Published in Image Processing On Line on 2025–06–17.
Submitted on 2024–07–19, accepted on 2025–03–11.
ISSN 2105–1232 © 2025 IPOL & the authors CC–BY–NC–SA
This article is available online with supplementary materials, software, datasets and online demo at
https://doi.org/10.5201/ipol.2025.566

Gaussian Splatting: An Introduction

Akash Malhotra[1,2], Nacéra Seghouani[1]

[1]LISN, Paris-Saclay University, Gif-sur-Yvette, France
[2]Amadeus, Sophia-Antipolis, France
{akash.malhotra, nacera.seghouani}@lisn.fr

Communicated by Roger Marí *Demo edited by* Akash Malhotra

Abstract

Gaussian Splatting has emerged as a powerful technique for signal representation, especially in 3D. This paper introduces Gaussian Splatting and demonstrates its application across 1D, 2D, and 3D cases. We also discuss Gaussian Splatting in relation to Neural Radiance Fields (NeRF), highlighting the computational trade-offs and performance benefits. Through this work, we aim to bridge the gap between foundational concepts in view synthesis and advanced research, making Gaussian Splatting a more approachable and widely understood technique in the field of signal processing and computer vision. We provide code examples and detailed explanations to make the topic accessible to a broader audience, enabling readers to dive into more advanced technical papers with ease.

Source Code

The source code and documentation for these algorithms are available from the web page of this article[1]. Usage instructions are included in the `README.md` file of the archive. The original implementations of the methods are available here: 2D Gaussian Splatting[2] and Gaussian Splatting[3].
This is an MLBriefs article. The source code has not been reviewed!

Keywords: Gaussian splatting; NeRF; view synthesis

1 Introduction

Novel view synthesis, the task of generating new viewpoints from existing images of a 3D object or a scene, is a well-explored area in computer vision and graphics. This problem is crucial for applications such as virtual reality, augmented reality, and 3D reconstruction. In this setting, we typically have many images of a 3D object or a scene, and we want to be able to reconstruct novel views from an arbitrary position and orientation. Figure 1 illustrates an example of such a setting

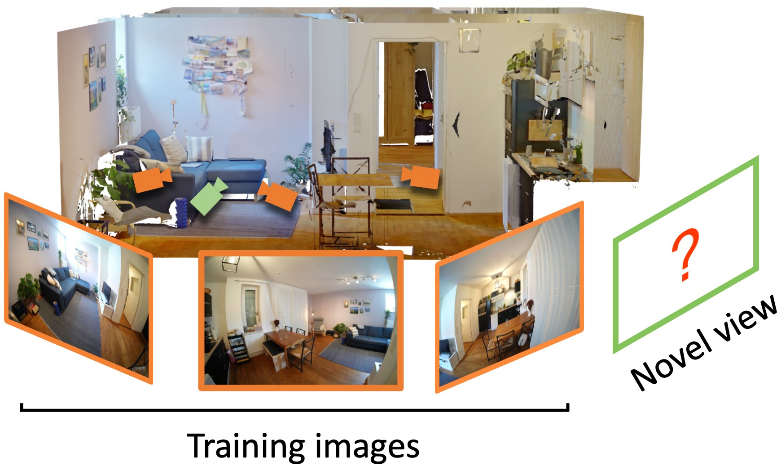


Figure 1: Novel View Synthesis in ScanNet++ dataset.

using the ScanNet++ dataset [28]. Numerous existing solutions model 3D scenes to render novel views. We discuss these solutions in detail in Section 2.

One of the most popular solutions for novel view synthesis, introduced in 2021, is Neural Radiance Fields (NeRF) [15]. It uses neural networks to implicitly model a 3D scene, enabling view reconstruction from arbitrary positions and orientations. NeRF has since evolved, with variants such as Zip-NeRF [1] currently considered the state of the art in terms of reconstruction quality. NeRF is renowned for its photorealistic rendering and compact neural network representation of large 3D scenes. However, its rendering speed is limited due to the need for extensive multi-layer perceptron (MLP) evaluations per view. Solutions such as InstantNGP [16] utilize hashing and CUDA optimizations to accelerate NeRF. Other techniques, for example MobileNeRF [2] and MeRF [20], rely on alternative parameterizations involving polygonal meshes or voxel grids, which are better suited for real-time rendering. However, NeRF accelerations often come at the expense of quality and encounter memory limitations as mentioned in the aforementioned sources.

3D Gaussian Splatting [9] addresses these limitations by using Gaussian kernels instead of MLPs to model the scene, retaining the photorealistic quality of NeRF while significantly reducing rendering time. Its similarity to point-cloud representation allows for the use of traditional, efficient point cloud rendering pipelines.

Gaussian Splatting represents a turning point in rasterization for computer graphics that goes beyond novel view synthesis. Unlike traditional triangle or polygon rasterization, which are discrete and non-differentiable, Gaussian Splatting provides a continuous and differentiable approach, facilitating the integration with deep learning algorithms.

While Kerbl et al. [9] introduced 3D Gaussian Splatting for multiview reconstruction, our work takes a pedagogical approach, providing a step-by-step introduction to Gaussian Splatting in 1D, 2D, and 3D. Rather than proposing a new architecture, we bridge fundamental signal processing concepts with recent advancements in view synthesis. By detailing the theoretical foundations and progressively extending them to 3D, we aim to make Gaussian Splatting more accessible and comprehensible to a broader audience.

This paper is organized as follows. Section 2 surveys relevant literature on both explicit and implicit 3D representations, highlighting their respective strengths and limitations. Section 3 offers a step-by-step explanation of Gaussian Splatting, spanning its 1D, 2D, and 3D formulations along

[1]`https://doi.org/10.5201/ipol.2025.566`

[2]`https://github.com/OutofAi/2D-Gaussian-Splatting`

[3]`https://github.com/nerfstudio-project/gsplat`

with the underlying mathematics. In Section 4, we detail our experiments, present the results, and provide the associated pseudocode for 3D Gaussian Splatting. We conclude by summarizing key findings and discussing future directions for Gaussian Splatting research.

2 Related Work

This section discusses the most common methods for 3D object/scene representation for view synthesis, as illustrated in Figure 2.

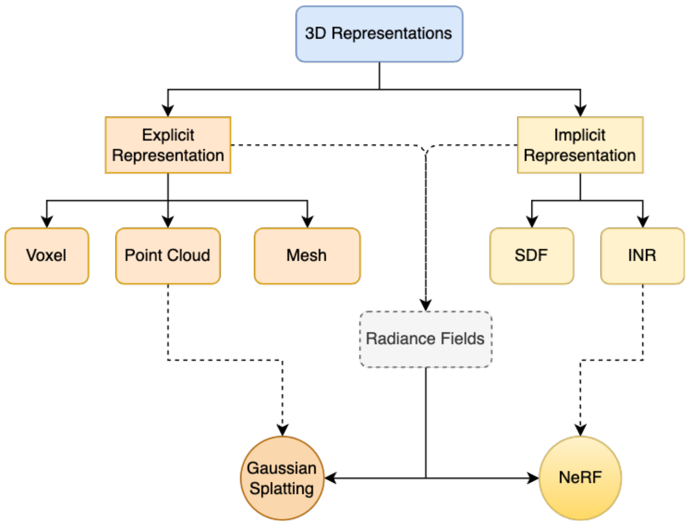


Figure 2: Most common structures and frameworks for 3D scene representation for view synthesis.

2.1 Explicit 3D Representations

These representations explicitly store 3D objects in predefined data structures.

Voxel Grid

A voxel grid is a 3D array that stores values in each voxel (grid element). Earlier methods utilized occupancy grids to denote filled or empty spaces [27, 23]. Subsequent techniques employed different features to capture finer details, such as RGB values or normal vectors [11]. While voxels are straightforward to use, working with 4D structures (e.g., $N \times N \times N \times F$ for an N-regular grid with feature dimensionality F) is memory-inefficient and lacks scalability. Furthermore, obtaining ground-truth voxel-based representations of objects or scenes is costly. Consequently, voxels are rarely the preferred choice in modern learning-based algorithms.

Mesh

Polygonal meshes represent surfaces rather than volumes. They are memory-efficient and have a well-established format for real-time rasterization. Mesh-based learning algorithms typically deform a base mesh to fit the shapes of a 3D scene. However, learning mesh deformation is challenging due to the discrete nature of mesh elements and topological limitations, as demonstrated in Pixel2Mesh [26]. Specialized models such as MobileNeRF [2] convert NeRF representations into meshes, but this conversion results in information loss. Continuous representations such as NeRF [15] are better suited to capture subtle details, such as semi-transparent objects and viewpoint-dependent color variations, which mesh representations struggle to retain.

Point Cloud

A point cloud is a set of 3D points, where each point is represented by a vector of 3D coordinates and other numerical attributes (e.g., color, opacity, semantic class). Point clouds strike a good balance between memory usage and volumetric detail. However, they are less effective for highly complex and dense scenes that require a large number of points. Point clouds can be generated from multiview image collections using Structure-from-Motion (SfM) techniques such as COLMAP [21]. They also serve as an excellent initialization for advanced view synthesis methods, enabling effective Gaussian Splatting [9] or depth-supervised extensions of NeRF [3].

2.2 Implicit 3D Representations

Implicit representations use mathematical functions or neural networks to query the 3D space for properties such as occupancy and color, eliminating the need for large data structures.

Implicit Neural Representation (INR)

Implicit Neural Representations (INRs) employ neural networks to output continuous values representing shape and appearance. Notable examples include Local Lightfield Fusion [14], DeepSDF [19], and NeRF [15]. These methods provide flexible, continuous representations that capture complex geometries and semi-transparent surfaces, though they require extensive training data and significant computational resources.

Signed Distance Function (SDF)

Signed Distance Functions (SDFs) describe 3D surfaces by encoding the distance between each point in space and the nearest surface point. Positive values are assigned to points outside the surface, negative values to points inside, and zero to points on the surface. Neural networks can approximate SDFs, e.g. as in DeepSDF [19]. Despite their utility, SDFs have limitations, including difficulty in representing complex topologies and sensitivity to noise in the data.

Neural Radiance Fields (NeRF)

NeRF [15] employs neural networks to create a compressed, yet highly detailed, representation of 3D scenes. The method maps 3D coordinates and viewing directions to volumetric properties (density and color), achieving remarkable compression of scene information within network parameters. This method is memory-efficient because it captures complex scene details in the parameters of a neural network, requiring less storage than traditional 3D data structures. However, NeRF's main drawback is its high computational cost during rendering, which is a consequence of it using ray marching [6]. Ironically, this also gives photorealism to NeRF's outputs. Techniques such as PlenOctrees [29] and MeRF [20] aim to alleviate this by caching NeRF outputs in a structured format, yet they often struggle with larger scenes due to the complexity and memory overhead involved. Variants such as InstantNGP [16] improve efficiency using feature grids and alternative sampling strategies. Advanced architectures such as PixelNeRF [30] and DietNeRF [8] incorporate image embeddings to significantly reduce the required number of training views. NeRF extensions have also been developed to predict additional scene properties, such as uncertainty or variance estimates, enabling applications in few-shot or in-the-wild scenarios, as demonstrated in ActiveNeRF [18], NeRF-W [13], and Sat-NeRF [12].

Gaussian Splatting

3D Gaussian Splatting (3DGS) [9] combines explicit and implicit field representations. The method models scenes as collections of 3D Gaussian primitives, each characterized by spatial parameters (position, scale, rotation) and appearance attributes (color, opacity). Unlike NeRF, Gaussian Splatting does not rely on neural networks and instead uses an explicit representation. This method does not require MLP evaluations for ray marching at render time, significantly speeding up the rendering process and making it suitable for real-time applications. The similarity of Gaussian Splatting to point clouds also allows the use of established, efficient point cloud rendering pipelines [10]. However, a significant limitation is its memory usage – Gaussian Splatting requires substantial memory to store detailed representations, especially as scene complexity increases. Multi-object scenes, for instance, may require several gigabytes of storage for an accurate depiction. Furthermore, unlike NeRF, Gaussian Splatting, due to not using a neural network, lacks the ability to easily incorporate *world models* or semantic understanding of the scene which may be useful for robotics [25]. While both NeRF and 3DGS excel at photorealistic view synthesis, the point cloud-like nature of these methods makes retrieving exact surface geometry challenging.

2.3 Hybrid Approaches

Hybrid methods combine the strengths of Gaussian Splatting and NeRF to balance photorealism, memory efficiency, and rendering speed. For instance, Compressed 3D Gaussian Splatting [17] and 2D Gaussians for 3D Reconstruction [7] optimize memory efficiency, but these methods are still in early development. CAT3D [5] integrates NeRF with a Multiview Diffusion Model [22] to learn 3D scene representations and uses Gaussian Splatting for fast rendering. Choosing between Gaussian Splatting and Neural Radiance Fields (NeRF) largely depends on the specific requirements of the task at hand, including factors such as computational resources, real-time rendering needs, and memory constraints. While NeRF remains preferred for applications requiring high photorealism, Gaussian Splatting is favored for real-time rendering. Emerging techniques using Multiview Diffusion [24] present a promising competitive direction.

3 Method Description

3.1 1D Gaussian Splatting

Similar to how a signal can be decomposed into different frequency components of sines and cosines using the Fourier transform, a signal can also be decomposed into different Gaussian probability density functions (or kernels). A Gaussian kernel is parametrized using μ (mean) and σ (standard deviation) and is defined at a value t as

$$f(t \mid \mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(t-\mu)^2}{2\sigma^2}\right). \tag{1}$$

Given a sufficient number K of Gaussian kernels, it is possible to reconstruct a signal $\hat{S}(t)$ as

$$\hat{S}(t) = \sum_{i=1}^{K} w_i f_i(t), \tag{2}$$

where w_i is the weight or opacity of the i^{th} kernel, f_i.

From here onwards we will refer to this case as 1D Gaussian Splatting. Note that the Fourier transform is a global representation of a signal whereas Gaussian Splatting is a local representation in which the value of the Gaussian kernel decays rapidly with respect to the input variable, t. It is also important to note that this approach differs from Gaussian Mixture Models (GMMs) [4]. In GMMs, the *signal distribution* is represented by Gaussian kernels, whereas in Gaussian Splatting, the *signal value* is represented using Gaussian kernels. To find the optimal K Gaussian kernels to represent a signal, we can initialize the parameters μ and σ of the Gaussians randomly, ensuring they are within reasonable ranges, and then optimize them using gradient descent. The loss function $\mathcal{L}$ used to minimize the difference between the reconstructed signal $\hat{S}(t)$ and the ground truth signal $S(t)$ is given by

$$\mathcal{L} = \frac{1}{T}\sum_{t=1}^{T}(\hat{S}(t) - S(t))^2. \tag{3}$$

3.2 2D Gaussian Splatting

Similar to the 1D case, we can represent 2D Gaussians as follows

$$f(x, y \mid \mu, \Sigma) = \frac{1}{2\pi\sqrt{|\Sigma|}} \exp\left(-\frac{1}{2}\begin{bmatrix} x-\mu_x \\ y-\mu_y \end{bmatrix}^\top \Sigma^{-1} \begin{bmatrix} x-\mu_x \\ y-\mu_y \end{bmatrix}\right). \tag{4}$$

In practice, we represent the covariance matrix Σ using rotation and scaling coefficients, leveraging the fact that any positive semidefinite covariance matrix can be decomposed into these components. Given a data distribution modeled as an isotropic Gaussian, a transformation T applied to the data modifies the covariance structure as follows. If the transformed data is given by $D' = TD$ and T is decomposed as $T = RS$, where R is a rotation matrix and S is a scaling matrix, then the covariance of the transformed data is

$$\Sigma = \mathbb{E}[D'D'^T] = T\mathbb{E}[DD^T]T^T. \tag{5}$$

Since D follows an isotropic Gaussian, its covariance is proportional to the identity matrix, i.e., $\sigma^2 I$, yielding

$$\Sigma = TT^T = RSS^TR^T \tag{6}$$

This result confirms that the covariance matrix can always be decomposed into a rotation R, which determines the orientation of the Gaussian, and a scaling matrix S, which controls the spread along the principal axes. In 2D, the rotation matrix takes the form

$$R = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}. \tag{7}$$

Thus, instead of storing a full 2×2 rotation matrix, we only need to represent a single parameter θ, which fully determines the orientation of the Gaussian.

If we have K Gaussian kernels, the reconstructed image $\hat{I}(x, y)$ is given by

$$\hat{I}(x, y) = \sum_{i=1}^{K} w_i f_i(x, y). \tag{8}$$

Analogous to the 1D case, the representation of an image as K Gaussians can be optimized via gradient descent. The loss function $\mathcal{L}$ used to minimize the difference between the reconstructed image $\hat{I}(x, y)$ and the ground truth image $I(x, y)$ includes an L_1 term (mean absolute error) and a DSSIM term (1 - SSIM), also called structural dissimilarity, and is given by

$$\mathcal{L} = (1-\lambda)\mathcal{L}_{L1} + \lambda\mathcal{L}_{DSSIM}, \tag{9}$$

where λ is a weight parameter that balances the two loss components. Combining both terms favors a compromise between pixel-level accuracy and perceptual quality, as DSSIM captures structural dissimilarity in a way that aligns with human perception. Although L_2 loss can also be used, L_1 loss is preferred, as suggested in the original 3D Gaussian Splatting framework [9], due to its robustness against outliers. This choice extends naturally to the 2D Gaussian Splatting approach proposed here.

3.3 3D Gaussian Splatting

Building on the concept of Gaussian Splatting in 1D and 2D, the approach is similarly extended to 3D in [9] for efficient representation and rendering of complex 3D scenes. A 3D Gaussian function is defined by its mean vector μ, which is equivalent to the position, and a covariance matrix Σ, representing the spatial distribution and orientation of the points in 3D space. A 3D Gaussian function $G(x, y, z)$ is given by

$$G(x,y,z \mid \mu, \Sigma) = \frac{1}{(2\pi)^{3/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2} \begin{bmatrix} x-\mu_x \\ y-\mu_y \\ z-\mu_z \end{bmatrix}^\top \Sigma^{-1} \begin{bmatrix} x-\mu_x \\ y-\mu_y \\ z-\mu_z \end{bmatrix} \right). \tag{10}$$

In Gaussian Splatting, we represent the reconstructed 3D scene $\hat{V}(x, y, z)$ using a sum of K Gaussian kernels

$$\hat{V}(x,y,z) = \sum_{i=1}^{K} \alpha_i G_i(x,y,z), \tag{11}$$

where each Gaussian $G_i(x, y, z)$ is parameterized by its parameters μ_i (mean), Σ_i (covariance) and α_i (blending weight or opacity).

To project these 3D Gaussians onto a 2D image plane for rendering, the covariance matrix Σ is transformed into the 2D image space using the viewing transformation matrix W. Since covariance transforms under a linear transformation T as $\Sigma' = T\Sigma T^\top$, applying W first gives the intermediate covariance

$$\Sigma_W = W\Sigma W^\top. \tag{12}$$

The projection onto the image plane is nonlinear, but locally it can be approximated by an affine transformation whose Jacobian matrix is denoted as J. Applying the Jacobian transformation to Σ_W, the final covariance in 2D camera coordinates is given by

$$\Sigma' = J\Sigma_W J^\top = JW\Sigma W^\top J^\top, \tag{13}$$

where Σ' is the covariance mapped on the 2D image plane. This makes sure that the transformed covariance correctly captures the effects of both the viewing transformation and the perspective projection, ensuring that the Gaussian remains elliptical in 2D space.

The complete rendering process is described in [9]. It uses alpha blending to accumulate the contributions of each Gaussian to the final image. For this purpose, each Gaussian is additionally characterized by a given color c_i and an opacity coefficient α_i on top of μ_i and Σ.

As discussed in the 2D Gaussian Splatting case, Σ in 3D can also be represented using rotation and scaling matrices as in Equation (6). In this case, the 3D rotation matrix is represented using quaternions.

Similar to the 2D scenario, the loss function is a combination of the L1 loss and DSSIM loss, ensuring both pixel-wise accuracy and structural similarity with the ground truth images. To add or remove Gaussians, if needed, the adaptive density control is employed. It removes Gaussians with negligible contribution (i.e., transparent according to the opacity α_i) and adds Gaussians in areas lacking detail, which exhibit large positional gradients.

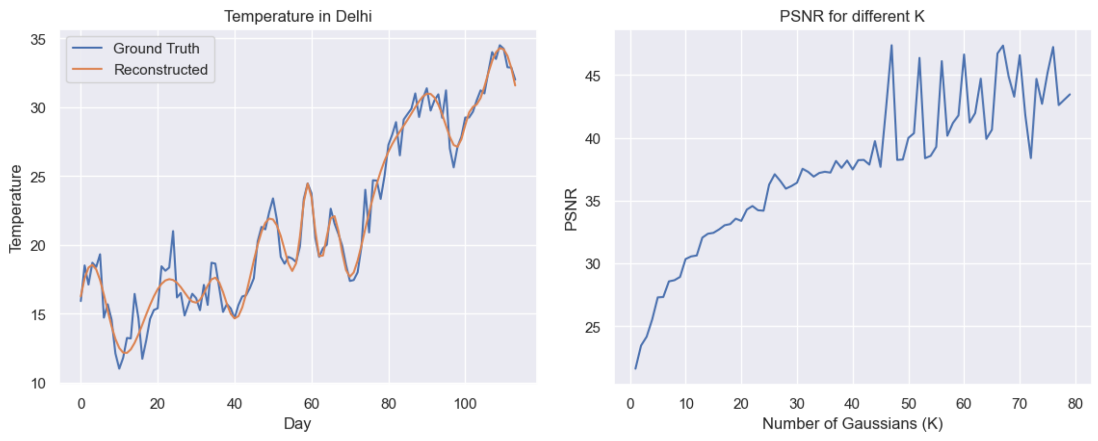
The 3D Gaussian Splatting technique benefits from GPU acceleration, particularly in the rasterization process. A tile-based rasterizer is used to efficiently handle the projection and blending of Gaussians, leveraging fast GPU sorting algorithms to maintain real-time performance.

4 Experiments and Results

4.1 1D Gaussian Splatting

We take a time series of daily temperature in Delhi[4] as an example, and show how it can be represented as Gaussian kernels (Gaussians). We initialized 10 Gaussians with means and variances respectively in the range of t (time). The variances are initialized as log variances for numerical stability. Then the parameters (μ and σ) are learned using gradient descent with 100000 iterations and learning rate 0.1 with Adam optimizer. Figure 3(a) shows the ground truth and the reconstructed signal. Note that the ground truth signal exhibits higher local variance and the reconstructed signal is smoother. This is due to the limited number of Gaussians, which serves as a form of compression. With more Gaussians, it is possible to model a high variance signal more accurately. But the smoothing side-effect may be desirable in some applications such as forecasting where overall behavior is more interesting than small variations. We measured the quality of the reconstructed signal in terms of PSNR (Peak Signal to Noise Ratio).

Figure 3(b) shows how the PSNR changes with the number K of Gaussians used to reconstruct the signal. Initially the PSNR increases with the number of Gaussians, but after a certain point, around $K = 47$, the reconstruction quality starts to oscillate between 40 and 45 dB. Possibly, the latter is due to the choice of hyper-parameters and with some tuning it can be stabilized. It should be noted that the reconstruction quality only improves until a certain number of Gaussians is reached. In fact, the number of Gaussians used to represent a signal is a measure of the compressibility of the signal. A lossless representation (i.e., with $PSNR = \infty$), requires at most as many parameters as in the original signal. In this case we have temperatures for 113 days, so 113 values in total. However, the reconstructed signal achieves a good approximation of 47 dB PSNR with only 45 Gaussians, compressing the data into 90 parameters (each Gaussian has 2 parameters μ and σ). In addition



(a) Ground truth vs Reconstructed signal using 1D Gaussian kernels

(b) Number of Gaussian Kernels used vs Quality of the reconstruction

Figure 3: Comparative analysis of Gaussian kernel reconstruction and its effects on signal quality

to the number of Gaussians, the PSNR can also be affected by other hyper-parameters such as the

[4]Delhi Weather Data. Weather Underground API

learning rate, optimizer type, choice of the loss function, etc. Please refer to the code available in the web page of the article for more details.[5]

4.2 2D Gaussian Splatting

In this section, we explore the application of 2D Gaussian Splatting for image representation and compression.

Experiment Setup

The experiment involved the decomposition of an RGB image into a set of 2D Gaussian kernels, each defined by parameters controlling position, spread, and orientation. These parameters include the means (μ_x, μ_y), standard deviations (σ_x, σ_y), and the angle (θ) used to compute the rotation matrix. The mean values are initialized randomly within the dimensions of the image, the x-y variances (or scales) are initialized in the range [0, 1], opacity is initialized within $[0, 1]$ and angle is initialized within $[-\frac{\pi}{2}, \frac{\pi}{2}]$. The optimization uses the L1 and DSSIM loss introduced in Section 3.2, with the weight parameter $\lambda = 0.2$. The number of Gaussians used to represent the image is dynamically changed in each optimization step in an interleaved manner. In particular, Gaussians with opacity lower than 0.01 are deleted (pruned), Gaussians having high gradient and high variance are split, and the Gaussians with high gradient and low variance are cloned. This is both for computational efficiency and for representing local details with more resolution. Both the loss and the adaptive density control are in line with the 3D Gaussian Splatting [9]. This 2D scenario setup is very similar to the one described in Algorithm 1 for the 3D scenario, except that the initialization is random and there is no camera rasterization step.

Results

The quality of the reconstructed image was quantitatively evaluated using PSNR. After 1001 training steps, the PSNR reaches a reasonable value of 28.52 dB. The compression ratio, determined by comparing the number of pixels in the original image with the number of parameters in the Gaussian model, was 3.35. This demonstrates the effectiveness of 2D Gaussian Splatting in reducing the storage requirements while maintaining high image quality. Figure 4 shows the reconstructed image at different optimization steps. The code is available in the web page of the article for more details.[6]

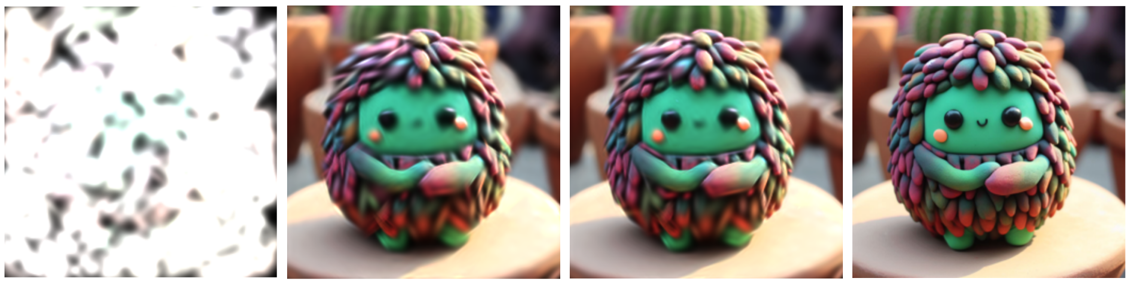

(a) Epoch 1 (b) Epoch 501 (c) Epoch 1001 (d) Ground Truth

Figure 4: Reconstruction of a 2D image using 2D Gaussian Splatting at different epochs. Early epochs show coarse approximations with fewer Gaussians, while the final epoch yields sharper details.

[5] Notebook 1D_approximation_with_gaussians.ipynb
[6] Notebook 2D_Gaussian_Splatting.ipynb

4.3 3D Gaussian Splatting

This section explains how to use 3D Gaussian Splatting for the reconstruction of a single image, as is done in the online demo associated with this article. Algorithm 1, originally presented in the 3DGS [9] paper, provides the pseudocode for the experiment.

Algorithm 1: 3D Gaussian Splatting

```
Input: Set of $N$ views and their camera matrices
Initialize:
  $M \leftarrow$ SfM Points                                  Positions
  $S, C, A \leftarrow$ InitAttributes()                       Covariances, Colors, Opacities
  $i \leftarrow 0$                                            (Iteration Count)

While not converged do
  $V, \hat{I} \leftarrow$ SampleTrainingView()                 Camera V and Image
  $I \leftarrow$ Rasterize($M, S, C, A, V$)
  $L \leftarrow$ Loss($I, \hat{I}$)                            (Loss)
  $M, S, C, A \leftarrow$ Adam($\nabla L$)                     (Backprop and Step)
  If IsRefinementIteration($i$) then
    For all Gaussians $(\mu, \Sigma, c, \alpha)$ in $(M, S, C, A)$ do
      If $\alpha < \epsilon$ or IsTooLarge($\mu, \Sigma$) then      Pruning
        RemoveGaussian()
      End If
      If $\nabla_p L > \tau_p$ then                            Densification
        If $\|S\| > \tau_S$ then                               Over-reconstruction
          SplitGaussian($\mu, \Sigma, c, \alpha$)
        Else                                                   Under-reconstruction
          CloneGaussian($\mu, \Sigma, c, \alpha$)
        End If
      End If
    End For
  End If
  $i \leftarrow i + 1$
End While
```

In 3D Gaussian Splatting, each Gaussian is represented by a mean, a covariance matrix (i.e., scaling and rotation matrices) and the opacity. Spherical harmonic coefficients are also used to capture view-dependent effects.

The means (position coordinates) of the Gaussians are initialized in the range of normalized bounds (in this case between -1 and 1). The scales (standard deviation) for all dimensions are initialized between 0 and 1. The unit quaternion q is initialized, with real part q_r and imaginary parts q_i, q_j, q_k. This can be converted to a rotation matrix R as follows

$$R(q) = 2 \begin{bmatrix} \frac{1}{2} - (q_j^2 + q_k^2) & (q_i q_j - q_r q_k) & (q_i q_k + q_r q_j) \\ (q_i q_j + q_r q_k) & \frac{1}{2} - (q_i^2 + q_k^2) & (q_j q_k - q_r q_i) \\ (q_i q_k - q_r q_j) & (q_j q_k + q_r q_i) & \frac{1}{2} - (q_i^2 + q_j^2) \end{bmatrix}. \tag{14}$$

The resulting rotation matrix is then used to calculate the covariance matrix. In the original 3DGS [9] the positions of the Gaussians are initialized using the sparse point clouds generated by a SfM reconstruction tool such as COLMAP [21].

The Gaussians are projected onto the 2D image plane using Equation (13). Rasterization then converts these projections into pixel values, accounting for overlapping Gaussians and their respective colors and opacities. Details can be found in the supplementary material of 3DGS [9].

Once Gaussians are projected onto the image plane and rasterized to obtain pixel values, these values are compared with the ground truth using the loss function defined in Equation (9), and the Adam optimizer is used to perform gradient descent.

Adaptive density control is applied in an interleaved manner (that is, after every specified iteration) in Algorithm 1. The Gaussians with opacities below a certain threshold are pruned. The ones with large positional gradients and large covariance and split, and the ones with large positional gradients but small covariance are cloned. The former is indicative of over-reconstruction and the latter of under-reconstruction.

IPOL Demo Application The described method has been implemented as part of an IPOL demo,[7] where users can give different inputs, modify parameters such as the number of Gaussians and learning rate, and visualize the effects of 3D Gaussian Splatting on a single image in real-time. Please refer to the code in the web page of the article for more details.[8]

5 Conclusion

In this paper we have explored the multifaceted domain of Gaussian Splatting by presenting its foundational principles in 1D, 2D, and 3D settings. Through experiments on time-series and image reconstruction tasks, we have illustrated the technique's versatility and potential as a powerful alternative to concurrent methods like NeRF. While offering comparable image quality, Gaussian Splatting often provides significantly faster training and inference speeds, as demonstrated in the associated online demo for single-image reconstruction.

Despite these advantages, important challenges remain. Chief among them is memory usage, which grows with scene complexity and may benefit from more advanced compression strategies or hierarchical data structures. Another limitation lies in global geometry constraints: although Gaussian kernels excel at locally representing appearance, they do not inherently handle occlusions or semantic relationships at the scene level. Further, the integration of Gaussian Splatting with learned neural networks—for instance, to capitalize on a global scene prior—remains an open field. Tackling these limitations offers promising directions for future work, enabling richer, large-scale reconstructions and improved real-time performance across a range of computer vision and graphics applications.

Image Credits

from OutofAi. (n.d.). 2D Gaussian Splatting[9].

References

[1] J. T. Barron, B. Mildenhall, D. Verbin, P. P. Srinivasan, and P. Hedman, *Zip-NeRF: Anti-Aliased Grid-Based Neural Radiance Fields*, in IEEE/CVF International Conference

[7] https://ipolcore.ipol.im/demo/clientApp/demo.html?id=566

[8] File main.py

[9] https://raw.githubusercontent.com/OutofAi/2D-Gaussian-Splatting/main/Image-01.png

on Computer Vision, 2023, pp. 19697–19705, https://doi.org/10.1109/ICCV51070.2023.01804.

[2] Z. Chen, T. Funkhouser, P. Hedman, and A. Tagliasacchi, *MobileNeRF: Exploiting the Polygon Rasterization Pipeline for Efficient Neural Field Rendering on Mobile Architectures*, in IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2023, pp. 16569–16578, https://doi.org/10.1109/CVPR52729.2023.01590.

[3] K. Deng, A. Liu, J.-Y. Zhu, and D. Ramanan, *Depth-Supervised NeRF: Fewer Views and Faster Training for Free*, in IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2022, pp. 12882–12891, https://doi.org/10.1109/CVPR52688.2022.01254.

[4] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification and Scene Analysis*, vol. 3, Wiley New York, 1973.

[5] R. Gao, A. Holynski, P. Henzler, A. Brussee, R. Martin-Brualla, P. Srinivasan, J. T. Barron, and B. Poole, *CAT3D: Create Anything in 3D with Multi-View Diffusion Models*, ArXiv Preprint ArXiv:2405.10314, (2024). https://doi.org/10.48550/arXiv.2405.10314.

[6] J. C. Hart, *Sphere Tracing: A Geometric Method for the Antialiased Ray Tracing of Implicit Surfaces*, The Visual Computer, 12 (1996), pp. 527–545, https://doi.org/10.1007/s003710050084.

[7] B. Huang, Z. Yu, A. Chen, A. Geiger, and S. Gao, *2D Gaussian Splatting for Geometrically Accurate Radiance Fields*, ArXiv Preprint ArXiv:2403.17888, (2024), https://doi.org/10.1145/3641519.3657428.

[8] A. Jain, M. Tancik, and P. Abbeel, *Putting NeRF on a Diet: Semantically Consistent Few-Shot View Synthesis*, in IEEE/CVF International Conference on Computer Vision, 2021, pp. 5885–5894, https://doi.org/10.1109/ICCV48922.2021.00583.

[9] B. Kerbl, G. Kopanas, T. Leimkühler, and G. Drettakis, *3D Gaussian Splatting for Real-Time Radiance Field Rendering*, ACM Transactions on Graphics, 42 (2023), pp. 1–14, https://doi.org/10.1145/3592433.

[10] P. E. Kivi, M. J. Mäkitalo, J. Žádník, J. Ikkala, V. K. M. Vadakital, and P. O. Jääskeläinen, *Real-Time Rendering of Point Clouds with Photorealistic Effects: a Survey*, IEEE Access, 10 (2022), pp. 13151–13173, https://doi.org/10.1109/ACCESS.2022.3146768.

[11] F. Liu, C. Shen, G. Lin, and I. Reid, *Learning Depth from Single Monocular Images Using Deep Convolutional Neural Fields*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 38 (2015), pp. 2024–2039, https://doi.org/10.1109/TPAMI.2015.2505283.

[12] R. Marí, G. Facciolo, and T. Ehret, *Sat-NeRF: Learning Multi-View Satellite Photogrammetry with Transient Objects and Shadow Modeling Using RPC Cameras*, in IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2022, pp. 1311–1321, https://doi.org/10.1109/CVPRW56347.2022.00137.

[13] R. Martin-Brualla, N. Radwan, M. S. Sajjadi, J. T. Barron, A. Dosovitskiy, and D. Duckworth, *NeRF in the Wild: Neural Radiance Fields for Unconstrained Photo Collections*, in IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2021, pp. 7210–7219, https://doi.org/10.1109/CVPR46437.2021.00713.

[14] B. Mildenhall, P. P. Srinivasan, R. Ortiz-Cayon, N. K. Kalantari, R. Ramamoorthi, R. Ng, and A. Kar, *Local Light Field Fusion: Practical View Synthesis with Prescriptive Sampling Guidelines*, ACM Transactions on Graphics (TOG), 38 (2019), pp. 1–14.

[15] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, *NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis*, Communications of the ACM, 65 (2021), pp. 99–106, https://doi.org/10.1007/978-3-030-58452-8_24.

[16] T. Müller, A. Evans, C. Schied, and A. Keller, *Instant Neural Graphics Primitives with a Multiresolution Hash Encoding*, ACM Transactions on Graphics (TOG), 41 (2022), pp. 1–15.

[17] S. Niedermayr, J. Stumpfegger, and R. Westermann, *Compressed 3D Gaussian Splatting for Accelerated Novel View Synthesis*, in IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2024, pp. 10349–10358, https://doi.org/10.1109/CVPR52733.2024.00985.

[18] X. Pan, Z. Lai, S. Song, and G. Huang, *ActiveNeRF: Learning Where to See with Uncertainty Estimation*, in European Conference on Computer Vision, Springer, 2022, pp. 230–246, https://doi.org/10.1007/978-3-031-19827-4_14.

[19] J. J. Park, P. Florence, J. Straub, R. Newcombe, and S. Lovegrove, *DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation*, in IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2019, pp. 165–174, https://doi.org/10.1109/CVPR.2019.00025.

[20] C. Reiser, R. Szeliski, D. Verbin, P. Srinivasan, B. Mildenhall, A. Geiger, J. Barron, and P. Hedman, *MERF: Memory-Efficient Radiance Fields for Real-Time View Synthesis in Unbounded Scenes*, ACM Transactions on Graphics (TOG), 42 (2023), pp. 1–12, https://doi.org/10.1145/3592426.

[21] J. L. Schonberger and J.-M. Frahm, *Structure-From-Motion Revisited*, in IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 4104–4113, https://doi.org/10.1109/CVPR.2016.445.

[22] Y. Shi, P. Wang, J. Ye, M. Long, K. Li, and X. Yang, *MVDream: Multi-View Diffusion for 3D Generation*, ArXiv Preprint ArXiv:2308.16512, (2023). https://doi.org/10.48550/arXiv.2308.16512.

[23] S. Song, F. Yu, A. Zeng, A. X. Chang, M. Savva, and T. Funkhouser, *Semantic Scene Completion from a Single Depth Image*, in IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 1746–1754, https://doi.org/10.1109/CVPR.2017.28.

[24] S. Tang, F. Zhang, J. Chen, P. Wang, and Y. Furukawa, *MVDiffusion: Enabling Holistic Multi-View Image Generation with Correspondence-Aware Diffusion*, ArXiv Preprint ArXiv:2307.01097, (2023). https://doi.org/10.48550/arXiv.2307.01097.

[25] G. Wang, L. Pan, S. Peng, S. Liu, C. Xu, Y. Miao, W. Zhan, M. Tomizuka, M. Pollefeys, and H. Wang, *NeRF in Robotics: A Survey*, ArXiv Preprint ArXiv:2405.01333, (2024). https://doi.org/10.48550/arXiv.2405.01333.

[26] N. Wang, Y. Zhang, Z. Li, Y. Fu, W. Liu, and Y.-G. Jiang, *Pixel2Mesh: Generating 3D Mesh Models from Single Rgb Images*, in European Conference on Computer Vision, 2018, pp. 52–67, https://doi.org/10.1007/978-3-030-01252-6_4.

[27] J. Wu, C. Zhang, X. Zhang, Z. Zhang, W. T. Freeman, and J. B. Tenenbaum, *Learning 3D Shape Priors for Shape Completion and Reconstruction*, in European Conference on Computer Vision, vol. 3, 2018.

[28] C. Yeshwanth, Y.-C. Liu, M. Niessner, and A. Dai, *Scannet++: A High-Fidelity Dataset of 3D Indoor Scenes*, in IEEE/CVF International Conference on Computer Vision, 2023, pp. 12–22, https://doi.org/10.1109/ICCV51070.2023.00008.

[29] A. Yu, R. Li, M. Tancik, H. Li, R. Ng, and A. Kanazawa, *PlenOctrees for Real-Time Rendering of Neural Radiance Fields*, in IEEE/CVF International Conference on Computer Vision, 2021, pp. 5752–5761, https://doi.org/10.1109/ICCV48922.2021.00570.

[30] A. Yu, V. Ye, M. Tancik, and A. Kanazawa, *PixelNeRF: Neural Radiance Fields from One or Few Images*, in IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2021, pp. 4578–4587, https://doi.org/10.1109/CVPR46437.2021.00455.