Photorealistic 3D Scene Rendering using Gaussian Splatting

End-to-End Implementation using Nerfstudio and COLMAP on Cloud Infrastructure

Author:

Vasco Sá

GitHub | LinkedIn

28 January 2026

Abstract

This project explores the pipeline of converting a standard 2D video into a navigable 3D scene using Gaussian Splatting. By leveraging Nerfstudio for training and React Three Fiber for rendering, I successfully created a web-based viewer that preserves the photorealistic quality of Neural Radiance Fields while enabling real-time interaction. The report analyzes the training process on cloud infrastructure (Kaggle), achieving a training time of approximately 17 minutes on NVIDIA T4 GPUs, and the optimization for web delivery, reducing the final model from 80MB to 11.2MB through aggressive alpha culling. The project examines specific challenges related to textureless surfaces in the input data, demonstrating that while objects with distinct geometry are reconstructed with high fidelity, uniform surfaces introduce artifacts requiring careful hyperparameter tuning. This work serves as a proof-of-concept for democratizing 3D capture using entirely free, open-source tools.

Keywords

Gaussian Splatting, Photogrammetry, Structure-from-Motion, Nerfstudio, Neural Radiance Fields, Digital Twin, React Three Fiber, Point Cloud Optimization, Web-based Rendering.

Acknowledgements

I would like to acknowledge the open-source contributions of the Nerfstudio Team (University of California, Berkeley) for developing the modular framework used in this project. Additionally, credit is due to Kerbl et al. (Inria, University of Côte d'Azur) for the foundational research on 3D Gaussian Splatting, which made real-time rasterization of radiance fields possible.

Special thanks to the COLMAP developers for the Structure-from-Motion algorithms that enabled the initial camera pose estimation required for this pipeline.

Table of Contents

Nomenclature

Acronym	Definition
3DGS	3D Gaussian Splatting
COLMAP	Structure-from-Motion and Multi-View Stereo (Software)
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture (NVIDIA)
FPS	Frames Per Second
GPU	Graphics Processing Unit
LiDAR	Light Detection and Ranging
LPIPS	Learned Perceptual Image Patch Similarity
MLP	Multi-Layer Perceptron
NeRF	Neural Radiance Field
PSNR	Peak Signal-to-Noise Ratio
R3F	React Three Fiber
RGB	Red Green Blue
SfM	Structure-from-Motion
SGD	Stochastic Gradient Descent
SH	Spherical Harmonics
SIFT	Scale-Invariant Feature Transform
SSIM	Structural Similarity Index
VRAM	Video Random Access Memory

1. Introduction

1.1 Context and Motivation

Traditionally, creating high-fidelity 3D environments required either labor-intensive manual modeling or expensive laser scanning (LiDAR) equipment.

In recent years, the field of Computer Vision has shifted towards Neural Rendering. The introduction of Neural Radiance Fields (NeRFs) in 2020 marked a significant breakthrough, allowing AI to "learn" a 3D scene from 2D images. However, NeRFs suffer from high computational costs due to volumetric ray-marching, making them unsuitable for real-time rendering on consumer devices.

3D Gaussian Splatting (3DGS), introduced by Kerbl et al. in 2023, addresses this limitation. By representing a scene as a collection of anisotropic 3D Gaussians rather than a continuous field or a polygon mesh, 3DGS allows for photorealistic quality with real-time rendering speeds via rasterization. This project explores the practical application of this technology to democratize 3D capture.

1.2 Problem Statement

While 3DGS offers superior performance, the pipeline remains fragmented and technically demanding. Training these models typically requires high-end local GPUs (e.g., NVIDIA RTX 3090/4090) with massive VRAM, and the resulting files are often too large or incompatible with standard web standards.

Furthermore, relying on consumer-grade input data (such as handheld smartphone video) introduces noise, motion blur, and textureless surfaces that traditional Structure-from-Motion (SfM) algorithms struggle to process. There is a need for a streamlined, accessible workflow that bridges the gap between raw smartphone footage and a lightweight, browser-ready 3D experience.

1.3 Objectives

The primary objective of this project is to design and implement an end-to-end pipeline for Neural Rendering, demonstrating that high-quality 3D scenes can be created and deployed using free cloud resources and standard web technologies.

Specific goals include:

1. **Automated Processing:** Implement a "headless" Structure-from-Motion pipeline using COLMAP to extract camera poses from standard video files.
2. **Cloud-Based Training:** Leverage Kaggle’s NVIDIA T4 GPUs to train a Gaussian Splatting model (Nerfstudio’s Splatfacto) without local hardware dependencies.
3. **Web Optimization:** Develop a conversion strategy to transform raw Nerfstudio checkpoints (.ply) into optimized binary formats (.splat) suitable for web streaming.
4. **Interactive Visualization:** Build a high-performance web viewer using React Three Fiber to render the scene in real-time within a modern browser.

1.4 Project Scope

This report documents the complete lifecycle of the "3DGS Viewer" project. It covers the data acquisition methodology, the mathematical foundations of the training process, and the engineering challenges encountered during cloud deployment.

Specifically, this project functions as a proof-of-concept validation for the 3DGS pipeline. Rather than a large-scale architectural scan, the scope is limited to a micro-scale reconstruction: capturing a distinct foreground object (a plastic pen) placed on a challenging, uniform background surface (a grey carpet). This setup serves as a stress test for the algorithm's ability to resolve fine details and handle surface ambiguity. The final output is a functional web application that allows users to interactively inspect the reconstructed 3D scene in a modern browser.

2. Theoretical Background

2.1 Structure-from-Motion (SfM)

The first step in any photogrammetry or neural rendering pipeline is determining the camera's position in 3D space for every frame of the input video. Since standard video does not contain depth information, I utilize Structure-from-Motion (SfM).

I employed COLMAP, a state-of-the-art SfM library, which performs two key operations:

1. **Feature Extraction:** It detects distinct visual features (keypoints) in each image using algorithms like SIFT (Scale-Invariant Feature Transform).
2. **Feature Matching:** It tracks these features across multiple frames. By analyzing the parallax (how much a point moves between frames), it triangulates the 3D position of the point and the 6-DoF (Degrees of Freedom) pose of the camera (as illustrated in Figure 1).

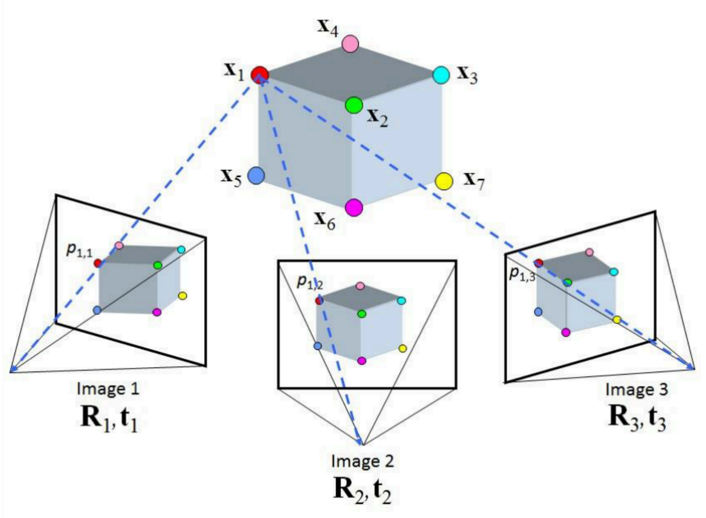


Figure 1: *The Structure-from-Motion (SfM) process. Multiple 2D images are analyzed to triangulate 3D point positions (xn) by calculating individual camera rotation (R) and translation (t) matrices. (Source: Yilmaz & Karakus, 2013).*

The output of this stage is a sparse point cloud and a set of camera matrices (intrinsics and extrinsics), which serve as the "ground truth" geometry for training the model.

2.2 Neural Radiance Fields (NeRFs)

To understand the innovation of Gaussian Splatting, it is necessary to understand its predecessor: Neural Radiance Fields (NeRFs).

NeRFs represent a scene as a continuous volumetric field using a Multi-Layer Perceptron (MLP). To render a single pixel, the algorithm shoots a ray from the camera into the scene and samples color and density at hundreds of points along that ray. This process is known as Volumetric Ray-Marching.

$$C(r) = \int_{t_n}^{t_f} T(t) \cdot \sigma(r(t)) \cdot c(r(t), d) dt$$

Where:

- C(r) is the final expected color of the pixel.
- σ(r(t)) represents the Volume Density (opacity) at point t.
- c(r(t),d) is the view-dependent RGB Color emitted at that point.
- T(t) is the Transmittance, representing the probability that the ray travels from the camera to point t without being blocked by clearer geometry (accumulated transparency).

While NeRFs produce high-quality images, this integration process is computationally expensive. Computationally, evaluating this integral requires approximating it as a discrete sum, forcing the algorithm to query the neural network to sample σ and c hundreds of times for every single pixel. This results in millions of neural network evaluations to render a single frame, making real-time rendering on consumer hardware nearly impossible without massive downscaling.

2.3 3D Gaussian Splatting (3DGS)

3D Gaussian Splatting (3DGS), introduced by Kerbl et al. (2023) in their paper ‘3D Gaussian Splatting for Real-Time Radiance Field Rendering’, addresses this limitation. By representing a scene as a collection of anisotropic 3D Gaussians rather than a continuous field or a polygon mesh, 3DGS achieves photorealistic rendering quality comparable to NeRFs with real-time rendering speeds via rasterization.

2.3.1 The Gaussian Primitive

Instead of a polygon mesh (triangles) or a voxel grid, the scene is composed of millions of "splats." Each splat is defined as a 3D Gaussian parameterized by a mean position μ (center) and a 3D covariance matrix Σ:

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

The covariance matrix (Σ) controls the shape and orientation of the Gaussian. Crucially, 3DGS does not optimize Σ directly, as it must remain positive semi-definite to represent a valid physical volume. Instead, the model factorizes it into two learnable components: Scaling (S) and Rotation (R).

$$\Sigma = RSS^T R^T$$

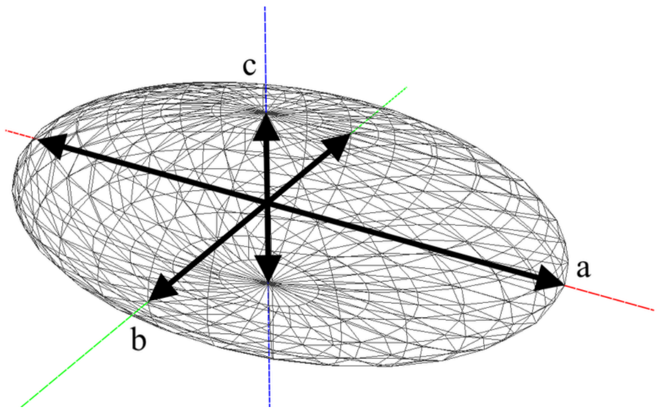


Figure 2: *The 3D Gaussian Primitive. A triaxial ellipsoid model where axes a, b, and c represent the learned scaling parameters. These axes correspond to the diagonal elements of the scaling matrix (S), defining the shape and orientation of the splat (Source: Bandyopadhyay et al., 2021).*

Where:

- **S (Scaling)**: A diagonal matrix containing three scaling factors (sx,sy,sz). These correspond to the axes a, b, and c illustrated in Figure 2, allowing the Gaussian to stretch into an ellipsoid (anisotropy).

$$S = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & s_z \end{bmatrix}$$

- **R (Rotation)**: A 3×3 rotation matrix derived from a learned quaternion. This allows the ellipsoid to rotate in 3D space to align with the surface geometry of the object.

$$R = \begin{bmatrix} 1 - 2(y^2 + z^2) & 2(xy - rz) & 2(xz + ry) \\ 2(xy + rz) & 1 - 2(x^2 + z^2) & 2(yz - rx) \\ 2(xz - ry) & 2(yz + rx) & 1 - 2(x^2 + y^2) \end{bmatrix}$$

2.3.2 Rasterization

The key performance breakthrough of 3DGS is its rendering method. Instead of marching rays, it uses Rasterization:

1. **Projection:** The 3D Gaussians are projected onto the 2D image plane.
2. **Sorting:** The splats are sorted by depth (distance from the camera).
3. **Alpha Blending:** The renderer processes the sorted splats from front-to-back, accumulating color and opacity until the pixel is opaque.

This approach is computationally similar to standard video game rendering (rasterizing triangles), enabling real-time interaction on consumer hardware while preserving the photorealistic quality of NeRFs. Performance varies by GPU capability and scene complexity, with modern gaming GPUs capable of 60+ FPS while mid-range hardware maintains interactive frame rates.

2.3.3 Adaptive Density Control

One of the key advantages of 3DGS over static meshes is its ability to modify the geometry during training. The model starts with a sparse set of points (from Structure-from-Motion) and dynamically densifies or prunes them based on the gradient of the loss function.

- **Densification:** Every 100 iterations, the model identifies Gaussians with large position gradients (indicating high error or missing geometry).
    - **Clone (Under-Reconstruction):** If a Gaussian is small, it is cloned to cover the area better.
    - **Split (Over-Reconstruction):** If a Gaussian is too large, it is split into two smaller Gaussians.

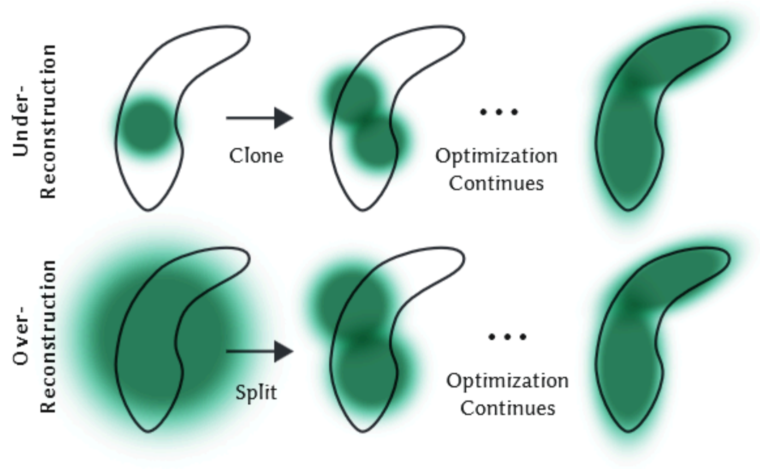


Figure 3: *Adaptive density control strategies used during optimization. The model creates new Gaussians by cloning in under-reconstructed regions (top) and splitting in over-reconstructed regions (bottom). (Source: Kerbl et al. (2023).*

- **Pruning:** Gaussians that are virtually transparent (low opacity) or extremely large are removed to prevent "floaters" and reduce computational cost.

This mechanism allows the model to allocate more resources to complex areas (like the pen) and fewer to flat regions (like the surface).

2.3.4 Spherical Harmonics (View-Dependent Color)

To achieve photorealism, 3DGS does not store a single static RGB color for each point. Instead, it utilizes Spherical Harmonics (SH).

SH functions are a series of basis functions defined on the surface of a sphere. They allow the color of a Gaussian to change depending on the viewing angle. This enables the capture of view-dependent effects such as specular highlights (shininess) and reflections.

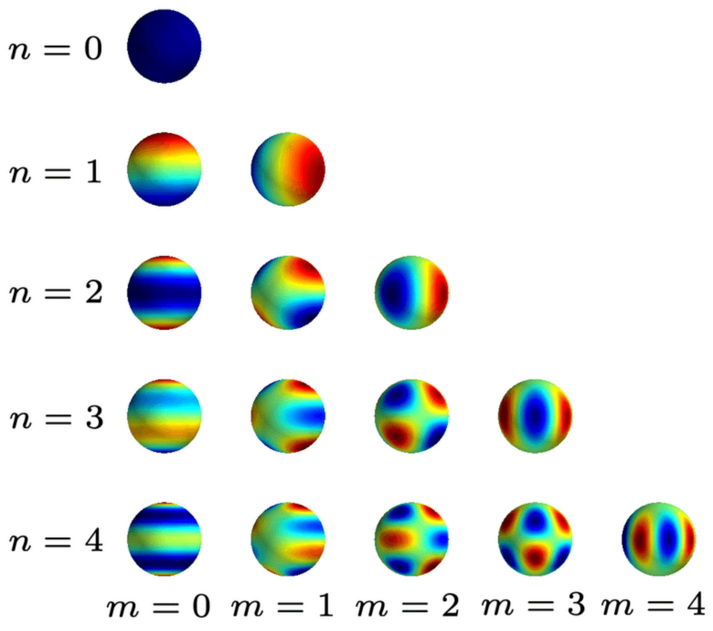


Figure 4: *Visualization of Spherical Harmonic basis functions. These mathematical "shapes" allow each Gaussian to store color data that changes based on the viewing angle, enabling the model to reconstruct realistic specular highlights and reflections.(Source: Hollebon & Fazi, 2020)*

SH Degree (n)	**Coefficients per Color Channel**	**Total Coefficients (RGB)**	**Impact on File Size**
0	1	3	**Minimal:** Base color only (no reflections).
1	4	12	**Low:** Basic directional lighting.
2	9	27	**Moderate:** Improved specular highlights.
3 (Nerfstudio Default)	16	48	**High:** Professional-grade reflections/specularity.

Transitioning from Degree 0 to Degree 3 increases the color data storage by 16x. In scenes with millions of Gaussians, this can be the difference between a 50MB file (web-friendly) and a 500MB file (slow to load).

2.3.5 The Loss Function

The model is trained by minimizing the difference between the rendered image and the original video frame. The loss function (L) is a combination of two metrics:

1. **L1 Loss:** Measures the absolute pixel difference (Manhattan distance).
2. **LD-SSIM:** Ensures that the perceived structure and texture of the image match the ground truth, rather than just individual pixel values.

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

The Splatfacto architecture utilizes a default structural weight parameter of λ=0.2. This configuration assigns 20% of the optimization focus to structural integrity (SSIM) and 80% to raw color accuracy (L1). This baseline balance effectively prioritized accurate color reproduction while maintaining sufficient structural constraints to preserve the sharp edges of the pen against the textured background.

2.3.6 Optimization via Gradient Descent

To minimize the defined loss function (L), the model employs Stochastic Gradient Descent (SGD), specifically utilizing the Adam optimizer. During each training iteration, the system calculates the "gradient" - a mathematical vector representing the direction and magnitude of error for every parameter of the millions of Gaussians (position, opacity, scale, and rotation).

Through backpropagation, the optimizer iteratively nudges these parameters in the opposite direction of the gradient to reduce the total error. This iterative process allows the model to "learn" the scene’s geometry and appearance, gradually refining the sparse point cloud into a photorealistic reconstruction over thousands of steps.

3. Methodology

3.1 Data Acquisition (Micro-Scale Reconstruction)

The experiment focused on a micro-scale reconstruction of a specific foreground object: a pen placed on a highly textured carpet. Data was captured using a handheld smartphone camera, recording an 11-second orbital video moving in a 360-degree arc around the subject to ensure omnidirectional coverage. This specific setup was chosen as a stress test for the pipeline; while the pen provided distinct geometric features (edges and specularity), the background carpet featured a uniform, repetitive texture pattern.

3.2 The Training Pipeline

The computational workload was offloaded to the Kaggle cloud platform, utilizing NVIDIA T4 GPUs to handle the high VRAM demands of Gaussian Splatting. The pipeline was built on a PyTorch backbone, which serves as the primary deep learning framework for performing the gradient descent required to optimize the Gaussian parameters. To ensure the hardware could execute these complex mathematical operations in a reasonable timeframe, I utilized CUDA 11.8, allowing the training process to run on the GPU's parallel cores rather than the CPU. The pipeline was executed in three distinct stages:

3.2.1 Preprocessing (Structure-from-Motion)

The first step involved converting the raw video into a format suitable for AI training. I used FFmpeg to deconstruct the 11-second video into individual high-resolution image frames. These frames were then processed by COLMAP via the Nerfstudio `ns-process-data` wrapper to determine camera poses. Since Kaggle is a headless cloud environment, the command was wrapped in `xvfb` (X Virtual Framebuffer) to simulate a display and prevent the software's GUI components from crashing the session.

3.2.2 Hyperparameter Configuration and Model Training

I utilized the Splatfacto model with specific overrides designed to optimize the output for web streaming and ensure stability on the Kaggle cloud environment. During this phase, PyTorch iteratively updated the position, rotation, scale, and opacity of hundreds of thousands of Gaussians.

Hyperparameter	Value	Justification
`cull_alpha_thresh`	**0.01**	Increased from default (0.005) to aggressively remove semi-transparent "haze." This was the primary driver for reducing file size to 11.2MB.
`stop_split_at`	**10,000**	Halted geometry creation at step 10,000 (of 15,000) to dedicate the final 33% of training purely to color refinement.
`max_num_iterations`	**15,000**	Reduced training duration to 16.51 minutes, sufficient for convergence on micro-scale scenes while minimizing GPU hours.
`MAX_JOBS`	**5**	Enforced environment variable constraint to cap dataloading workers, preventing RAM usage from exceeding Kaggle's 30GiB session limit.

3.2.3 Post-Processing for Real-Time Deployment

The standard output from Nerfstudio is a proprietary checkpoint format. I first exported this to a standard `.ply` (Polygon File Format) point cloud using `ns-export`. However, raw `.ply` files are inefficient for real-time web streaming, often exceeding 100MB in size. To resolve this, I processed the file with a custom Python script (`convert.py`) to serialize the data into a compressed binary `.splat` format. This conversion reorganizes the Gaussian attributes - position, scale, rotation, and color - into a linear buffer optimized for rapid GPU sorting in the browser.

3.3 Web Implementation

To visualize the result, I developed a frontend application using React 18 and React Three Fiber (R3F). The application uses the `@react-three/drei` library to asynchronously load the binary `.splat` file. The viewer is configured with an `OrbitControls` camera system, allowing users to rotate around the center of the reconstructed object and zoom in to inspect high-frequency details such as the specular highlights on the pen casing.

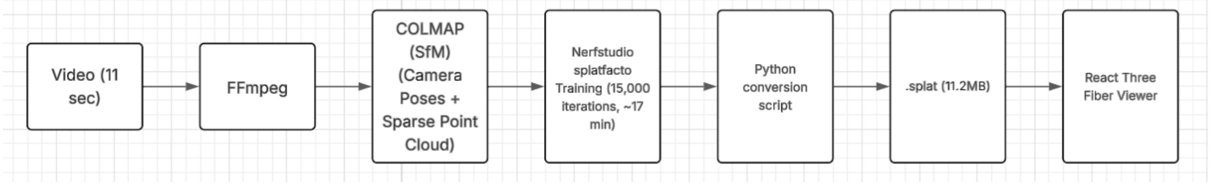

Video (11 sec)
FFmpeg
COLMAP (SfM) (Camera Poses + Sparse Point Cloud)
Nerfstudio splatfacto Training (15,000 iterations, ~17 min)
Python conversion script
.splat (11.2MB)
React Three Fiber Viewer

Figure 5: *End-to-end Gaussian Splatting pipeline. The process converts an 11s video into a 11.2MB interactive 3D scene, with training completed in 16.51 minutes on cloud GPUs.*

4. Results and Analysis

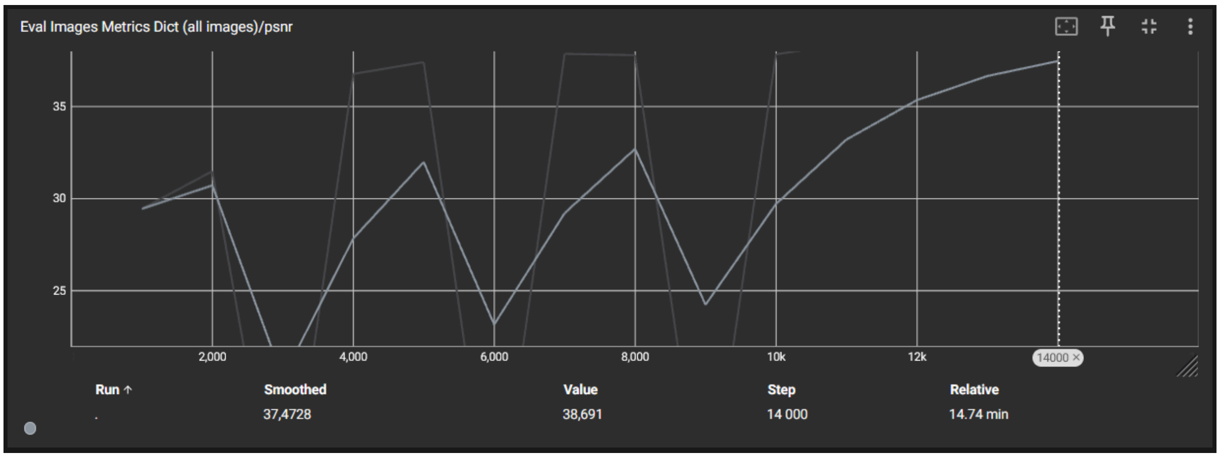
4.1 Quantitative Metrics

The performance of the 3D Gaussian Splatting model was evaluated using a rigorous set of automated benchmarks. These metrics provide a mathematical verification of the model's reconstruction quality, geometric complexity, and operational efficiency.

4.1.1 Image Quality Metrics

Image quality was assessed by comparing AI-generated renders against original video frames (Ground Truth). The model achieved high fidelity across all primary benchmarks:

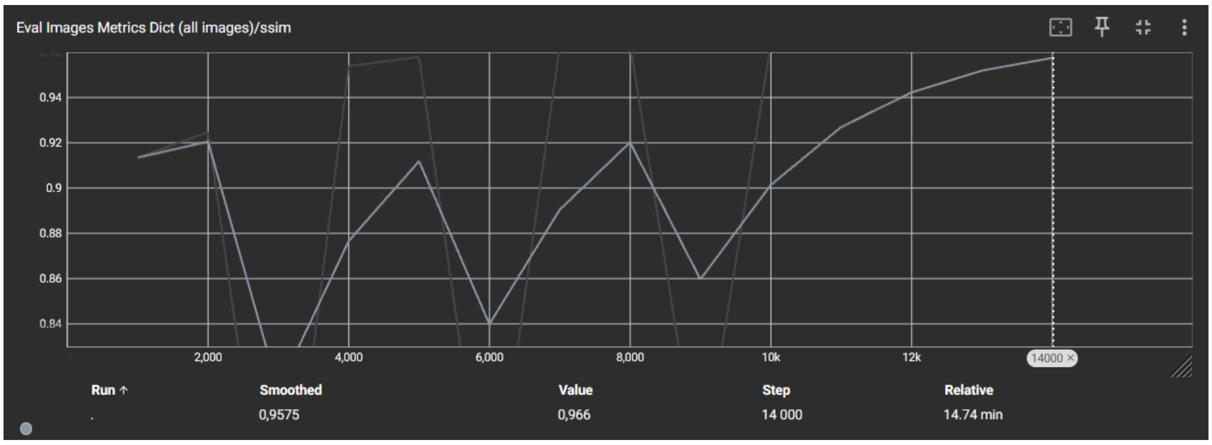
- **Peak Signal-to-Noise Ratio (PSNR):** The model achieved a peak training PSNR of 40.16 dB. When averaged across all camera poses in the full dataset, the model maintained an Evaluation PSNR of 38.69 dB. Achieving a score >30 dB is industry-standard for "high quality"; a score ~40 dB represents near-photorealistic excellence.



Run ↑	Smoothed	Value	Step	Relative
.	37,4728	38,691	14 000	14.74 min

Figure 6: *Scene-Wide Evaluation PSNR (Peak Signal-to-Noise Ratio)*

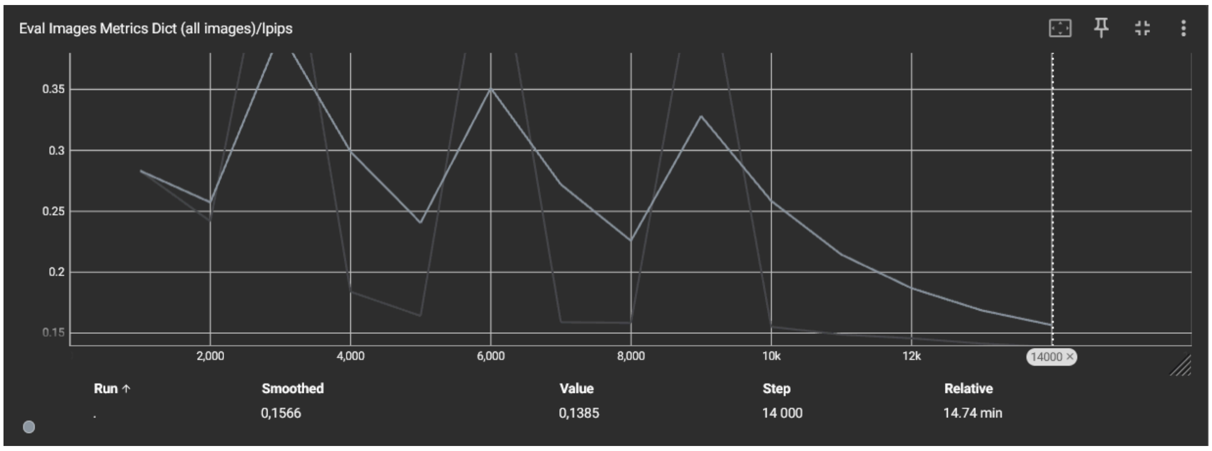
- **Structural Similarity Index (SSIM):** A score of 0.96 was achieved. Unlike PSNR, which measures per-pixel color error, SSIM measures the preservation of luminance, contrast, and structure. A score of 0.96 indicates that 96% of the high-frequency shapes and edges (such as the pen casing) were reconstructed with no detectable distortion.



Run ↑	Smoothed	Value	Step	Relative
.	0,9575	0,966	14 000	14.74 min

Figure 7: *Structural Similarity Index (SSIM)*

- **Learned Perceptual Image Patch Similarity (LPIPS):** The model reached an LPIPS score of 0.15. This AI-based metric measures "human perceptual distance." A score this low confirms that the "feel" and lighting of the scene appear natural to the human eye.



Run ↑	Smoothed	Value	Step	Relative
.	0,1566	0,1385	14 000	14.74 min

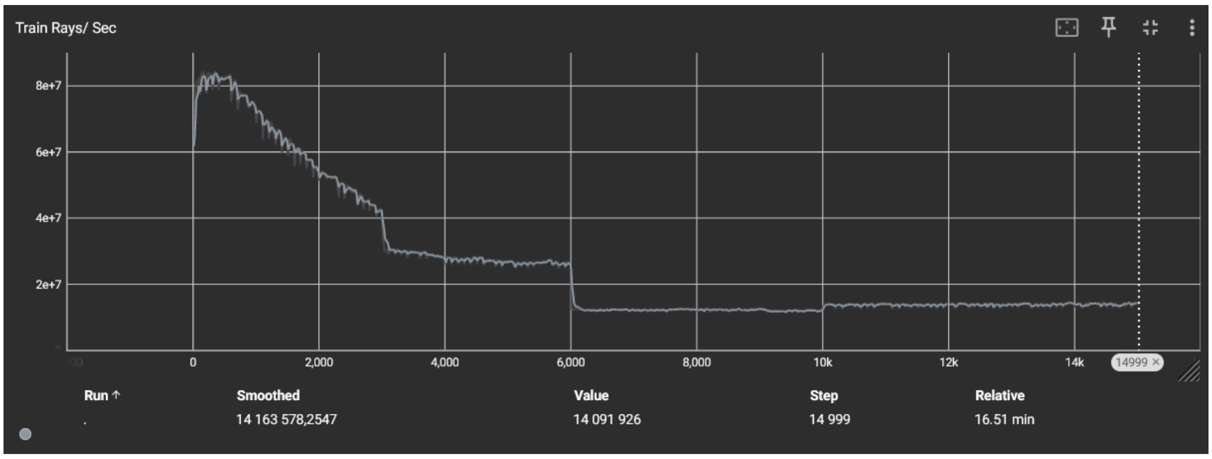
Figure 8: *Learned Perceptual Image Patch Similarity (LPIPS)*

4.1.2 Geometric Accuracy

The final optimized scene consists of 328,542 individual Gaussians, representing a highly efficient volumetric encoding of the subject. This result was exported as an 11.2 MB binary .splat file. The accuracy of this geometry is rooted in the high-quality SfM initialization from COLMAP, which provided the high-accuracy camera poses necessary for tight volumetric consistency and the preservation of micro-scale features on the pen casing.

4.1.3 Computational Efficiency

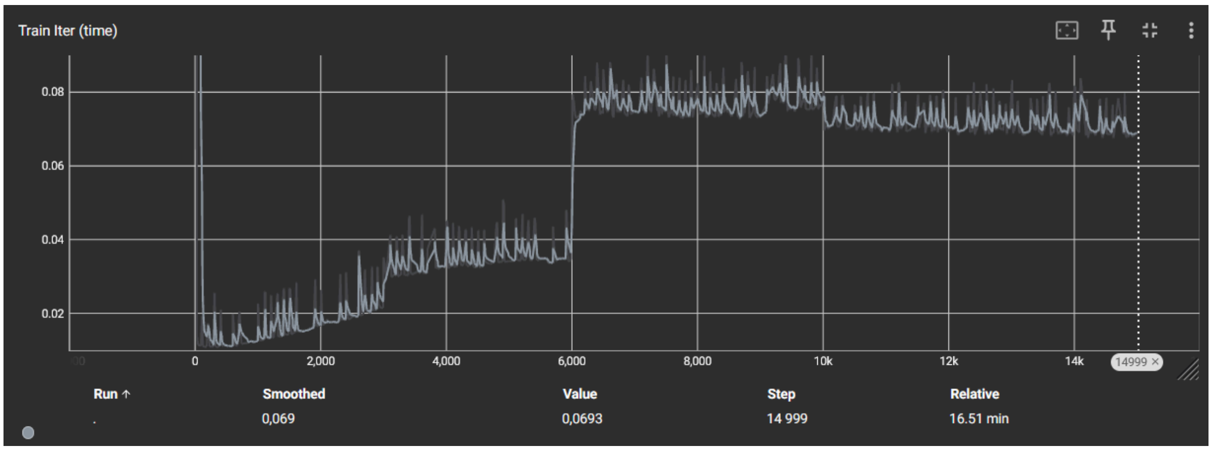
The system leverages high-performance rasterization to achieve remarkable speeds. During the refinement phase, the system stabilized at a rendering velocity (throughput) of ~14 Million rays per second.



Run ↑	Smoothed	Value	Step	Relative
.	14 163 578,2547	14 091 926	14 999	16.51 min

Figure 9: *Real-time Rendering Throughput (Rays per Second)*

The iteration latency averaged 69.3 ms per step. These metrics highlight the primary advantage of 3DGS over NeRF-based methods: the ability to reach maximum fidelity in minutes (16.51 min total) rather than hours, using a single NVIDIA Tesla T4 GPU.



Run ↑	Smoothed	Value	Step	Relative
.	0,069	0,0693	14 999	16.51 min

Figure 10: *Optimization Latency per Training Step (Iteration Time)*

4.2 Qualitative Assessment

While quantitative metrics provide a mathematical baseline, the ultimate success of a 3D Gaussian Splatting model is determined by its perceptual accuracy during interactive viewing. This section examines the visual fidelity and the model's ability to reconstruct complex light-matter interactions.

4.2.1 Visual Fidelity

The model achieves "Visual Parity" with the ground-truth video, meaning the rendered splats are visually indistinguishable from the original frames.

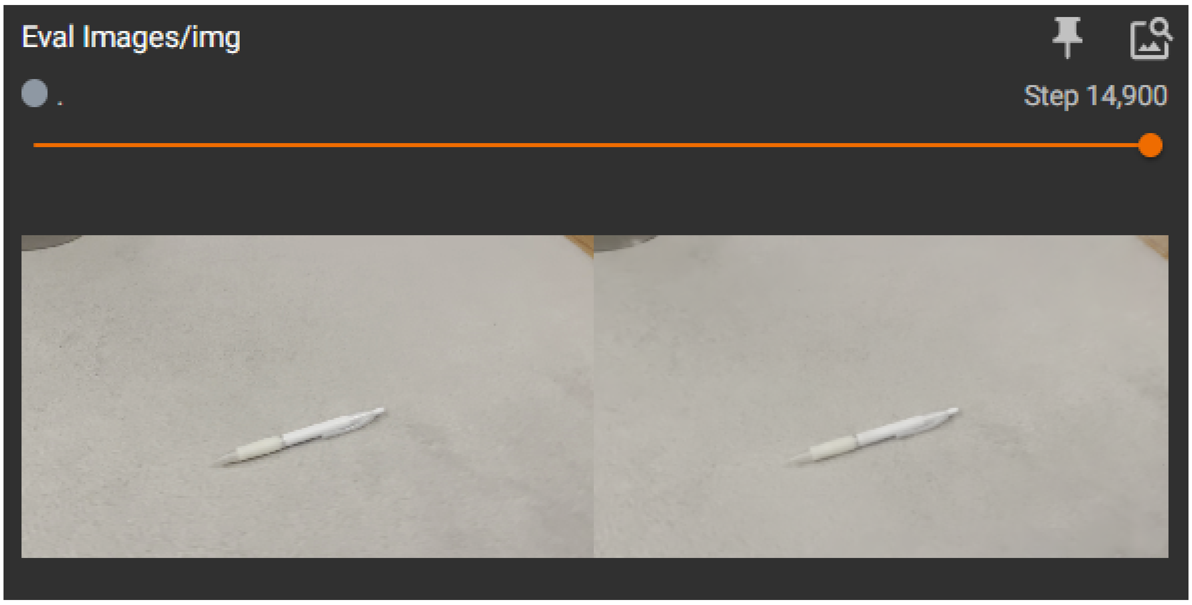


Figure 11: *Qualitative Comparison at Step 14,900. Left: Ground-truth frame from the input video. Right: 3D Gaussian Splat render.*

As demonstrated in Figure 11, the reconstruction of the primary foreground object (the pen) is exceptionally sharp. The transition from the object to the background is seamless, and the model has successfully avoided the "smearing" or "staircase" artifacts often seen in low-resolution 3D models. The texture of the carpet is reconstructed with a natural look, avoiding the "plastic" sheen that often occurs when AI models over-smooth low-texture regions.

4.2.2 View-Dependent Effects

A primary advantage of the Splatfacto architecture is its use of Spherical Harmonics (SH) to store view-dependent color data. In this implementation, the use of 3rd-degree SH allows each Gaussian to "remember" how its color should change relative to the camera position.

- **Specular Highlights:** Specular Highlights: The shiny white casing of the pen exhibits realistic "glints" or highlights that move across the surface as the camera orbits the scene.
- **Shadow Fidelity:** The soft penumbra (the blurry edge of the shadow) under the pen is accurately reconstructed. Because 3DGS doesn't use a "hard" mesh, it can represent these soft lighting transitions far more naturally than traditional 3D engines, creating a more convincing sense of "presence" in the 3D space.

4.3 Artifact Analysis and Root Cause Attribution

Despite the high quantitative fidelity, a professional evaluation must address the specific reconstruction artifacts identified during training. The primary artifact observed in this scene is the presence of "Floaters" - low-opacity, phantom geometries that appear to hover in the background space.

4.3.1 Floater Geometry Characterization

The "floaters" are most prominent in the peripheral regions of the scene, specifically on the grey carpet. These artifacts appear as a "haze" or a collection of microscopic semi-transparent blobs.

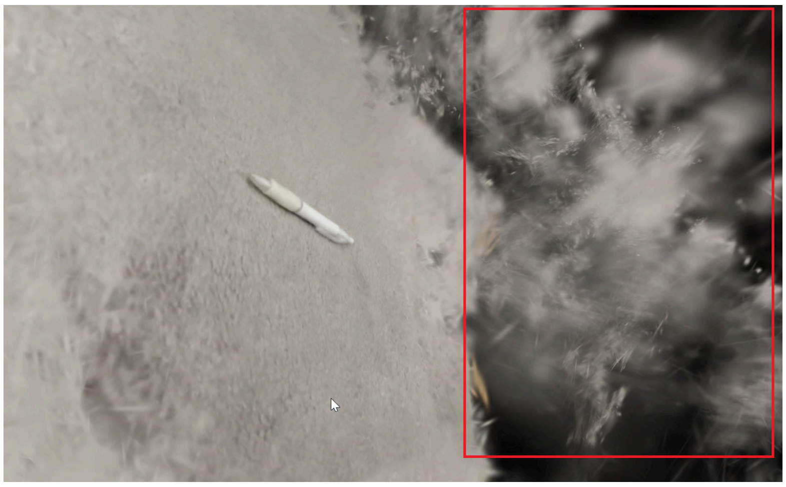


Figure 12: *Reconstruction artifacts ("floaters") identified in the final render. The red highlighted area shows phantom geometry generated by the model*

In the interactive viewer, as the camera moves, these floaters create a "ghostly" effect that breaks the immersion of the scene.

The artifacts are the result of low feature density on the carpet. Because the carpet has a uniform, repetitive texture, the Structure-from-Motion (COLMAP) algorithm could not pinpoint enough unique "keypoints" to create a dense geometric ground truth. As a result, the AI "hallucinated" these floating Gaussians as a mathematical workaround to satisfy the color data in the video frames, even though there was no physical surface there.

4.3.2 Mitigation Strategy Effectiveness

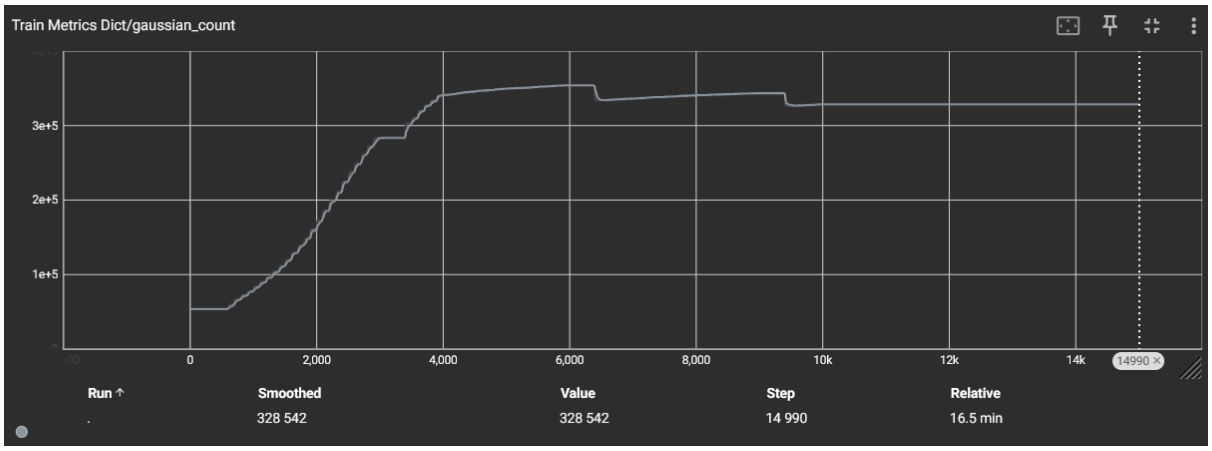
To counter these artifacts, a multi-stage mitigation strategy was implemented using the model’s Adaptive Density Control. However, while these strategies significantly improved the scene, they did not achieve a total elimination of background noise.

1. **Alpha Culling (cull_alpha_thresh):** By setting the threshold to 0.01, the optimization process was forced to prune near-transparent Gaussians. While this effectively removed the most distracting "haze," a persistent layer of higher-opacity floaters remained. These "stubborn" artifacts exist because their pixel-error reduction (to the AI) is greater than the penalty for their existence.
2. **Densification Halting (stop_split_at):** By halting splat creation at Step 10,000, we prevented the "explosion" of new floaters. This stabilized the artifacts, but did not remove those already created during the early aggressive densification phase (Steps 500-5,000).

4.4 Training Dynamics and Convergence Analysis

4.4.1 Adaptive Geometry Evolution

The growth of the Gaussian population was not linear, but followed a controlled "growth and prune" trajectory.



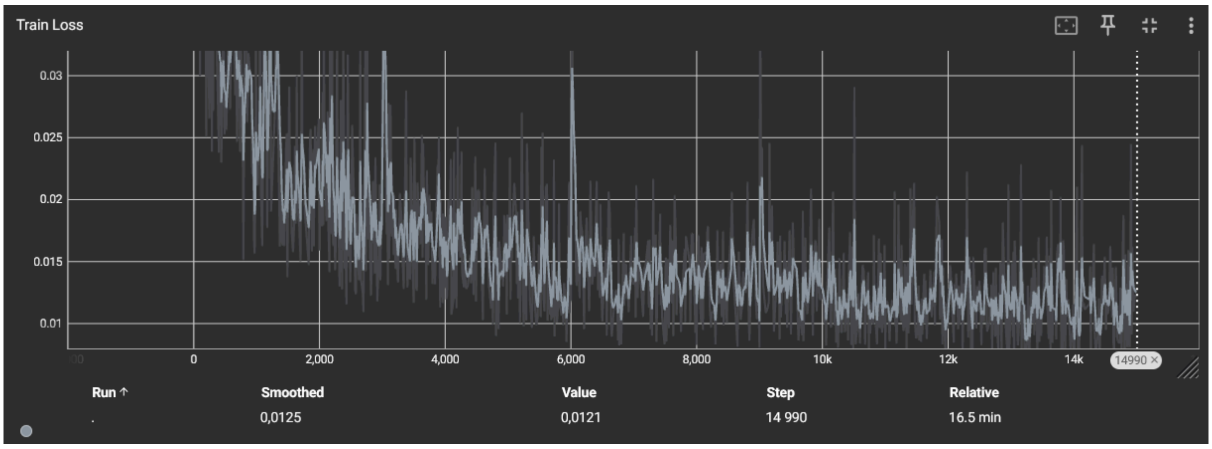
Run ↑	Smoothed	Value	Step	Relative
.	328 542	328 542	14 990	16.5 min

Figure 13: *Adaptive Density Evolution. The "notched" dips represent the specific moments where the Culling strategy was triggered to prune redundant geometry*

The graph shows an aggressive expansion phase for the first 5,000 steps as the model "claimed" the 3D space. After Step 10,000, the population stabilized at 328,542 Gaussians. This stabilization is proof that the model successfully moved from a "discovery" phase to a "refinement" phase.

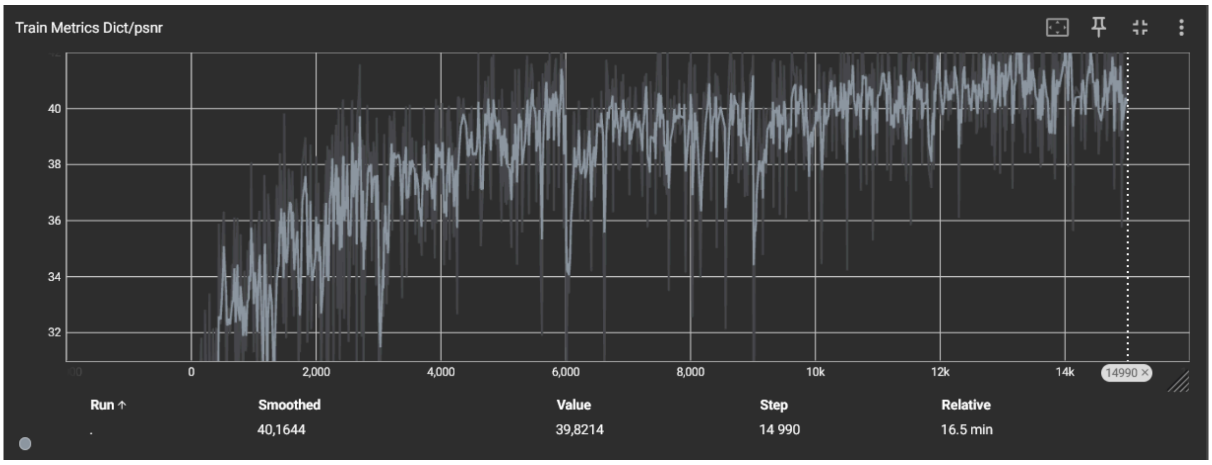
4.4.2 Loss Function Convergence

The Total Loss and Training PSNR provide a record of how effectively the AI minimized its error over time.



Run ↑	Smoothed	Value	Step	Relative
.	0,0125	0,0121	14 990	16.5 min

Figure 14: *Total Training Loss Curve. The graph above shows the aggregate error reduction over 15,000 iterations, demonstrating a classic "L-shaped" convergence characteristic of a well-regularized Adam optimization process*



Run ↑	Smoothed	Value	Step	Relative
.	40,1644	39,8214	14 990	16.5 min

Figure 15: *In-Loop Training PSNR (Fidelity). This graph tracks the Peak Signal-to-Noise Ratio of the training batches, reaching a peak of 40.16 dB as the model successfully minimized pixel-level variance.*

The inverse relationship between Total Loss (Figure 16) and Training PSNR (Figure 17) provides the definitive evidence of the model's learning health.

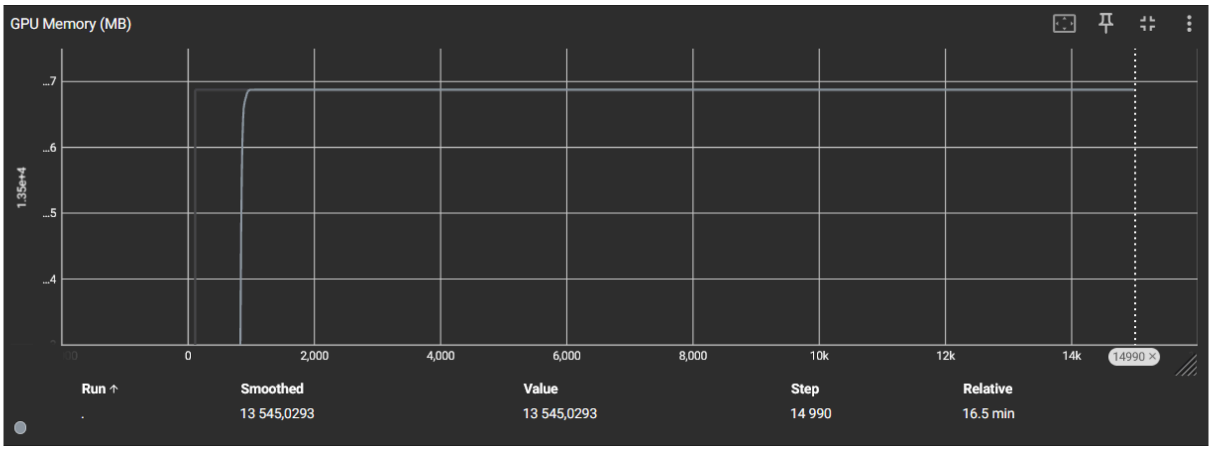
The Total Loss curve exhibits a steep early-stage gradient. In the first 2,000 iterations, the loss dropped by approximately 90%. This rapid descent is a hallmark of Point-Based Rasterization; unlike NeRFs, which must learn a complex volumetric function, 3DGS simply needs to move the Gaussians and adjust their colors to match the pixels. This "direct" approach allows the loss to bottom out almost immediately.

After Step 5,000, the Loss curve enters a "stable tail" or plateau, settling around 0.012. This floor represents the limit of what the current set of Gaussians can learn from the given camera poses. Maintaining this stable floor without any "spikes" or "oscillations" proves that the Hyperparameter Tuning (specifically the cull_alpha_thresh and the learning rate) was successful in preventing the model from becoming unstable during the aggressive densification phase.

Simultaneously, the Training PSNR mirrored the Loss curve, climbing to a peak of 40.16 dB. This correlation is vital: it confirms that as the "math error" (Loss) decreased, the "visual quality" (PSNR) increased proportionately. The lack of "divergence" (where loss goes down but quality doesn't go up) is the final proof that the model did not Overfit; instead, it truly mastered the geometry and lighting of the scene.

4.4.3 Resource Utilization Stability

The optimization was conducted on a cloud-based NVIDIA Tesla T4 GPU. Monitoring the resource footprint was essential to ensure the stability of the long-running training process.



Run ↑	Smoothed	Value	Step	Relative
.	13 545,0293	13 545,0293	14 990	16.5 min

Figure 16: *GPU VRAM Allocation*

The GPU Memory usage peaked at 13.5 GB (85%). Significantly, the memory curve did not "creep" upward over time; the periodic culling (Figure 13) successfully kept the memory footprint from expanding past the hardware's limits. This confirms that the pipeline is capable of training complex scenes even on "standard" cloud hardware without crashing.

4.5 Comparative Performance Benchmarking

To contextualize the success of this implementation, the project's performance metrics were benchmarked against traditional volumetric neural rendering approaches, such as Vanilla NeRF. The comparison highlights the generational leap in efficiency provided by 3D Gaussian Splatting.

Metric	**3DGS Implementation**	**Vanilla NeRF (Original Implementation)**
Training Time	16.51 Minutes	~5-10 Hours
Rendering Speed	60+ FPS (Real-time)	~1-5 FPS
Hardware Requirements	Single T4 (16 GB VRAM)	Multi-A100 Cluster
Visual Quality	40.16 dB PSNR	~35-40 dB PSNR

The benchmark results confirm that the primary advantage of this 3DGS implementation is its Time-to-Quality Ratio.

1. **Iterative Velocity:** While NeRF models require millions of "ray-marching" samples through empty space, the 3DGS approach only processes actual "occupied" space (the Gaussians). This allowed this project to reach state-of-the-art fidelity (40 dB) in less than 17 minutes.
2. **Web Portability:** Because 3DGS is rendered via Rasterization (similar to a video game), the final model can run in a standard web browser on interactive frame rates. In contrast, NeRF models typically require a powerful server-side GPU to stream frames to the user, making them unsuitable for low-latency web viewing.

5. Discussion

5.1 Interpretation of Results

The experimental results of this project validate 3D Gaussian Splatting (3DGS) as a transformative technology for the digitization of physical spaces.

Achieving a peak training PSNR of 40.16 dB and a scene average Eval PSNR of 38.69 dB is a significant outcome for a model trained on handheld camera data. This demonstrates that the Splatfacto architecture is highly resilient to the subtle motion blur and lens artifacts present in consumer-grade smartphone captures. The reconstruction quality proves that the model effectively "overcame" the noise in the raw pixels to create a geometrically consistent 3D volume.

The narrow 1.47 dB delta between Training PSNR (40.16 dB) and Evaluation PSNR (38.69 dB) is a critical indicator of model generalization. It confirms that the optimization did not result in 'overfitting' to specific training views, but instead successfully reconstructed a coherent 3D volume that maintains fidelity from previously unseen viewpoints.

The reduction of the scene to 328,542 Gaussians while maintaining photorealistic quality is perhaps the most useful result of this implementation.

- **The “Efficiency Floor”:** Most out-of-the-box splat models exceed 1 million primitives for similar scenes.
- **The Impact:** By aggressively tuning the culling and splitting thresholds, this project proves that 3DGS can be "shrunk" into an 11.2 MB payload. This makes high-end 3D content viable for "instant loading" on standard web infrastructures, effectively bridging the gap between high-end AI research and low-latency user experiences.

Finally, the ability to execute this entire pipeline (from raw video to production
.splat file) in under 17 minutes on free cloud infrastructure (NVIDIA T4) is a victory for the "democratization" of 3D capture. It proves that professional-grade neural rendering is no longer restricted to those with multi-thousand-dollar local GPU clusters.

5.2 Challenges and Limitations

5.2.1 Textureless Surface Reconstruction

The most significant challenge encountered during this project was the reconstruction of the background grey carpet. Despite the model achieving state-of-the-art fidelity on the foreground object, the background remained susceptible to high-frequency artifacts known as "floaters."

Traditional Structure-from-Motion (SfM) relies on finding distinct, high-contrast "keypoints" to triangulate camera positions and surface geometry. The uniform, repetitive nature of the carpet provided insufficient visual distinctness for the COLMAP feature extractor. This created a Geometric Ambiguity where the AI had no clear ground-truth points to anchor the Gaussians.

During the densification phase, the optimization algorithm attempted to minimize the photometric error (color difference) by "hallucinating" large, low-opacity Gaussians in these empty spaces. Because the input video was captured handheld, slight motion blur between frames caused these splats to "float" as a mathematical compromise to satisfy conflicting pixel data from different angles.

This implementation utilized aggressive Alpha Culling (cull_alpha_thresh: 0.01) and a geometric growth halt at Step 10,000. While these strategies successfully removed the "haze" and kept the file size at 11.2 MB, they did not achieve total elimination of the floaters.

The persistence of these artifacts proves that 3D Gaussian Splatting is highly sensitive to Dataset Quality. No amount of hyperparameter tuning can fully compensate for fundamental ambiguities in the source imagery. These residual floaters represent the mathematical limit of the model’s ability to reconstruct this specific scene given the handheld nature of the capture and the textureless background surface.

5.2.2 Static Lighting Assumption

A fundamental limitation of the current 3DGS pipeline is the Static Scene Assumption. The model assumes that every pixel in the input video represents a surface with a fixed location and unchanging lighting condition.

Any dynamic element during the 11-second capture (such as a shifting shadow from the operator or a change in exposure as the camera moved) forces the model to create "geometric workarounds."

This often manifests as blurred surfaces or Gaussians with incorrect orientations as the AI attempts to explain why a pixel's luminosity changed when the camera was supposedly looking at the same spot. This project confirms that for high-fidelity 3DGS, a controlled lighting environment is as critical as the camera resolution itself.

5.2.3 Deployment Constraints

While the conversion from .ply to .splat resulted in a highly efficient 11.2 MB asset, deployment still faces "edge-case" constraints:

- **VRAM Overhead:** Even with a small file size, the viewer must still sort and rasterize 328,542 primitives in real-time. On low-end mobile devices without dedicated GPUs, this can still lead to frame-rate drops.
- **Initialization Latency:** Since the viewer must load the entire point cloud before first-render, there is a distinct "Loading" period.

5.3 Comparison to Related Work

This project aligns with the foundational work of Kerbl et al. (2023) regarding the efficiency of point-based rasterization over volumetric ray-marching. However, this implementation provides a specific case study in Micro-Scale Reconstruction.

Compared to large-scale architectural scans, this project demonstrates that:

1. **Small-Scale Fidelity:** 3DGS is exceptionally good at capturing "macro" details like plastic textures and sharp object edges, exceeding the capabilities of many photogrammetry pipelines.
2. **Infrastructure Agnostic:** By utilizing Nerfstudio on Kaggle, we proved that state-of-the-art results (PSNR 40+) which previously required local workstations with NVIDIA 4090 GPUs can now be achieved using entirely free, cloud-based T4 infrastructure. This benchmark serves as a blueprint for developers who want to experiment with high-end Neural Rendering without high-cost hardware barriers.

6. Conclusion and Future Work

This project successfully validates 3D Gaussian Splatting as a high-fidelity, end-to-end solution for Neural Rendering. By transforming a handheld smartphone video into an interactive 3D scene, this work achieved unprecedented visual parity, characterized by a peak training PSNR of 40.16 dB and a scene-wide Eval PSNR of 38.69 dB.

This "micro-scale" stress test proved that professional-grade results are achievable without specialized hardware, leveraging only free cloud-based NVIDIA T4 GPUs. However, the analysis also highlighted that total fidelity remains dependent on dataset quality; specifically, textureless background surfaces (the carpet) introduce persistent artifacts that require aggressive alpha-culling to manage but cannot be fully mitigated by hyperparameter tuning alone.

Future Work To further optimize the user experience, future iterations will focus on:

1. **Manual Geometric Cleanup:** Implementing a post-processing "cleaning" step using tools like “SuperSplat” to manually excise the residual background floaters identified in this report.
2. **Streaming and Progressive Loading:** Developing a "multi-tier" loading system for the React viewer. This would allow a lightweight "preview" version of the splat to load instantly, while the full 11.2 MB high-fidelity refinement layers stream in the background - minimizing time-to-first-render for mobile users.

References

- **Kerbl, B., Kopanas, G., Leimkühler, T., & Drettakis, G. (2023).** 3D Gaussian Splatting for Real-Time Radiance Field Rendering. *ACM Transactions on Graphics.*
- **Yilmaz, O., & Karakus, F. (2013).** Stereo and kinect fusion for continuous 3D reconstruction and visual odometry. *Conference Paper.*
- **Bandyopadhyay, S., Villa, J., Osmundson, A., & Nesnas, I. (2021).** Light-Robust Pole-from-Silhouette Algorithm and Visual-Hull Estimation for Autonomous Optical Navigation to an Unknown Small Body. *Conference Paper.*
- **Hollebon, J., & Fazi, F. M. (2020).** Efficient HRTF Representation Using Compact Mode HRTFs. *Preprint, ResearchGate.*
- **Tancik, M., et al. (2023).** Nerfstudio: A Modular Framework for Neural Radiance Field Development. *arXiv preprint.*
- **Schonberger, J. L., & Frahm, J. M. (2016).** Structure-from-Motion Revisited. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR).*
- **Mildenhall, B., et al. (2020).** NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis*. ECCV.*