,2);
%create the gradient maps, the pixel weighting matrix (see
   paper section), and laplacians
W=make_pixel_weights(target,semanticMap,op); w=W(:);
Dx=gradient_matrix_x(ht,wt); Dy=gradient_matrix_y(ht,wt); L2=
   laplacian_matrix_2d(ht,wt);
Gx=Dx*B; Gy=Dy*B; Lp=L2*B;
gxY=Dx*y; gyY=Dy*y; lpY=L2*y;
%here I combine them just cause most are in the form of inner
   products/
%A^TA
BtB=B.'*B; GxTGx=Gx.'*Gx; GyTGy=Gy.'*Gy; LpTLp=Lp.'*Lp;
%this ridge thing is not actually used, but it is an idea to
   keep
%coefficients above zero, a separate ridge term (which turns
   into
%projection in the dual form, but it actually performs better
   if you use
%negative coefficients)
if isfield(op,'useRidge')&&~op.useRidge, gammaVec=zeros(n,1);
   else, gammaVec=make_area_ridge(Gaussians,op); end
regDiag=gammaVec+op.rhoV*ones(n,1);
rhsPrior=zeros(n,1);
if ~isempty(prior)&&~isempty(idxPrior)
    regDiag(idxPrior)=regDiag(idxPrior)+op.eta;
    rhsPrior(idxPrior)=op.eta*prior(:);
end
%this is the formula for A as shown in the paper, it just
   makes it make
%more sense/look cleaner in the below code afterwards.
```

```
A=op.rhoZ*BtB+op.rhoG*(GxTGx+GyTGy)+op.rhoT*LpTLp+spdiags(
   regDiag,0,n,n);
useChol=true;
try, R=chol(A); catch, useChol=false; end
if isempty(xInit)
    x=amps_from_Gaussians(Gaussians);
    if numel(x)~=n, x=0.5*ones(n,1); end
else
    x=xInit(:);
    if numel(x)~=n, x=amps_from_Gaussians(Gaussians); end
end
if isfield(op,'useNonneg')&&op.useNonneg, x=max(0,x); end
%first wwe initialize everything, just so we dont have to
   multiply
%everything oer and over again
Bx=B*x; Gxx=Gx*x; Gyy=Gy*x; Lxx=Lp*x;
z=Bx; gx=Gxx; gy=Gyy; lp=Lxx; v=x;
uz=zeros(size(z)); ugx=zeros(size(gx)); ugy=zeros(size(gy));
   ulp=zeros(size(lp)); uv=zeros(size(v));
objHist=zeros(op.maxIter,1); primalHist=zeros(op.maxIter,1);
   dualHist=zeros(op.maxIter,1);
denZ=op.lambdaPix*(w.^2)+op.rhoZ; denG=op.lambdaGrad+op.rhoG;
   denT=op.lambdaLap+op.rhoT;

%% main loop here for admm
for k=1:op.maxIter
    %initialize the old variables
    %primal variable updates, same equations as paper
    zOld=z; gxOld=gx; gyOld=gy; lpOld=lp; vOld=v;
    rhs=op.rhoZ*(B.'*(z-uz))+op.rhoG*(Gx.'*(gx-ugx)+Gy.'*(gy-
       ugy))+op.rhoT*(Lp.'*(lp-ulp))+op.rhoV*(v-uv)+rhsPrior;
    if useChol, x=R\(R'\rhs); else, x=A\rhs; end
    Bx=B*x; Gxx=Gx*x; Gyy=Gy*x; Lxx=Lp*x;
    z =(op.lambdaPix*(w.^2).*y + op.rhoZ*(Bx+uz))./denZ;
    gx=(op.lambdaGrad*gxY + op.rhoG*(Gxx+ugx))./denG;
    gy=(op.lambdaGrad*gyY + op.rhoG*(Gyy+ugy))./denG;
    lp=(op.lambdaLap*lpY + op.rhoT*(Lxx+ulp))./denT;
    v=x+uv;
```

```
    if isfield(op,'useNonneg')&&op.useNonneg, v=max(0,v); end
    %dual variable update (using the old and new)
    rz=Bx-z; rgx=Gxx-gx; rgy=Gyy-gy; rlp=Lxx-lp; rv=x-v;
    uz=uz+rz; ugx=ugx+rgx; ugy=ugy+rgy; ulp=ulp+rlp; uv=uv+rv;
    sz=op.rhoZ*(z-zOld); sgx=op.rhoG*(gx-gxOld); sgy=op.rhoG*(
       gy-gyOld); slp=op.rhoT*(lp-lpOld); sv=op.rhoV*(v-vOld);
    %these two are for the break clause, based on an epsDual
       and epsPri set
    %at the top
    % their saved for each iteration just for housekeeping, it
        helps to see
    % if something isnt working if the primal his and dual
       hist + the eps
    % is actually going down
    primalHist(k)=sqrt(norm(rz)^2+norm(rgx)^2+norm(rgy)^2+norm
       (rlp)^2+norm(rv)^2);
    dualHist(k)=sqrt(norm(sz)^2+norm(sgx)^2+norm(sgy)^2+norm(
       slp)^2+norm(sv)^2);
    %main objective bookkeeping
    obj=0.5*op.lambdaPix*sum((w.*(Bx-y)).^2)+0.5*op.lambdaGrad
       *(norm(Gxx-gxY)^2+norm(Gyy-gyY)^2)+0.5*op.lambdaLap*
       norm(Lxx-lpY)^2+0.5*sum(gammaVec.*(x.^2));
    if ~isempty(prior)&&~isempty(idxPrior), obj=obj+0.5*op.eta
       *norm(x(idxPrior)-prior(:))^2; end
    objHist(k)=obj;
    epsPri=sqrt(numel(rz)+numel(rgx)+numel(rgy)+numel(rlp)+
       numel(rv))*op.absTol + op.relTol*max([norm(Bx),norm(z),
       norm(Gxx),norm(gx),norm(Gyy),norm(gy),norm(Lxx),norm(lp
       ),norm(x),norm(v)]);
    epsDual=sqrt(numel(sz)+numel(sgx)+numel(sgy)+numel(slp)+
       numel(sv))*op.absTol + op.relTol*sqrt(norm(op.rhoZ*uz)
       ^2+norm(op.rhoG*ugx)^2+norm(op.rhoG*ugy)^2+norm(op.rhoT
       *ulp)^2+norm(op.rhoV*uv)^2);
    if primalHist(k)<=epsPri&&dualHist(k)<=epsDual
        primalHist=primalHist(1:k); dualHist=dualHist(1:k);
           objHist=objHist(1:k); break;
    end
end
```

```
% and then return V and Bx (which is the image rendering)
x=v; fit=struct(); fit.z=reshape(B*x,ht,wt);
info=struct('iter',numel(objHist),'obj',objHist,'primal',
   primalHist,'dual',dualHist,'pixelWeights',reshape(w,ht,wt))
   ;
end
```