INVESTIGATION INTO 2D GAUSSIAN SPLATTING FOR IMAGE REPRESENTATION

Jiacheng Hua [*] [†]
School of Computer and Communication Sciences, EPFL
Lausanne, Switzerland

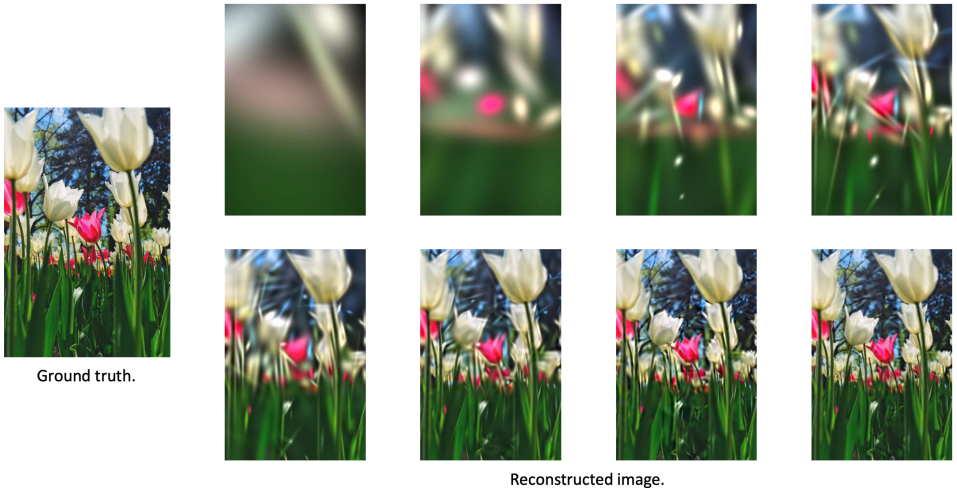


Figure 1: **2D Gaussian splatting for image reconstruction, with progressive addition of Gaussians**: Rather than optimizing the parameters of all 2D Gaussians (position, color, shape) at once, we begin by optimizing a small number of Gaussians that capture the large-scale features of the image. Additional Gaussians are gradually introduced to refine finer details. This coarse-to-fine approach reduces complexity and redundancy, resulting in a more efficient reconstruction process.

ABSTRACT

This work explores the use of Gaussian splatting as a method for representing images as a collection of 2D Gaussians, with a particular focus on balancing image quality and compression. Gaussian splatting, originally developed for representing 3D scenes as collections of 3D Gaussians by minimizing reconstruction loss across multiple photographic viewpoints, can be similarly applied in 2D. By optimizing the positions, colors, and shapes of 2D Gaussians, we aim to minimize reconstruction loss with respect to the target image. This 2D Gaussian splatting approach shares similarities with vectorization techniques like DiffVG and LIVE, which instead optimize the positions and colors of polygons or closed Bezier shapes. However, 2D Gaussians offer the advantage of easier differentiation of pixel values with respect to their positions and colors, resulting in faster optimization. Additionally, 2D Gaussians naturally create smoother color gradients within objects compared to uniformly-colored shapes. We observe a trade-off between compression (fewer Gaussians) and reconstructed image quality. A key finding from our preliminary investigation is that the current implementation uses significantly more 2D Gaussians than necessary. To address this, we introduce

[*] Report of the research project ("semester project") "Exploring the Benefits of 2D Gaussian Splatting in Image Representation and Compression", conducted in the Image and Visual Representation Lab (IVRL), EPFL. Minor revisions were made after Jiacheng returned to his home university, Tsinghua University, China.

[†] Contact: hjc21@mails.tsinghua.edu.cn

an optimization strategy that begins with a small number of Gaussians to capture broad image features, progressively adding more Gaussians to refine finer details. This approach improves the compression-quality trade-off, achieving better reconstruction quality with fewer Gaussians. Furthermore, we address a cropping artifact present in the current implementation and experiment with various reconstruction loss functions.

1 INTRODUCTION

Image representation and compression are fundamental tasks in computer vision and graphics. With the exponential growth of image and video data and the increasing demand for real-time image processing, efficient image compression techniques have become essential. These techniques can significantly reduce storage space requirements, thereby lowering storage costs. Additionally, efficient image representation and compression can reduce processing burdens and improve system response times, making them crucial for almost all visual applications. To address these challenges, new techniques are continually emerging.

One such technique is Scalable Vector Graphics (SVG), which offers scalability and resolution independence. SVG enables high-quality, lightweight, and interactive graphics that integrate seamlessly with web technologies, enhancing both performance and search engine optimization. Both DiffVG (Li et al., 2020) and LIVE (Ma et al., 2022) effectively utilize SVG, using a combination of polygons and closed Bézier curves to represent images. However, there are some limitations. Polygons, with their unnaturally sharp edges, are not common in nature and achieving good image reconstruction quality requires many polygon points. Additionally, the uniform color within each polygon does not reflect the natural color variations in most images, and polygons can be challenging to differentiate.

Another technique is the implicit representation of an image, where a neural network represents an image by taking input coordinates (e.g., x and y) and outputting the corresponding color at those coordinates. An example of this approach is SIREN (Sitzmann et al., 2020). This concept also extends to 3D, where a 3D scene is represented by a neural network, as demonstrated by NeRF (Mildenhall et al., 2020). However, a significant drawback of implicit representation is the complexity of editing. For instance, moving a single point is challenging, and the implicit nature of the representation makes analysis difficult since it relies on a neural network rather than tangible image elements.

Therefore, we explore the possibility of creating a method similar to DiffVG and LIVE that offers scalability and lightweight properties, but with a more natural representation than polygons and without uniform colors. Additionally, we aim to avoid implicit representations to make editing more feasible. A similar solution already exists for 3D scenes, known as 3D Gaussian splatting (Kerbl et al., 2023). Inspired by that, we propose a new technique to represent 2D images by adapting 3D Gaussian splatting to two dimensions, which we call 2D Gaussian splatting.

2 RELATED WORKS

Differentiable vector graphics rasterization for editing and learning. This work (Li et al., 2020) proposes a novel approach to bridging vector graphics and raster image domains through a differentiable rasterizer. This rasterizer leverages raster-based loss functions, optimization procedures, and machine learning techniques to facilitate the editing and generation of vector content. The authors propose two prefiltering options: an analytical prefiltering technique, which is faster but can suffer from artifacts, and a multisampling anti-aliasing technique. This work significantly advances the field by enabling new possibilities for rasterization that are compatible with gradient-based optimization, thereby enhancing the flexibility and capability of vector graphics manipulation in various applications.

Towards layer-wise image vectorization. This work (Ma et al., 2022) introduces a hierarchical optimization approach for image vectorization. The proposed method, LIVE, effectively converts raster images into vector graphics by optimizing the vector graph in a layer-wise manner. This hierarchical process allows for improved fidelity and control in the vectorization process, addressing the challenge of maintaining high quality while ensuring computational efficiency. The paper demon-

Figure 2: An illustration of the optimization process for 2D Gaussian splatting.

strates that the LIVE method outperforms existing techniques in terms of reconstruction accuracy and visual quality, making it a significant contribution to the field of image vectorization.

3D Gaussian splatting for real-time radiance field rendering. This work (Kerbl et al., 2023) presents an innovative approach to real-time rendering using 3D Gaussian splatting. This method involves accumulating color and opacity per pixel from the closest to the farthest Gaussian in a scene, creating a volumetric representation that maintains state-of-the-art quality while enabling real-time performance. The technique addresses the challenges of efficiently rendering complex radiance fields and significantly improves the speed and accuracy of real-time rendering applications.

3 METHODOLOGY

In this section, we will go into details about how our 2D Gaussian splatting works.

The input to our method is a single image, and our goal is to optimize a visual representation that consist with the input (see Fig. 2).

3.1 DEFINITION OF 2D GAUSSIAN

Similar to the methodology of 3D Gaussians (Kerbl et al., 2023), our Gaussians are defined by a full 2D covariance matrix, denoted as Σ, with its center at the point (mean) μ:

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

, where $\Sigma \in \mathbb{R}^{2\times 2}$ and $\mu \in \mathbb{R}^2$. Note that the covariance matrix of Gaussian distribution need to be semi-definite to have physical meaning. It's crucial to note that the covariance matrix of a Gaussian distribution must be semi-definite to hold physical significance. However, direct optimization of the covariance matrix Σ, while seemingly intuitive, poses challenges in maintaining its validity.

To circumvent this issue, we employ the correlation coefficient ρ to deconstruct the covariance matrix Σ:

$$\Sigma = \begin{pmatrix} \sigma_x^2 & \sigma_{xy} \\ \sigma_{xy} & \sigma_y^2 \end{pmatrix} = \begin{pmatrix} \sigma_x^2 & \rho\sigma_x\sigma_y \\ \rho\sigma_x\sigma_y & \sigma_y^2 \end{pmatrix}$$

. This decomposition allows us to optimize σ_x, σ_y, and ρ independently, while ensuring the resulting matrix remains positive semi-definite. The necessity for this approach arises from the fact that the determinant of Σ must satisfy $|\Sigma| = (1-\rho^2)\sigma_x^2\sigma_y^2 \geq 0$, which holds true for $\rho \in [-1, 1]$.

In addition to the parameters derived from the formula of Gaussian distribution, we also introduce a vector to represent color, denoted as $c \in \mathbb{R}^3$, which stores the RGB values of the Gaussian.

The methodology of 3D Gaussians (Kerbl et al., 2023) includes an additional parameter α to indicate the opacity of the Gaussian. However, our work focus on image representation, making the sorting of Gaussians and the implementation of α-blending unnecessary. To simplify, we do not use a specialized variable for opacity, arguing instead that the optimizing the color vector c will effectively manage the opacity of Gaussians well.

Overall, eight parameters are required to determine a Gaussian: the variances $\sigma \in \mathbb{R}^2$, the center coordinate $\mu \in \mathbb{R}^2$, the correlation coefficient $\rho \in \mathbb{R}$, and the color vector $c \in \mathbb{R}^3$.

3.2 Initialization of Gaussians

Since our work uses only a single image as input, we cannot initialize the Gaussians from a set of Structure-from-Motion (SfM) points (Schonberger & Frahm, 2016). Instead, we provide two initialization methods in our implementation:

Random initialization. This is a basic and straightforward approach. We generate Gaussians with parameters uniformly distributed: the center coordinates u are within the height and weight of the image, σ values are within the interval $(0, 1)$, ρ values are within the interval $(-1, 1)$, and the color c is derived from the difference between the current image and the target image at the corresponding coordinates.

Selective initialization. We adopt the initialization method used by LIVE (Li et al., 2020). First, we compute the l_1 pixel-wise loss between the current image and the target image, rejecting differences smaller than a threshold. Next, we quantify all the differences into several bins, with boundaries uniformly distributed between 0 and the maximum difference. We then identify the largest connected component based on this quantization and use its center of mass as the initial coordinate for the next Gaussian. If we need to add K more Gaussians, we select the top K components for the next-stage initialization.The area of a Gaussian is proportional to the square of σ and the kernel size. Denote the area of the connected component as A and the kernel size as s. We initialize σ as:

$$\sigma_{x,y} = \sqrt{\frac{A}{rs^2}}$$

, where r is a constant describing the ratio of the proportion, empirically set to 0.07 in our implementation. The ρ values are always set to 0, and the color c is initialized the same way as in random initialization. If the number of components is less than the number of Gaussians that need to be initialized, the coordinates of the remaining Gaussians will be set using random initialization, as described above.

3.3 Schedule of Adding Gaussians

We provide with several methods for adding Gaussians:

All-at-once. This is a basic and straightforward approach, where we generate the Gaussians all at once at the start.

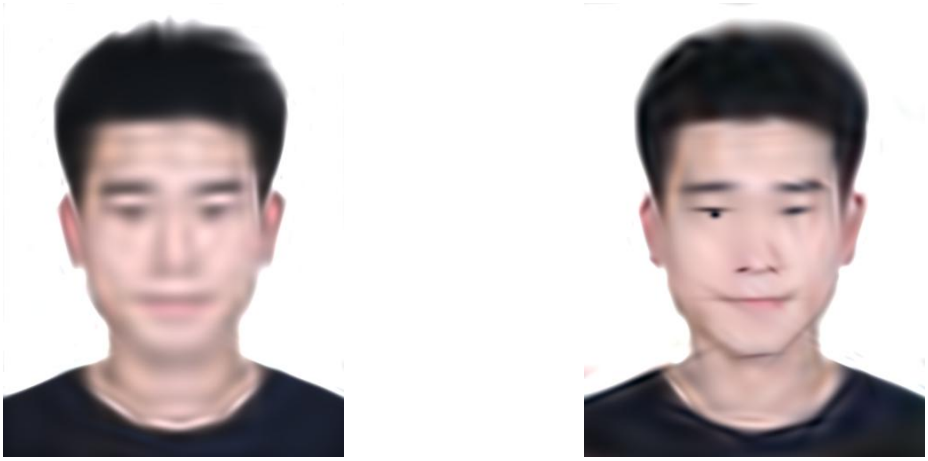
Linear addition. Gaussians are added at a linear rate, meaning a fixed number of Gaussians being introduced after every set number of epochs.

Exponential addition. Gaussians are added at an exponentially increasing rate. In the ith round, the number of newly added Gaussians is set to $min(2^i, m)$, where m is a predetermined upper limit on the number of Gaussians that can be added in a single round. Each round encompasses a fixed number of epochs.

Intuitively, the schedule of adding Gaussians will make a difference, with exponential addition expected to perform the best. Ideally, this method first captures low-frequency features using large, few Gaussians, and subsequently captures the high-frequency feature using small, numerous Gaussians (see Fig. 3). However, since each addition of Gaussians requires epochs for optimization, this method might take more time. Therefore, we still offer the all-at-once and linear addition options as alternatives.

3.4 Gaussian Kernel

To blend all the Gaussians into one image, we utilize a Gaussian kernel. The kernel size is a fixed value set in the training configuration. First, we place a fine grid at the center of the image and draw the Gaussian according to the Gaussian distribution with parameters σ and ρ. Each pixel in the kernel is calculated as follows:



(a) Adding Gaussians all at once.

(b) Adding Gaussians exponentially.

Figure 3: Comparison of methods for adding Gaussians: (a) all at once and (b) exponentially, using a total of 127 Gaussians.

$$c(\mathbf{x}) = \phi(\mathbf{x}) \cdot \mathbf{c} = \frac{1}{2\pi\sqrt{|\Sigma|}} \cdot G(\mathbf{x}) \cdot \mathbf{c}$$

, where $\phi(\mathbf{x})$ is the probability density function of the two-dimensional Gaussian distribution. And then we use a translation transformation to move the Gaussian kernel to the correct position determined by μ.

However, if the kernel size is simply fixed, large σ values can result in a Gaussian that appears cropped (see Fig. 4a). To address this, we apply an adaptive kernel size. When σ exceeds a threshold, empirically set to 2.0 in our implementation, we simultaneously scale the kernel size and σ to maintain the Gaussian's size and prevent it from being cropped (see Fig. 4b).

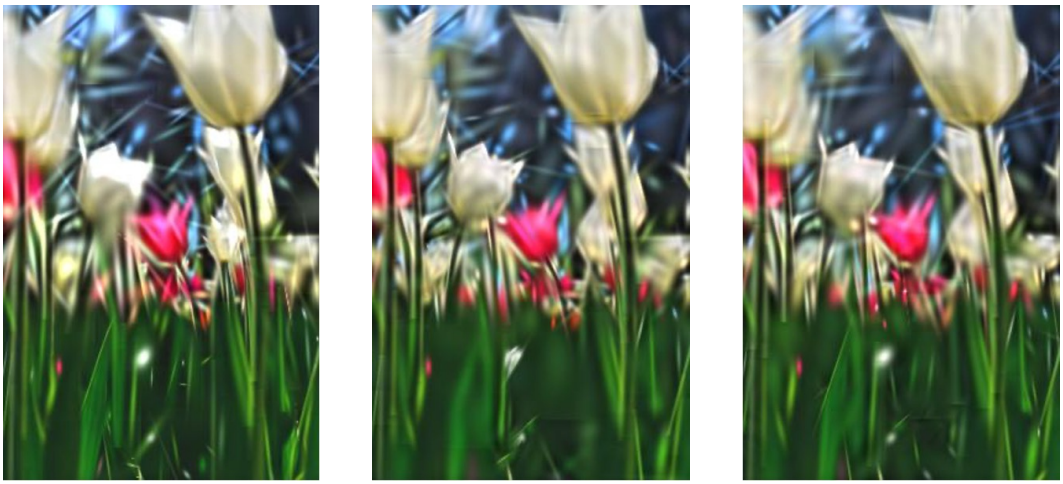
(a) Using a fixed kernel size.

(b) Using an adaptive kernel size.

Figure 4: Comparison of fixed kernel size and adaptive kernel size methods.

3.5 Loss Function

We also experimented with different loss functions in our work (see Fig. 5), and observed that each loss function impacted the results differently. When using a combined loss of MSE and SSIM, the resulting image tended to have sharper lines, although highlights might appear overexposed. With MSE loss alone, the image often appeared blurrier due to the focus on minimizing mean value errors. Combining L1 loss with SSIM produced results that were intermediate, balancing sharpness and blur to some extent. To simplify our approach and better demonstrate the potential of our work, we use MSE loss in other examples.



(a) Combining MSE with SSIM. (b) Using MSE loss only. (c) Combining L1 with SSIM.

Figure 5: Comparison of different loss functions.

4 Conclusion and Limitation

In this work, we propose an efficient method for image representation using 2D Gaussian splatting. Our approach demonstrates impressive performance compared to existing methods, highlighting its substantial potential. Not previously mentioned, we discovered that the images reconstructed using 2D Gaussian splatting can also be exported in SVG format, as utilized by DiffVG and LIVE, allowing us to leverage the benefits of this format. Furthermore, we believe it holds considerable promise for image compression, as suggested by preliminary results from our early experiments. However, due to time constraints, this work lacks extensive experiments and detailed comparisons. To fully assess the capabilities of our method, further experiments and evaluations on additional benchmarks are necessary.

Acknowledgments

This work was conducted during Jiacheng's exchange program at EPFL, as a research project ("semester project") in the Image and Visual Representation Lab (IVRL), under the supervision of Martin Nicolas Everaert and Prof. Sabine Süsstrunk.

References

Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics*, 42(4), July 2023. URL `https://repo-sam.inria.fr/fungraph/3d-gaussian-splatting/`.

Tzu-Mao Li, Michal Lukáč, Gharbi Michaël, and Jonathan Ragan-Kelley. Differentiable vector graphics rasterization for editing and learning. *ACM Trans. Graph. (Proc. SIGGRAPH Asia)*, 39 (6):193:1–193:15, 2020.

Xu Ma, Yuqian Zhou, Xingqian Xu, Bin Sun, Valerii Filev, Nikita Orlov, Yun Fu, and Humphrey Shi. Towards layer-wise image vectorization. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2022.

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 *ECCV*, 2020.

Johannes L. Schonberger and Jan-Michael Frahm. Structure-from-motion revisited. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

Vincent Sitzmann, Julien N.P. Martel, Alexander W. Bergman, David B. Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. In *Proc. NeurIPS*, 2020.