Analysis of Gaussian Splatting and implementation of 2D

Aymane Hamdaoui
aymane.hamdaoui@telecom-paris.fr
ENS Paris-Saclay
Paris, France

Titouan Duhazé
titouan.duhaze@telecom-paris.fr
ENS Paris-Saclay
Paris, France

Abstract

This report presents an analysis of 3D Gaussian Splatting, a recent method for real-time radiance field rendering. We deconstruct the mathematical formulation of the method, focusing on the anisotropic covariance optimization and adaptive density control. To demonstrate a critical understanding of these mechanisms, we implement a simplified 2D Gaussian Splatting solver. Our experiments visualize the splitting and cloning strategies in a controlled 2D environment, verifying the convergence properties independent of 3D projection artifacts. We conclude by discussing the limitations of explicit representations regarding memory consumption and initialization constraints.

Keywords

Gaussian Splatting, Radiance Fields, Differentiable Rendering

1 Introduction

The field of novel-view synthesis has recently been revolutionized by Neural Radiance Fields (NeRFs). While generating new images from a set of photos and sparse point clouds has been a challenge for decades, Zwicker et al. [2001] originally proposed rendering point clouds using splatting primitives with an extent larger than a pixel.

Historically, a conflict has emerged between geometry-based and volumetric rendering. The "Explicit" school utilized meshes or point clouds, typically generated via Structure-from-Motion (SfM) [Schönberger and Frahm 2016]. These methods benefited from fast GPU rasterization but struggled with topology (e.g., holes, jagged edges) and semi-transparent phenomena. Conversely, the "Implicit" school, introduced by NeRF [Mildenhall et al. 2020], solved topological issues by representing scenes as continuous volumetric fields, offering photorealism but at a significant computational cost.

The current state-of-the-art in quality, Mip-NeRF360 [Barron et al. 2022], remains insufficient for real-time applications. While achieving outstanding visual fidelity, its training and rendering times are prohibitive. This limitation is intrinsic to the NeRF approach: the technique relies on volumetric ray marching, which requires sampling hundreds of points along a single ray to query a neural network [Kerbl et al. 2023]. This stochastic sampling is computationally expensive, leading to low frame rates (< 1 FPS).

While recent works have attempted to address this via spatial data structures like grids or hashes [Fridovich-Keil et al. 2022; Müller et al. 2022], Kerbl et al. [2023] argue that these attempts still rely on the ray-marching paradigm, limiting their efficiency in representing empty space versus detailed objects.

Kerbl et al. [2023] propose using 3D Gaussians as the scene primitive to bridge the gap between these two schools. The core insight stems from the mathematical similarity between the two image formation models.

In volumetric rendering (NeRF), the color C is obtained by integrating density σ and color c along a ray:

$$C = \sum_{i=1}^{N} T_i \alpha_i c_i \tag{1}$$

where $\alpha_i = (1 - \exp(-\sigma_i \delta_i))$ and transmittance is $T_i = \prod_{j=1}^{i-1}(1-\alpha_j)$.

In Point-Based Splatting, the color is computed by blending $\mathcal{N}$ ordered points overlapping the pixel:

$$C = \sum_{i \in \mathcal{N}} c_i \alpha_i \prod_{j=1}^{i-1}(1 - \alpha_j) \tag{2}$$

Since these formulations are mathematically equivalent, the authors demonstrate that expensive ray marching can be abandoned in favor of rasterization (sorting points), provided the primitive is sufficiently flexible.

This report analyzes the three key contributions of [Kerbl et al. 2023]:

- **Representation:** The use of Anisotropic 3D Gaussians (ellipsoids), which allow for the compact representation of thin structures.
- **Optimization:** An Interleaved Optimization strategy that includes Adaptive Density Control to dynamically "Clone" or "Split" Gaussians during training.
- **Rendering:** A Differentiable Tile-Based Rasterizer that sorts Gaussians by depth to enable correct α-blending.

The resulting method challenges the trend of using Neural Networks for scene representation, demonstrating that an explicit, unstructured representation—if optimized correctly—can surpass implicit neural methods in both speed (≥ 30 FPS) and quality.

2 Method

The method operates as a continuous pipeline: it initializes a set of 3D Gaussians from sparse SfM points, then iteratively optimizes their parameters (position, covariance, opacity, color) while dynamically densifying the geometry, finally projecting them via a tile-based rasterizer for real-time rendering. In the next sections we will focus more in details on the different steps of this pipeline.

2.1 Gaussian Representation

To represent the scene geometry, the method abandons traditional planar surfels, which require estimated surface normals, in favor of explicit 3D Gaussians. Traditional surface splatting requires a normal vector at each point to orient the splat. Standard estimation methods rely on fitting a plane to a local neighborhood of k-nearest neighbors. However, because SfM point clouds are extremely sparse and noisy, these neighborhoods often span disjoint objects or large voids, resulting in chaotic and unreliable normal estimates. Given

the sparsity of the initialization points from Structure-from-Motion (SfM), estimating accurate normals is ill-posed [Kerbl et al. 2023].

Instead, the scene is modeled as a set of 3D Gaussians defined by a position (mean) μ and a 3D covariance matrix Σ:

$$G(x) = \exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right) \tag{3}$$

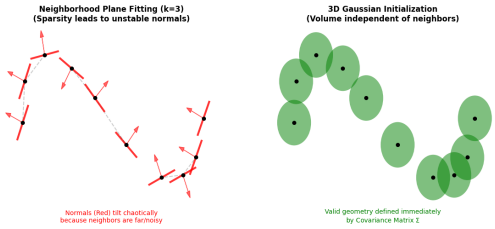


Figure 1: Comparison of geometric primitives on sparse SfM data.

To render the scene, these 3D Gaussians are projected into 2D image space. Given a viewing transformation W and the Jacobian J of the affine approximation of the projective transformation, the 2D covariance Σ' is computed as:

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

This projection allows the algorithm to utilize standard α-blending techniques for rendering [Zwicker et al. 2001].

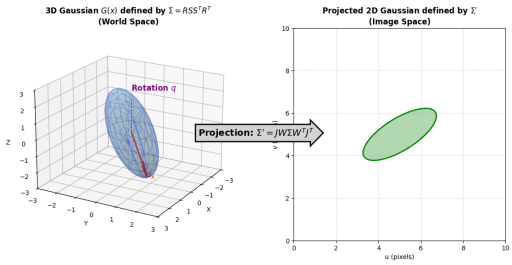


Figure 2: Visual representation of the Gaussian transformation.

2.1.1 Covariance Parameterization. A significant technical challenge is optimizing the covariance matrix Σ via gradient descent. The matrix must remain positive semi-definite to represent a physically valid ellipsoid. Direct optimization of the matrix coefficients often leads to invalid matrices [Kerbl et al. 2023].

To enforce validity, the authors factorize the covariance matrix into a rotation matrix R and a scaling matrix S:

$$\Sigma = RSS^TR^T \tag{5}$$

For optimization purposes, these are stored separately as a unit quaternion q (representing rotation) and a 3D vector s (representing scaling). This parameterization ensures that Σ remains valid throughout the optimization while allowing the Gaussians to stretch anisotropically to represent complex geometry such as thin walls or fibers.

2.2 Optimization and Density Control

A key contribution of the paper is the dynamic densification strategy. The core of the method is an interleaved optimization process. Unlike traditional mesh fitting where topology is fixed, or voxel grids where resolution is static, 3D Gaussian Splatting dynamically alters the number and density of primitives to match the scene complexity.

2.2.1 Optimization Objective. The parameters optimized for each Gaussian are the position μ, covariance Σ, opacity α, and Spherical Harmonic (SH) coefficients c representing direction-dependent color [Kerbl et al. 2023]. The optimization is performed using Stochastic Gradient Descent (SGD).

The loss function combines pixel-wise error ($\mathcal{L}_1$) with a structural similarity term (D-SSIM) to preserve local structure:

$$\mathcal{L} = (1-\lambda)\mathcal{L}_1 + \lambda\mathcal{L}_{\text{D-SSIM}} \tag{6}$$

where $\lambda = 0.2$ in the authors' implementation.

2.2.2 Adaptive Density Control. A critical contribution is the strategy to populate empty space and refine details. The authors observe that high view-space positional gradients $\nabla_p\mathcal{L}$ indicate regions where the geometry is currently insufficient [Kerbl et al. 2023].

The densification logic follows two distinct branches based on the scale S of the Gaussian:

(1) **Under-Reconstruction (Clone):** If a Gaussian is small but has a high gradient, it implies a lack of geometry in that region (a "hole"). The algorithm clones the Gaussian to fill the void.
(2) **Over-Reconstruction (Split):** If a Gaussian is large and has a high gradient, it implies the primitive is too coarse to represent the underlying fine detail. The algorithm splits the Gaussian into two smaller instances, reducing their scale by a factor of $\phi = 1.6$.

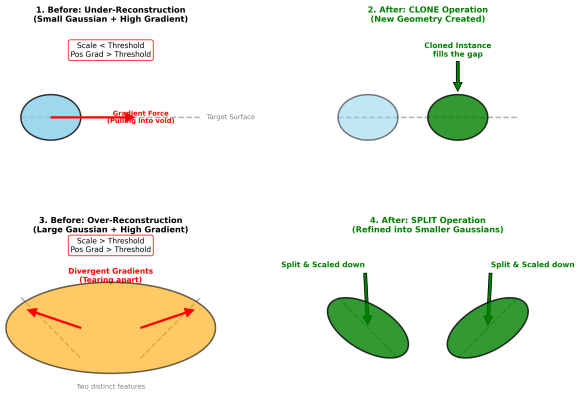


Figure 3: Visual representation of the Adaptive Density Control. Up: a case of Under-Reconstruction. Down: a case of Over-Reconstruction.

2.2.3 Regularization and Pruning. To maintain a clean representation, the optimization includes periodic pruning. Gaussians with opacity α below a threshold ϵ_α are removed to eliminate invisible redundancy. Furthermore, to remove "floaters" (artifacts close to the camera) and prevent local minima, the opacity of all Gaussians is reset to a low value every 3000 iterations, forcing the optimization to re-validate the existence of valid geometry [Kerbl et al. 2023].

2.3 Fast Differentiable Rasterizer

To achieve real-time performance, the method employs a tile-based rasterizer inspired by software rasterization [Lassner and Zollhofer 2021]. The screen is divided into 16×16 tiles, and 3D Gaussians are culled against the view frustum.

2.3.1 Sorting Strategy. The critical bottleneck in splatting is correct visibility ordering (handling occlusion). Rather than performing expensive per-pixel sorting, the authors employ a fast GPU Radix sort [Merrill and Grimshaw 2010].

Each Gaussian is assigned a 64-bit key: the higher bits represent the *Tile ID* and the lower bits represent the *View-Space Depth*. Sorting this single list effectively orders all primitives for rendering.

It is important to note that this sorting is performed at the tile level, not the pixel level. Therefore, depth is calculated based on the primitive's center. For intersecting, large Gaussians, this single depth value may not represent the correct visibility order for every pixel covered by the splat. However, the authors argue that this approximation becomes negligible as the splats converge to pixel size [Kerbl et al. 2023].

2.3.2 Alpha Blending and Backpropagation. For each pixel, threads traverse the sorted Gaussians front-to-back, accumulating color and opacity using the standard α-blending equation (Eq. 3). To optimize performance, the traversal stops once the accumulated opacity saturates ($\alpha \approx 1$).

Unlike previous approaches that limit gradient computation to the top-K primitives [Lassner and Zollhofer 2021], this method backpropagates gradients to *all* contributing Gaussians. This design choice, while memory-intensive during training, ensures that the optimization does not stagnate in local minima due to vanishing gradients on occluded primitives.

3 Experiment: 2D Gaussian Splatting

To critically evaluate the optimization dynamics of the method independent of the complexities of 3D-to-2D projection, we implemented a simplified 2D Gaussian Splatting solver using PyTorch. This experiment aims to isolate the behavior of the **Adaptive Density Control** strategy and verify its ability to reconstruct sharp geometries from sparse initialization.

3.1 Experimental Setup

We defined a synthetic target image (128×128 pixels) containing two distinct features to test different reconstruction capabilities:

- **A Red Circle:** Represents smooth, low-frequency geometry (only a few Gaussian required to describe it).
- **A Blue Square:** Represents sharp, high-frequency edges, challenging the soft Gaussian primitives.

To represent view-dependent appearance effects such as specular highlights and reflections, the method optimizes coefficients for Spherical Harmonics (SH) rather than static RGB values [Kerbl et al. 2023]. This allows the emitted radiance of each Gaussian to vary smoothly based on the viewing angle. In our this experiment, we simplified this to static RGB parameters, as 2D image reconstruction does not involve changing viewing angles.

The scene was initialized with $N = 20$ random Gaussians. Unlike the full 3D method which uses Structure-from-Motion points, we initialized our Gaussians with a large scale ($\sigma \approx 15$ pixels) to ensure initial overlap with the target features. The optimization ran for 2,500 iterations using the Adam optimizer.

3.2 Methodology

We adapted the 3D covariance formulation to 2D. Each primitive is defined by a 2D position $\mu \in \mathbb{R}^2$, a 2D scaling vector s, and a single rotation angle θ. The covariance Σ is constructed as:

$$\Sigma = R(\theta) \begin{pmatrix} s_x & 0 \\ 0 & s_y \end{pmatrix} \begin{pmatrix} s_x & 0 \\ 0 & s_y \end{pmatrix}^T R(\theta)^T \tag{7}$$

We implemented the **Adaptive Density Control** logic described in the paper: every 100 iterations, we identified Gaussians with the highest positional gradients. Primitives exceeding a scale threshold were *Split*, while smaller primitives were *Cloned*.

3.3 Results and Analysis

The progression of the optimization is visualized in Figure 8. The system successfully converged from 20 initial "blobs" to a dense reconstruction of 600 Gaussians.

Visual Convergence: As seen in Figure 7, the method successfully reconstructed both shapes. Notably, the adaptive density control heavily populated the edges of the blue square with small, highly anisotropic Gaussians to approximate the hard edge, while the red circle required fewer, larger primitives.

Growth Dynamics: The metrics in Figure 8 confirm the effectiveness of the density control. The Gaussian count (blue line) exhibits a stepwise growth corresponding to the density intervals (every 100 iterations). We observe a direct correlation between the spikes in Gaussian count and the rapid decrease in L1 Loss (red line), validating that adding geometry is essential for escaping local minima.

3.4 Difficulties Encountered

During the implementation, we encountered significant stability challenges that highlight the limitations of explicit optimization:

1. The "Cold Start" Problem: Initial experiments failed to converge when Gaussians were initialized with small scales. If a Gaussian does not overlap with a target pixel, its gradient is zero, and it never moves. We resolved this by initializing primitives with large variances (see Fig. 6b) to guarantee non-zero gradients flow through the system at $t = 0$.

2. Numerical Instability (NaN Loss): We observed that the covariance inversion is prone to numerical instability. If the optimization drives a scale parameter s near zero, the covariance matrix becomes singular, causing the loss to explode to `NaN`. We addressed this by enforcing hard constraints (clamping) on the minimum

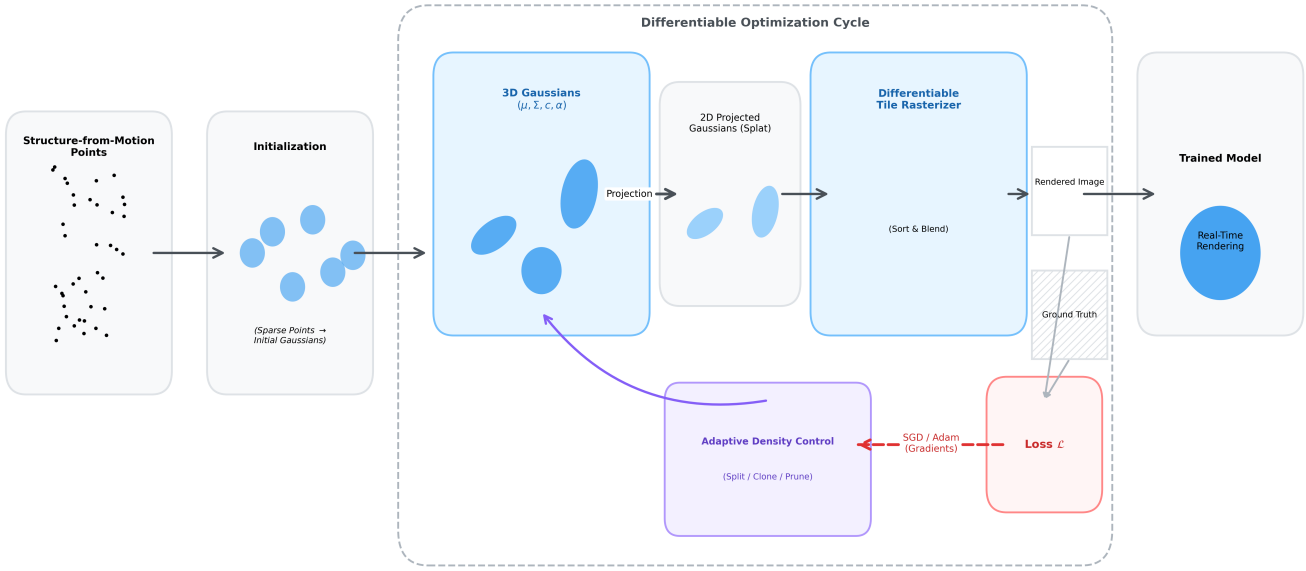


Figure 4: Overview of the complete pipeline. The system integrates explicit Gaussian optimization with a fast tile-based rasterizer to achieve real-time rendering performance.

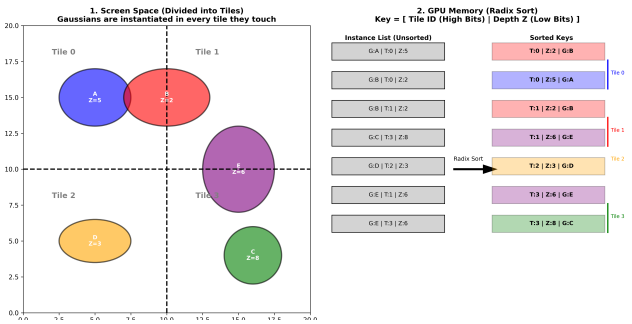
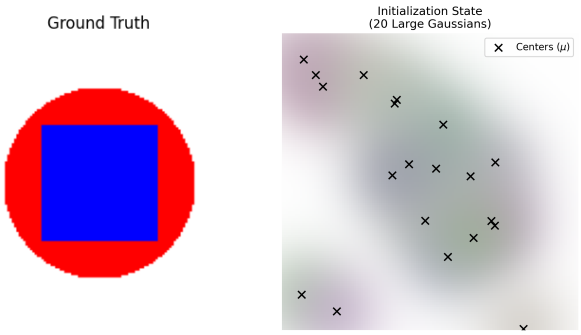


Figure 5: Illustration of the tile-based sorting mechanism. Left: Gaussians are instantiated in every tile they overlap (e.g., Gaussian B overlaps Tile 0 and 1). Right: A single Radix Sort orders these instances based on a 64-bit key combining Tile ID (high bits) and View-Space Depth (low bits). This produces a contiguous list per tile (e.g., the range for Tile 0) sorted front-to-back, enabling efficient α-blending without per-pixel sorting.



(a) Target (Ground Truth) **(b) Initialization ($N = 20$)**

Figure 6: Experimental Setup. We reconstruct a red circle (soft) and blue square (sharp) starting from sparse, large Gaussians.

scale ($\sigma_{min} = 1.0$) and adding an epsilon term to the determinant calculation.

3. Lazy Convergence: Without aggressive intervention, the system tended to stagnate at a low number of Gaussians (≈ 30), representing the target as blurry blobs. To achieve the sharpness shown in Figure 7, we implemented a "Forced Growth" strategy that targets the top 10% of high-error Gaussians for splitting, regardless of an absolute threshold.

4 Results and Evaluation

While our 2D experiment validates the convergence properties of the optimization, we now examine the performance of the full 3D method on standard benchmarks. Kerbl et al. [2023] evaluate their approach on 13 real-world scenes from the Mip-NeRF360, Tanks&Temples, and Deep Blending datasets.

4.1 Quantitative Comparison

The method is compared against two primary classes of baselines:

- **Quality Baselines:** Mip-NeRF360 [Barron et al. 2022], representing the state-of-the-art in visual fidelity but requiring prohibitive training times.

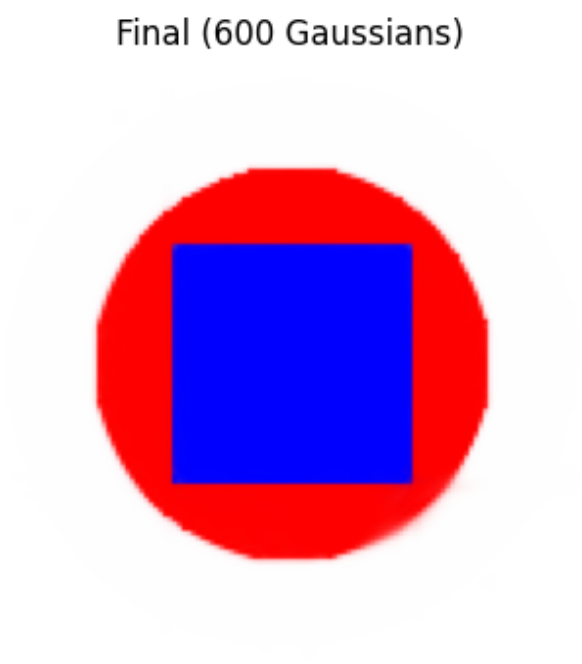


Figure 7: Final Reconstruction. The optimization grew the scene from 20 to 600 Gaussians. Note how the algorithm automatically placed dense, small Gaussians along the sharp edges of the blue square to minimize the L1 loss.

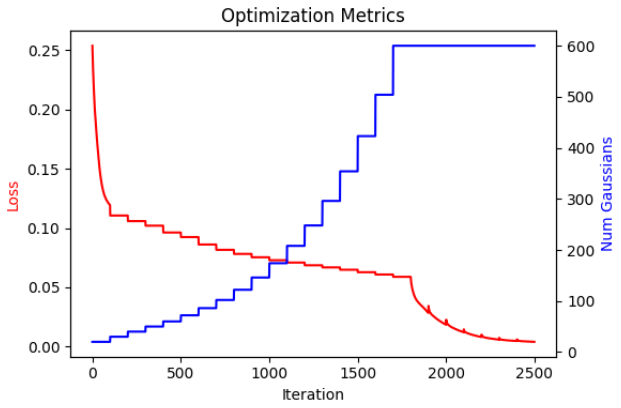


Figure 8: Optimization Metrics. The stepwise increase in Gaussian Count (blue) correlates with drops in Loss (red), demonstrating that the Adaptive Density Control effectively adds capacity to the model where needed.

- **Speed Baselines:** InstantNGP [Müller et al. 2022] and Plenoxels [Fridovich-Keil et al. 2022], which prioritize fast training but often sacrifice high-frequency detail.

As summarized in Table 1, 3D Gaussian Splatting successfully bridges this gap. It matches the visual quality of Mip-NeRF360 (PSNR 27.21 vs. 27.69) while training in a fraction of the time (41 minutes vs. 48 hours). Crucially, it is the only method to achieve high-fidelity results at real-time frame rates (>130 FPS), whereas Mip-NeRF360 renders at 0.06 FPS [Kerbl et al. 2023].

Table 1: Comparison of State-of-the-Art Methods on the Mip-NeRF360 dataset. Gaussian Splatting achieves quality (PSNR) comparable to the slow Mip-NeRF360 while maintaining the real-time frame rates of InstantNGP.

Method	PSNR ↑	SSIM ↑	Train Time	FPS
Plenoxels	23.08	0.626	25 min	7
InstantNGP	25.59	0.699	7 min 30	**9**
Mip-NeRF360	**27.69**	**0.792**	48 hours	0.06
Gaussian Splatting	27.21	0.815	41 min	**135**

4.2 Qualitative Analysis

Beyond the metrics, the explicit nature of the representation offers specific visual advantages. The use of anisotropic Gaussians allows the method to capture thin structures (e.g., bicycle spokes, fences) that are often blurred by the fixed grid resolution of Plenoxels or the spectral bias of MLPs.

However, this explicitness also introduces specific artifacts. Unlike the continuous volumetric fog of NeRFs, the discrete Gaussians can occasionally be seen as individual "splats" or popping artifacts when the camera moves to extreme angles not seen during training. This confirms that while the method solves the rendering speed bottleneck, it trades continuous smoothness for discrete efficiency.

5 Limitations

While 3D Gaussian Splatting offers a compelling alternative to Neural Radiance Fields, the shift from implicit to explicit representations introduces distinct limitations that restrict its robustness and universality.

5.1 Dependency on Prior Geometry

Our experiments reveal that the method's performance is inextricably linked to the quality of its initialization, a dependency that is partially masked in the original paper by the use of high-quality SfM data.

- **The "Cold Start" Failure Mode:** In our 2D experiment, initializing with small-scale, random Gaussians resulted in complete optimization failure. Because the Gaussian function decays exponentially, primitives positioned far from the target signal (relative to their variance Σ) produce negligible pixel contributions, causing gradients to effectively underflow to zero. This clarifies a nuance in the authors' claim that random initialization "performs relatively well" [Kerbl et al. 2023]: such robustness is contingent on the initial primitives essentially covering the camera frustum or having sufficiently large scales to overlap with the data. Unlike implicit volumetric methods (NeRFs) which stochastically sample the entire space, Gaussian Splatting lacks an intrinsic exploration mechanism to "pull" geometry from empty space without a valid initial gradient signal.

- **Background Collapse & Floaters:** Even when random initialization succeeds, the authors admit it leads to degradation in background quality and the appearance of "floaters" in unseen regions [Kerbl et al. 2023]. This confirms that while the method can *refine* a good initialization, it struggles to *reconstruct* scene topology from scratch, unlike implicit volumetric methods (NeRFs) which sample the entire space stochastically.

5.2 Fragility of Density Control Hyperparameters

Our experimental struggle with "Lazy Convergence" exposes a critical fragility in the density control mechanism: its reliance on scale-dependent absolute thresholds.

- **Failure of Absolute Thresholds:** The original method triggers densification only if the view-space positional gradient exceeds a fixed value $\tau_{pos} = 0.0002$ [Kerbl et al. 2023]. In our experiment, once the Gaussians formed a blurry approximation of the blue square, the gradient magnitudes dropped below this fixed threshold. The algorithm falsely determined that the geometry was "converged," leaving the result in a suboptimal, blurry state. This demonstrates that τ_{pos} is not a universal constant but depends on the image frequency and contrast.
- **Necessity of Relative Metrics:** To overcome this stagnation, we had to implement a "Forced Growth" strategy based on a *relative* criterion (splitting the top 10% of high-error Gaussians). The fact that a relative metric succeeded where the paper's absolute threshold failed suggests that the original method may struggle to generalize to new datasets without manual hyperparameter tuning.

5.3 Memory and Storage Consumption

Implicit methods compress a scene into the weights of a small MLP (often < 10 MB). In contrast, Gaussian Splatting stores millions of explicit primitives. A trained scene can easily consume gigabytes of disk space. During training, the unoptimized VRAM usage can peak above 20 GB [Kerbl et al. 2023], limiting the method's accessibility compared to memory-efficient grid methods like InstantNGP.

5.4 Sorting Artifacts

To achieve real-time speeds, the rasterizer sorts Gaussians by depth at the *tile* level, not the *pixel* level. This approximation can lead to visual artifacts (popping) at tile boundaries or when Gaussians intersect, as the blending order may flip incorrectly during camera movement.

6 Future Directions

Our analysis identifies initialization robustness as the primary bottleneck of the method. Future research should prioritize removing the dependency on Structure-from-Motion (SfM) priors. Potential avenues include:

- **Generative Initialization:** Leveraging diffusion models or monocular depth estimators to "hallucinate" a dense point cloud for textureless regions, preventing the "Cold Start" problem.
- **Neural-Explicit Hybrids:** Using a lightweight NeRF to discover coarse scene topology first, then distilling that geometry into Gaussians for real-time fine-tuning.

7 Conclusion

We have presented a comprehensive analysis of 3D Gaussian Splatting, validating its core mechanisms through a custom 2D implementation. Our experiments confirm that the combination of anisotropic covariance optimization and adaptive density control allows for the accurate reconstruction of complex geometries without the computational overhead of neural networks.

The method represents a paradigm shift in novel-view synthesis, proving that explicit, rasterization-friendly geometry can match the visual fidelity of implicit volumetric fields. However, our analysis reveals that this performance comes at the cost of robustness. Unlike NeRFs, which explore the entire volume, Gaussian Splatting is heavily dependent on the quality of its initialization and the tuning of density control hyperparameters.

Ultimately, while 3D Gaussian Splatting effectively solves the *rendering* bottleneck, it shifts the complexity to the *initialization* phase. We conclude that while it is a state-of-the-art solution for SfM-compatible scenes, achieving true universality will require developing robust mechanisms to generate geometry in the absence of sparse point clouds.

References

Jonathan T Barron, Ben Mildenhall, Dor Verbin, Pratul P Srinivasan, and Peter Hedman. Mip-nerf 360: Unbounded anti-aliased neural radiance fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5470–5479. IEEE, 2022.

Sara Fridovich-Keil, Alex Yu, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. Plenoxels: Radiance fields without neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5501–5510. IEEE, 2022.

Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics (TOG)*, 42(4):1–14, 2023.

Christoph Lassner and Michael Zollhofer. Pulsar: Efficient sphere-based neural rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1440–1449. IEEE, 2021.

Duane G Merrill and Andrew S Grimshaw. Revisiting sorting for gpgpu stream architectures. In *Proceedings of the 19th International Conference on Parallel Architectures and Compilation Techniques*, pages 545–546, New York, NY, USA, 2010. ACM.

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 (ECCV)*, pages 405–421. Springer, 2020.

Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Transactions on Graphics (TOG)*, 41(4):1–15, 2022.

Johannes L Schönberger and Jan-Michael Frahm. Structure-from-motion revisited. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4104–4113. IEEE, 2016.

Matthias Zwicker, Hanspeter Pfister, Jeroen Van Baar, and Markus Gross. Surface splatting. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, pages 371–378, New York, NY, USA, 2001. ACM.

A Failure Modes

To further validate our analysis of the method's limitations, we visualize the specific failure modes encountered during our experimentation.

A.1 Initialization Sensitivity: The "Zero Gradient" Problem

Figure 9 (Top Row) illustrates the "True Cold Start" failure mode. We initialized Gaussians with small variance distributed along the **image periphery**, ensuring they surround the central target without spatial overlap. As predicted, the loss curve remains completely flat (Fig. 9, Top Right). Because the primitives do not touch the target pixels, the view-space positional gradients are exactly zero ($\nabla_p \mathcal{L} = 0$). The optimization stagnates immediately, proving that the method lacks any mechanism to "pull" geometry into empty space without an initial overlap.

A.2 Stagnation via Absolute Thresholds

Figure 9 (Bottom Row) illustrates the "Lazy Convergence" phenomenon. Here, we initialized the Gaussians randomly with sufficient overlap (Standard Random). While the optimization successfully minimizes the initial error, it stagnates at a suboptimal local minimum. Once the Gaussians roughly covered the shapes, the gradient magnitudes dropped below the fixed absolute threshold τ_{pos}. The system failed to trigger the necessary *Split* operations, resulting in a blurry result that under-fits the high-frequency edges of the blue square.

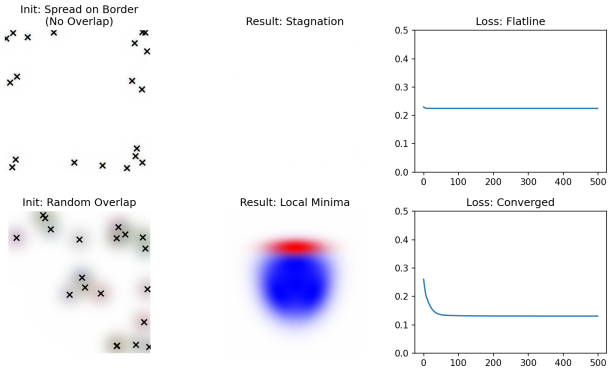


Figure 9: Visualizing Failure Modes. Top Row (Cold Start): Gaussians initialized along the image border (marked with 'x') fail to overlap the target, resulting in a flat loss curve and zero learning. Bottom Row (Lazy Convergence): With random overlap, the method learns the rough shape but stagnates due to the failure of fixed splitting thresholds to capture high-frequency details.