



Gaussian Splatting: Numerical Foundations and Performance Evaluation

Nicolas Duque Molina

Facultad de Ciencias
Departamento de Matemáticas
Sede Bogotá, Colombia
2025

Gaussian Splatting: Numerical Foundations and Performance Evaluation

Nicolas Duque Molina

Tesis presentada como requisito parcial para optar por el título de:
Pregrado - Trabajo de Grado

Director:

Dr. Juan Carlos Galvis Arrieta
Profesor Titular - Departamento de Matemáticas
Facultad de Ciencias
Universidad Nacional de Colombia — Sede Bogotá

Codirectora:

Diana Patricia Aldana Moreno
Magíster en Matemáticas Aplicadas — Computación Gráfica, IMPA
Investigadora independiente

Línea de investigación:

Computación Gráfica

Grupo de investigación:

Semillero de Cálculo Científico - Línea de Computación Espacial

Universidad Nacional de Colombia — Sede Bogotá
Facultad de Ciencias
Departamento de Matemáticas

2025

If it looks like computer graphics, it is not good computer graphics. - Si se ve como gráficos hechos por computadora, no son buenos gráficos hechos por computadora.

Jeremy Birn

Declaración

Me permito afirmar que he realizado esta monografía de trabajo de grado de manera autónoma y con la única ayuda de los medios permitidos y no diferentes a los mencionados el presente texto. Todos los pasajes que se han tomado de manera textual o figurativa de textos publicados y no publicados, los he reconocido en el presente trabajo. Ninguna parte del presente trabajo se ha empleado en ningún otro tipo de tesis.

Sede Bogotá, 12 de Diciembre 2025

A handwritten signature in black ink, appearing to read 'Nicolás Duque Molina', written over a horizontal line.

Nicolás Duque Molina

Agradecimientos

Quiero dedicarle este trabajo a mi abuela Christina, mujer dulce y fuerte a la vez. Muchas gracias por cuidarme desde que tengo memoria y por todas las empanadas y chucherías que compartes conmigo cada vez que nos vemos.

Este logro pertenece en gran medida a mis padres. A Yasmín y Juan Carlos, que con su constante trabajo y dedicación han hecho posible que estudiara lo que quisiera, cuando quisiera, donde quisiera. Su apoyo incondicional, su firmeza, sus consejos y anécdotas, y su gran amor son el gran impulso que me permite seguir caminando a diario. Gracias por prestarme tanta atención cuando decido compartir conocimientos de los que ni yo mismo estoy seguro a veces.

Quiero agradecer a Laura, tanto por su labor como hermana mayor, silenciosa pero presente, como por abrir mi mente a la ciencia y al mundo de la investigación. Gracias a Dani, que con todo su amor y entendimiento me ha sostenido durante momentos difíciles y elevado durante momentos alegres. Gracias a mi tía Jenifer, que siempre se ha interesado profundamente por mi desarrollo y siempre está allí para celebrar mis logros.

No puedo olvidar a todos mis amigos que han sido muchas veces los únicos que me han mantenido de pie cuando la vida hace que el suelo tiemble. Gracias Marcos, por ser realmente mi hermano menor y secuaz en cualquier aventura que me proponga. Gracias María Paz, Samuel y Julián por sacarme tantas risas y hacer de ir a la universidad una experiencia tan agradable. Gracias Matthew, Tato, Vytis y España por estar desde siempre y estar cerca incluso cuando están lejos. Gracias Danica, Armin y David por recibirme como en casa en tierras desconocidas y hacerme encontrar otra familia que no creía posible.

Un académico solo es tan bueno como todos los profesores que lo apoyaron en su camino. Muchas gracias Profesor Juan Galvis por darme la oportunidad y confianza de profundizar en mis intereses y compartirlos con la comunidad. Muchas gracias a Diana, por aportar tantas ideas y aclarar la tarea enorme que es producir un texto académico. Muchas gracias a los profesores Jaime Rueda y Héctor Carranza, por plantar en mí la semilla de la curiosidad por la ciencia, las matemáticas y la enseñanza.

Finalmente, agradezco a mis dos compañeros que sin palabras siempre estuvieron a mi lado. Gracias Tango, por crecer conmigo y ser tan noble maestro. Gracias Vito, por tenerme tanto cariño y recordarme lo que es realmente importante en la vida.

Gracias a todos ustedes y también a aquellos que no alcancé a mencionar. Les prometo que habrá más por celebrar en el futuro.

Acknowledgements

I want to dedicate this work to my grandmother Christina, a woman who is both sweet and strong. Thank you very much for taking care of me since I can remember and for all the empanadas and treats you share with me every time we see each other.

This achievement belongs in large part to my parents. To Yasmín and Juan Carlos, who, with their constant work and dedication, have made it possible for me to study whatever I wanted, whenever I wanted, wherever I wanted. Their unconditional support, their firmness, their advice and anecdotes, and their great love are the significant drive that allows me to keep walking daily. Thank you for paying so much attention to me when I decide to share knowledge of which even I am not always sure.

I want to thank Laura, both for her work as a big sister, silent but present, and for opening my mind to science and the world of research. Thanks to Dani, who with all her love and understanding has sustained me during difficult moments and elevated me during joyful ones. Thanks to my aunt Jenifer, who has always taken a deep interest in my development and is always there to celebrate my achievements.

I cannot forget all my friends who have often been the only ones to keep me standing when life makes the ground tremble. Thank you Marcos, for truly being my younger brother and accomplice in any adventure I propose. Thank you María Paz, Samuel, and Julián for giving me so many laughs and making going to university such a pleasant experience. Thank you Matthew, Tato, Vytis, and España for being there since the beginning and staying close even when you are far away. Thank you Danica, Armin, and David for welcoming me as if I were home in unknown lands and helping me find another family that I didn't think was possible.

A scholar is only as good as all the professors who supported them on their way. Thank you very much Professor Juan Galvis for giving me the opportunity and confidence to delve into my interests and share them with the community. Thank you very much to Diana, for contributing so many ideas and clarifying the enormous task of producing an academic text. Thank you very much to Professors Jaime Rueda and Héctor Carranza, for planting in me the seed of curiosity for science, mathematics, and teaching.

Finally, I am grateful to my two companions who were always by my side without words. Thank you Tango, for growing up with me and being such a noble teacher. Thank you Vito, for having so much affection for me and reminding me what is truly important in life.

Thank you to all of you and also to those I failed to mention. I promise you there will be more to celebrate in the future.

List of Symbols and Abbreviations

Abbreviation	Term
% Cov.	Coverage Percentage
3DGS	3D Gaussian Splatting [1]
BGD	Batch Gradient Descent
FPS	Frames per Second
GD	Gradient Descent
GS	Gaussian Splatting
Inp. Imgs.	Input Images
Iter.	Iterations
LPIPS	Learned Perceptual Image Patch Similarity
P-Thr.	Percentile of Laplacian Variance Threshold
PSNR	Peak Signal-to-Noise Ratio
Rec. Imgs.	Images used for Reconstruction
Reg. Imgs.	Registered Images
SGD	Stochastic Gradient Descent
SSIM	Structural Similarity Index
T.D.	Training Duration

Abstract

Gaussian Splatting: Numerical Foundations and Performance Evaluation

3D Gaussian Splatting (3DGS) offers a powerful alternative to traditional photogrammetry, yet its sensitivity to input data and hyperparameters remains underexplored. This work systematically evaluates the 3DGS pipeline to analyze the impact of sampling rates and training parameters on reconstruction fidelity and efficiency. Contrary to the assumption that higher data density linearly improves quality, our results reveal a non-monotonic relationship where excessive redundancy destabilizes the Structure-from-Motion process. Furthermore, we demonstrate that exhaustive optimization strategies are inefficient; extended training durations lead to “geometric bloat”, a phenomenon where model size increases disproportionately to visual gains and compromises real-time performance on consumer hardware. We also validate the necessity of anisotropic covariance for preserving high-frequency details to resolve discrepancies between quantitative metrics and perceptual quality. Ultimately, this work identifies an optimal configuration that balances visual fidelity with storage and rendering constraints to provide a validated framework for the efficient deployment of 3DGS.

Keywords: 3D Gaussian Splatting, Photogrammetry, Structure-from-Motion (SfM), Sensitivity Analysis, Novel View Synthesis

Resumen

Gaussian Splatting: Fundamentos Numéricos y Evaluación de Rendimiento

El 3D Gaussian Splatting (3DGS) ofrece una alternativa potente a la fotogrametría tradicional; sin embargo, su sensibilidad a las características de los datos de entrada y a la configuración de hiperparámetros permanece poco explorada. Este trabajo evalúa sistemáticamente el flujo de trabajo de 3DGS para analizar el impacto de las tasas de muestreo y los parámetros de entrenamiento sobre la fidelidad de la reconstrucción y la eficiencia computacional. Contrario a la suposición de que una mayor densidad de datos mejora linealmente la calidad, nuestros resultados revelan una relación no monótona en la que la redundancia excesiva desestabiliza el proceso de Structure-from-Motion. Además, demostramos que las estrategias de optimización exhaustiva resultan ineficientes; los tiempos de entrenamiento extendidos conducen a una “inflación geométrica”, un fenómeno donde el tamaño del modelo aumenta de manera desproporcionada respecto a las ganancias visuales y compromete el rendimiento en tiempo real en hardware de consumo. Asimismo, validamos la necesidad de la covarianza anisotrópica para preservar los detalles de alta frecuencia y resolver las discrepancias entre las métricas cuantitativas y la calidad perceptual. Finalmente, este trabajo identifica una configuración óptima que equilibra la fidelidad visual con las restricciones de almacenamiento y renderizado para proporcionar un marco validado para el despliegue eficiente de 3DGS.

Palabras clave: 3D Gaussian Splatting, Fotogrametría, Estructura a partir del Movimiento (SfM), Análisis de Sensibilidad, Síntesis de Vistas

List of Figures

2-1	Isocontours of an example Multinormal Function. Based on [2].	10
2-2	Representation of the first 4 SH bands on a unit sphere. Based on [3, 4, 5]	12
3-1	Diagram of the Rasterization Rendering Pipeline. Based on [3].	17
3-2	2D Representation of the ModelView Transform. Based on [3].	18
3-3	2D Representation of View Volume Transforms of Perspective Transformations.	19
3-4	Fragments generated from a triangle. Based on [3].	20
4-1	Overview of the COLMAP pipeline implementation of Incremental SfM. Figure from [6].	25
4-2	Isosurfaces of a 3D Gaussian with covariance matrix Σ	26
6-1	Visual progression of Estatuas Scene. Top: Full scene geometry. Bottom: High-frequency texture details.	38
6-2	Visual progression of Sala Profes Scene. Top: Full scene geometry. Bottom: High-frequency texture details.	39
6-3	Visual progression of UNAL Scene. Top: Full scene geometry. Bottom: High-frequency texture details.	40
6-4	Visual progression of Virgen Scene. Top: Full scene geometry. Bottom: High-frequency texture details.	41
7-1	Impact of Extraction Rate on quality metrics at P_{Base}	46
7-2	Impact of P-Threshold on quality metrics at the optimal FPS extraction rate.	47
7-3	Ablation results for Custom Dataset Experiments. The top figure shows the trade-off between Weighted Quality Score and Training Duration. The bottom figure shows the trade-off between Weighted Quality Score and Model Size (Memory).. . . .	48
7-4	Ablation results for the 3DGS Dataset Experiments. The top figure shows the trade-off between Weighted Quality Score and Training Duration. The bottom figure shows the trade-off between Weighted Quality Score and Model Size (Memory).. . . .	49
7-5	Mean FPS per testing platform, averaged across all scenes.	49
7-6	1% Low FPS per testing platform, averaged across all scenes.	50
A-1	Visual progression of the drjohnson scene from the 3DGS dataset.	53
A-2	Visual progression of the playroom scene from the 3DGS dataset.	54
A-3	Visual progression of the train scene from the 3DGS dataset.	54
A-4	Visual progression of the truck scene from the 3DGS dataset.	55

B-1	Select Renders from Experiments 2 and 3 from the Custom Dataset.	56
------------	--	----

List of Tables

5-1	Hardware Configuration for Model Optimization.	31
5-2	Hardware configurations sorted by theoretical GPU compute performance (FP32 TFLOPS).	32
5-3	Baseline Gaussian Splatting training configurations.	35
5-4	Ablation configurations disabling scaling and/or rotation.	35
6-1	Results of the initial P value search for the captured dataset.	42
6-2	Systematic Search for Optimal Frame Rate ($FPS_{Optimal}$) Using Baseline Quality	42
6-3	Systematic Search for Optimal Threshold ($P_{Optimal}$) Using $FPS_{Optimal}$	43
6-4	Performance Metrics, Size and Training Duration of the Models generated from the Custom Dataset.	43
6-5	Performance Metrics, Size and Training Duration of the Models generated from the 3DGS Dataset, Sourced from Tanks and Temples [7] and Deepblending [8].	44
6-6	Performance Metrics for Models generated from the Custom Dataset.	44
6-7	Performance Metrics for Models generated from the 3DGS Dataset.	45

Contents

Agradecimientos	ii
List of Symbols and Abbreviations	iv
Resumen	vi
List of figures	vii
List of tables	ix
Table of contents	x
1 Introduction	1
2 Mathematical Background	3
2.1 Projective Geometry	3
2.1.1 Definition of Projective Space	3
2.1.2 Homogeneous Representation of Geometrical Elements	4
2.1.3 Projective Transformations	5
2.2 Fundamentals of the Multivariate Normal Distribution	9
2.2.1 Mahalanobis Distance	9
2.2.2 Multivariate Normal Distributions	9
2.2.3 Geometric Interpretation	10
2.3 Spherical Harmonics	11
2.3.1 Recursive Evaluation of SH Basis Functions	11
2.3.2 Projection and Reconstruction	12
2.3.3 Properties of SH Projections	13
2.4 Gradient Descent	13
2.4.1 Stochastic Gradient Descent	14
3 Foundations of 3D Rendering	16
3.1 Rasterization-based Rendering	16
3.1.1 Application Stage	16
3.1.2 Geometry Processing	17
3.1.3 Rasterization	20
3.1.4 Pixel Processing	21

3.2	Rendering beyond Polygonal Meshes	21
3.2.1	Volumetric Rendering	22
3.2.2	Point-Based Rendering	22
3.2.3	Radiance Fields and Image-Based Scene Reconstruction	22
4	Gaussian Splatting	24
4.1	SfM Point Clouds	24
4.2	Definition of the Rendering Primitive	25
4.2.1	Color Representation via Spherical Harmonics	26
4.2.2	Gaussian Projection	27
4.3	Splat Rendering	27
4.3.1	Splat Culling and Sorting	27
4.3.2	Forward Rendering Pass	28
4.4	Training and Optimization	28
4.4.1	Gradient Computation and Backpropagation	29
4.4.2	Primitive Restructuring	30
5	Methodology	31
5.1	Experimental Setup	31
5.1.1	Model Optimization Environment	31
5.1.2	Performance Benchmark Environments	32
5.2	Data Used for Evaluation	32
5.2.1	3DGS Dataset	32
5.2.2	Proprietary Scene Collection	33
5.3	Training Evaluation	35
5.4	Performance Evaluation	36
5.5	Image Quality Metrics	36
6	Results	37
6.1	Qualitative Visual Analysis	37
6.2	Quantitative Reconstruction Results	37
6.3	Performance Benchmarks	44
7	Discussion and Final Comments	46
7.1	Dynamics of Input Sampling and Pre-processing	46
7.2	Sensitivity of Reconstruction Fidelity to Hyperparameter Tuning	47
7.3	Performance Benchmarks	48
7.4	Limitations	50
7.5	Future Work	50
7.6	Conclusions	51
A	Comparisons of 3DGS Dataset Scenes	53
B	Qualitative Analysis of Suboptimal Reconstructions	56
	Bibliography	57

1 Introduction

Akenine-Möller et al. [3] posit that triangle-mesh rendering remains the dominant paradigm for real-time three-dimensional scene representation. However, this approach requires inherent computational overheads, often necessitating specialized hardware to achieve photorealism or the management of discrete Level-of-Detail (LOD) hierarchies to sustain frame rates [3]. These limitations motivated Levoy and Whitted [9] to propose point-based rendering in 1985 as a means to decouple geometric complexity from display resolution. Despite its theoretical promise, point-based rendering remained impractical for real-time applications until hardware capabilities matured nearly fifteen years later [3]. Concurrently, novel primitives emerged to address the sampling artifacts inherent to discrete points. A pivotal development was EWA splatting [10], which utilized 3D Gaussian kernels to approximate continuous surfaces. Yet, it was not until two decades later that Kerbl et al. [1] achieved the first fully real-time implementation of this concept. Today, while polygonal rendering can deliver near-photorealistic results, 3D Gaussian Splatting (3DGS) has emerged as a compelling, lightweight alternative specifically optimized for the high-fidelity reconstruction of real-world scenes [11].

In this context, this work aims to review the theoretical foundations of the Gaussian Splatting technique, guided by a central inquiry: to what extent do the specific numerical optimizations employed in the Gaussian Splatting framework determine its real-time performance relative to established rendering pipelines? Consequently, the primary objective of this research is to analyze and evaluate the 3D Gaussian Splatting pipeline as a state-of-the-art technique for Novel View Synthesis, quantifying its capacity to balance photographic fidelity, real-time rendering speed, and training efficiency. To achieve this, we first aim to deconstruct the mathematical foundations of the model, encompassing anisotropic Gaussians, Spherical Harmonics, and optimization via gradient descent. Subsequently, we seek to contextualize the technique within the traditional rasterization pipeline, highlighting the paradigm shift regarding graphic primitives and rendering logic. With this comparative framework established, we proceed to a dedicated exploration of the specific theoretical and algorithmic mechanics of the Gaussian Splatting technique itself. Moving from theory to practice, this study then establishes optimal data capture strategies for proprietary scenes by defining sampling parameters and recording conditions that maximize quality, while quantitatively comparing these reconstructions against standard datasets. Finally, we evaluate the original code implementation by analyzing the impact of specific ablations and convergence stability on the quality of the final reconstruction.

To systematically address these objectives, the remainder of this document is structured to provide a comprehensive analysis moving from theoretical foundations to experimental validation. Chapter 2 establishes the necessary mathematical background, reviewing essential concepts in projective geometry, the fundamentals of the multivariate normal distribution, spherical harmonics, and optimization via stochastic gradient descent. This is followed by Chapter 3, which places the research in context by analyzing traditional rasterization pipelines and contrasting them with alternative approaches such as volumetric rendering and point-based rendering that paved the way for radiance fields. Chapter 4 provides a technical dissection of the 3DGS algorithm itself, detailing the initialization via Structure-from-Motion (SfM) point clouds, defining the rendering primitive and its projection, and explaining the splat rendering pipeline alongside the training and densification strategies.

The validation of these concepts is detailed in Chapter 5, which outlines the methodological framework, including the experimental setup, the curation of both standard and proprietary datasets, and the specific metrics used for quality and performance evaluation. Chapter 6 then presents the results of this investigation, offering a qualitative visual analysis, quantitative reconstruction results, and hardware performance benchmarks. The document concludes with Chapter 7, which discusses these findings, addresses the limitations of the current implementation, and proposes avenues for future research.

2 Mathematical Background

This chapter delineates the mathematical foundations essential to the Gaussian Splatting architecture. The discussion proceeds in four stages: (1) an overview of projective geometry and homogeneous coordinates to formalize the rendering space; (2) a definition of the multivariate normal distribution as the fundamental geometric primitive; (3) an introduction to Spherical Harmonics for high-fidelity color reproduction; and (4) an analysis of Stochastic Gradient Descent as the driving optimization mechanism. Together, these concepts form the theoretical basis for the rendering pipeline described in subsequent sections.

2.1 Projective Geometry

Throughout history, geometry has been formulated in two complementary ways. The synthetic approach defines primitive geometric elements such as points, lines, and planes, and specifies their relationships through axioms that mirror visual intuition. By contrast, analytic geometry is based on algebraic formulations, particularly linear algebra, which makes it more general and better suited for the description of complex structures. For this reason, analytic methods are widely preferred in the field of computer graphics [12].

Perspective, an artistic technique for representing the apparent distortion of objects in visual perception, provided the original motivation for the development of projective geometry. The subject was introduced by the architect Girard Desargues in the 17th century as the mathematical foundation of perspective representation [12].

Formally, a projective space $\mathcal{P}^n$ over the field $\mathbb{R}$ can be defined as the set of all straight lines in $\mathbb{R}^{n+1}$ passing through the origin. In computer graphics, three-dimensional projective space $\mathcal{P}^3$ is of primary importance, as it provides a framework for perspective projection and geometric transformations [12].

2.1.1 Definition of Projective Space

Intuitively, projective space identifies vectors that differ only by a nonzero scalar. In this sense, two nonzero vectors in $\mathbb{R}^4$ represent the same point in projective space if one is a scalar multiple of the other, that is given $\mathbf{u}, \mathbf{v} \in \mathbb{R}^4 \setminus \{\mathbf{0}\}$ they are *homogeneously equivalent* if there exists a scalar $\lambda \in \mathbb{R} \setminus \{0\}$ such that $\mathbf{u} = \lambda \mathbf{v}$ [13].

Theorem 2.1.1. *The homogeneous relation in $\mathbb{R}^4 \setminus \{\mathbf{0}\}$ is an equivalence relation.*

Proof. Let $\mathbf{u}, \mathbf{v}, \mathbf{w} \in \mathbb{R}^4 \setminus \{\mathbf{0}\}$. Then:

- **Reflexivity:** For any $\mathbf{u}$, we have $\mathbf{u} = 1 \cdot \mathbf{u}$, hence $\mathbf{u} \sim \mathbf{u}$.

- **Symmetry:** If $\mathbf{u} \sim \mathbf{v}$, then $\mathbf{u} = \lambda \mathbf{v}$ for some $\lambda \neq 0$. It follows that $\mathbf{v} = \frac{1}{\lambda} \mathbf{u}$, so $\mathbf{v} \sim \mathbf{u}$.
- **Transitivity:** If $\mathbf{u} \sim \mathbf{v}$ and $\mathbf{v} \sim \mathbf{w}$, then $\mathbf{u} = \lambda \mathbf{v}$ and $\mathbf{v} = \alpha \mathbf{w}$ for some $\lambda, \alpha \neq 0$. Thus, $\mathbf{u} = (\lambda \alpha) \mathbf{w}$, which implies $\mathbf{u} \sim \mathbf{w}$.

Therefore, it is an equivalence relation. □

As a result of this theorem, vectors can be grouped into classes that behave like single geometric points. Collecting all such classes yields the definition of projective space.

Definition 2.1.2. The *projective space* $\mathbb{P}^3$ is the set of equivalence classes of $\mathbb{R}^4 \setminus \{\mathbf{0}\}$ under the homogeneous relation. That is, $\mathbb{P}^3 = (\mathbb{R}^4 \setminus \{\mathbf{0}\}) / \sim$. Each equivalence class $[\mathbf{u}]$ corresponds to a unique point in projective space [13].

2.1.2 Homogeneous Representation of Geometrical Elements

Homogeneous coordinates provide a unified framework to represent both finite and infinite elements of geometry. By embedding $\mathbb{R}^3$ into projective space $\mathbb{P}^3$, points, lines, and planes can be described algebraically in a consistent manner, where notions such as parallelism are naturally expressed through intersections at infinity. This representation is fundamental in projective geometry and widely used in computer graphics, as it simplifies the formulation of geometric transformations and perspective projections [13, 12].

Homogeneous Representation of Points

The vector $\mathbf{x} = (x_1, x_2, x_3, x_4)^\top \in \mathbb{P}^3$ with $x_4 \neq 0$ represents a point $\mathbf{x}' = (x'_1, x'_2, x'_3)^\top \in \mathbb{R}^3$ with its Cartesian components obtained by the following normalization

$$x'_1 = x_1/x_4, \quad x'_2 = x_2/x_4, \quad x'_3 = x_3/x_4.$$

As a result of the equivalence relation described earlier, any scalar multiple $\lambda \mathbf{x} \in \mathbb{P}^3$ with $\lambda \neq 0$ describes the same point x' . A common choice of representative is the homogeneous vector with fourth component equal to one [13].

Vectors where $x_4 = 0$ are called ideal points or points at infinity. Note that as their fourth component is 0, the normalization process cannot be carried out, and therefore no corresponding point exists in $\mathbb{R}^3$. Geometrically, these points capture directions in space: for example, two parallel lines in $\mathbb{R}^3$ intersect at a common ideal point in $\mathbb{P}^3$. In the field of computer graphics, they are used to represent vanishing points in perspective projection, thereby providing a precise mathematical model for these phenomena [13].

Homogeneous Representation of Planes

A plane in Euclidean space $\mathbb{R}^3$ is described by a scalar equation of the form

$$\pi_1 x + \pi_2 y + \pi_3 z + \pi_4 = 0,$$

with $(\pi_1, \pi_2, \pi_3)^\top \neq \mathbf{0}$. This equation can be homogenized by the replacements $x = x_1/x_4$, $y = x_2/x_4$, $z = x_3/x_4$. The equation thus becomes

$$\begin{aligned} \pi_1(x_1/x_4) + \pi_2(x_2/x_4) + \pi_3(x_3/x_4) + \pi_4 &= 0, \\ \pi_1 x_1 + \pi_2 x_2 + \pi_3 x_3 + \pi_4 x_4 &= 0, \end{aligned}$$

or equivalently, $\boldsymbol{\pi}^\top \mathbf{x} = 0$ [13].

Definition 2.1.3. A plane in projective space $\mathbb{P}^3$ is represented by a nonzero vector $\boldsymbol{\pi} = (\pi_1, \pi_2, \pi_3, \pi_4)^\top \in \mathbb{R}^4$. The corresponding plane is the set of homogeneous points $\mathbf{x} \in \mathbb{P}^3$ satisfying $\boldsymbol{\pi}^\top \mathbf{x} = 0$ [13].

Note that the representations of points and planes in homogeneous coordinates are *dual*. By this duality, the incidence properties are symmetric: three non-collinear points define a unique plane, and dually, three independent planes define a unique point. Likewise, two distinct planes intersect in a line, just as two distinct points determine a line [13]. The notion of planes in $\mathbb{P}^3$ gives way to the following results.

Theorem 2.1.4. *The set of all ideal points in $\mathbb{P}^3$ forms the plane at infinity π_∞ , which is represented (up to scale) by the homogeneous plane-vector $\boldsymbol{\pi}_\infty = (0, 0, 0, 1)^\top$ [13].*

Proof. An ideal point has the form $\mathbf{x} = (x_1, x_2, x_3, 0)^\top$. Hence

$$\boldsymbol{\pi}_\infty^\top \mathbf{x} = (0, 0, 0, 1)^\top \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ 0 \end{pmatrix} = 0,$$

so all such $\mathbf{x}$ lie in π_∞ . □

Corollary 2.1.5. *Two distinct parallel planes in $\mathbb{R}^3$ intersect in a projective line at infinity contained in π_∞ [13].*

Proof. Let the two parallel planes be represented in homogeneous form by $\mathbf{U} = (a, b, c, d)^\top$ and $\mathbf{V} = (a, b, c, d')^\top$ with $d \neq d'$ so they share the same normal (a, b, c) but have different offsets. Their intersection is the set of points $\mathbf{X} = (x_1, x_2, x_3, x_4)^\top$ satisfying

$$\mathbf{U}^\top \mathbf{X} = ax_1 + bx_2 + cx_3 + dx_4 = 0, \quad \mathbf{V}^\top \mathbf{X} = ax_1 + bx_2 + cx_3 + d'x_4 = 0.$$

Subtracting these two equations yields

$$(d - d')x_4 = 0.$$

Since $d \neq d'$, we must have $x_4 = 0$. Hence, every intersection point is an *ideal point*. With $x_4 = 0$, the remaining condition reduces to

$$ax_1 + bx_2 + cx_3 = 0,$$

which defines a projective line in the plane at infinity $\pi_\infty = \{x_4 = 0\}$. Therefore, the intersection of the two parallel planes is a line at infinity contained in π_∞ . □

Geometrically, this corresponds to the perspective distortion in which parallel planes in $\mathbb{R}^3$ “meet” at infinity along a line determined by their shared normal direction [13].

2.1.3 Projective Transformations

Projective transformations (also known as projectivities, collineations, or homographies) can be defined in several equivalent ways. One common perspective is as functions that preserve the collinearity of points, that is functions of the form $h : \mathbb{P}^3 \rightarrow \mathbb{P}^3$ such that $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3 \in \mathbb{P}^3$ are collinear if and only if $h(\mathbf{x}_1)$, $h(\mathbf{x}_2)$, and $h(\mathbf{x}_3)$ do as well. In this thesis, we adopt the following matrix-based description.

Definition 2.1.6. A planar projective transformation is a linear transformation on homogeneous 4-vectors represented by a nonsingular 4×4 matrix H ,

$$\mathbf{x}' = H\mathbf{x} = \begin{bmatrix} A & \mathbf{t} \\ \mathbf{v}^\top & v \end{bmatrix} \mathbf{x}.$$

Note that multiplying H by a nonzero scalar factor does not change the projective transformation, since $\alpha H\mathbf{x} = \alpha \mathbf{x}' \sim \mathbf{x}'$. As a result, H is named a homogeneous matrix [13].

Theorem 2.1.7. A projective transformation preserves the collinearity of points [13].

Proof. Consider $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3 \in \mathbb{P}^2$ collinear points. By definition, there exists a line $\mathbf{l}$ such that $\mathbf{x}_i^\top \mathbf{l} = 0$ for $i \in \{1, 2, 3\}$. Then, we have that

$$\begin{aligned} \mathbf{x}_i^\top H^\top H^{-\top} \mathbf{l} &= (H\mathbf{x}_i)^\top (H^{-\top} \mathbf{l}) \\ &= (H\mathbf{x}_i)^\top \mathbf{l}'. \end{aligned}$$

Thus all transformed points $\mathbf{x}'_i = H\mathbf{x}_i$ are on the line $\mathbf{l}' = (H^{-\top} \mathbf{l})$. □

Note that since H is nonsingular, its inverse H^{-1} also defines a projective transformation. Therefore, collinearity is preserved not only under the forward transformation but also under its inverse.

The set of projective linear transformations, denoted $PL(n)$, is constructed by identifying nonsingular matrices in $GL(n, \mathbb{R})$ that differ by a nonzero scalar multiple. In other words, every $H \in GL(n, \mathbb{R})$ induces a projective transformation $\mathbf{x}' = H\mathbf{x}$ on homogeneous coordinates, and two such matrices represent the same transformation exactly when they differ by a scalar factor. Thus $PL(n)$ can be realized as the quotient of $GL(n, \mathbb{R})$ by the subgroup of scalar matrices, and in particular it forms a group [13].

Theorem 2.1.8. Let $GL(n) = \{A \in M_n(\mathbb{R}) : \det A \neq 0\}$ and let $Z = \{\lambda I_n : \lambda \in \mathbb{R}^\times\} \subset GL(n, \mathbb{R})$. Then Z is a normal subgroup of $GL(n)$, the quotient $PL(n) := GL(n, \mathbb{R})/Z$ is a group [13].

Proof. Note that Z is a subgroup of $GL(n, \mathbb{R})$, because $(\lambda I_n)(\mu I_n) = (\lambda\mu)I_n$ and $(\lambda I_n)^{-1} = (\lambda^{-1})I_n$. As every scalar matrix commutes with every $A \in GL(n)$, $Z \subseteq Z(GL(n))$, the center of $GL(n)$. Therefore Z is normal and the quotient $GL(n)/Z$ is a group. □

Projective transformations admit several important subclasses, each distinguished by the geometric invariants they preserve. These subclasses are particularly relevant in computer graphics, where different applications require the preservation of specific properties. Isometries preserve Euclidean distances and angles, similarities extend this to include uniform scaling, and affinities preserve parallelism but not necessarily lengths or angles [13]. In the following sections, these families are presented in detail.

Elementary Transformations

Linear transformations preserve both vector addition and scalar multiplication. Although translations are not linear, they are commonly considered together with linear transformations as the basic transformations, serving as fundamental building blocks for the more general classes of projective transformations [3]. The following descriptions are adapted from [3].

Rotations. In three dimensions, rotations are most commonly described about the coordinate axes. In homogeneous coordinates, the corresponding matrices are:

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \theta & -\sin \theta & 0 \\ 0 & \sin \theta & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad R_y(\theta) = \begin{bmatrix} \cos \theta & 0 & \sin \theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \theta & 0 & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad R_z(\theta) = \begin{bmatrix} \cos \theta & -\sin \theta & 0 & 0 \\ \sin \theta & \cos \theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Any rotation about an arbitrary axis can be obtained by composing these elementary rotations.

Translations. A translation by vector $t = (t_x, t_y, t_z)^\top$ is expressed in homogeneous coordinates as:

$$T(t) = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Scalings. A scaling by factors $\lambda_x, \lambda_y, \lambda_z \in \mathbb{R}$ along each axis is given by:

$$S(\lambda_x, \lambda_y, \lambda_z) = \begin{bmatrix} \lambda_x & 0 & 0 & 0 \\ 0 & \lambda_y & 0 & 0 \\ 0 & 0 & \lambda_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Reflections. Reflections across the coordinate planes are obtained by negating the corresponding coordinate. For instance:

$$R_{xy} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad R_{xz} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad R_{yz} = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Isometric Transformations

The main characteristic of isometric transformations (also known as isometries) is the preservation of Euclidean distance. These are transformations of the form,

$$\mathbf{x}' = M H_E \mathbf{x} = M \begin{bmatrix} R & \mathbf{t} \\ \mathbf{0}^\top & 1 \end{bmatrix} \mathbf{x},$$

where R is a rotation matrix, $\mathbf{t}$ is a translation vector, and M is either the identity matrix I or a reflection matrix. When $M = I$, the transformation is an Euclidean transformation, that is an isometry that preserves orientation as it is a composition of a translation and a rotation. When M is a reflection matrix, the transformation reverses orientation, which can be interpreted as a reflection composed with an Euclidean transformation. Properties such as length, angle, and area are invariant under these types of transformations [13].

Similarity Transformations

Similarity transformations or similarities are isometries composed with an isotopic scaling. These have the form,

$$\mathbf{x}' = H_S \mathbf{x} = \begin{bmatrix} S & \mathbf{0} \\ \mathbf{0}^\top & 1 \end{bmatrix} M \begin{bmatrix} R & \mathbf{t} \\ \mathbf{0}^\top & 1 \end{bmatrix} \mathbf{x},$$

where S represents an isotropic scaling matrix. Angles and the ratio between lengths and areas are invariant under similarities [13].

Affine Transformations

Affine transformations, also referred to as affinities, are mappings that combine of a nonsingular transformation and a translation. These have the form

$$\mathbf{x}' = H_A \mathbf{x} = \begin{bmatrix} A & \mathbf{t} \\ \mathbf{0}^\top & 1 \end{bmatrix} \mathbf{x},$$

where A is a nonsingular matrix [13]. A geometric interpretation of A can be derived from the following theorem.

Theorem 2.1.9. *The affine matrix $A_{n \times n}$ can be decomposed as $A = R(\theta)R(-\phi)DR(\phi)$, where $R(\theta)$ and $R(\phi)$ correspond to rotations by their respective angles and D is a diagonal matrix [13].*

Proof. The Singular Value Decomposition theorem states that as A is a real matrix, there exist orthogonal matrices $U, V \in \mathbb{R}^{n \times n}$ and a diagonal matrix $D \in \mathbb{R}^{n \times n}$ such that

$$\begin{aligned} A &= UDV^T \\ &= U(V^T V)DV^T \\ &= (UV^T)(VDV^T) \\ &= R(\theta)R(-\phi)DR(\phi) \end{aligned}$$

for some θ, ϕ as the multiplication of orthogonal matrices is orthogonal. $\square$

Theorem 2.1.9 shows that affine transformation matrix can be interpreted as a composition of elementary linear operations. Specifically, it corresponds to a rotation of the coordinate axes by an angle ϕ , followed by a (potentially anisotropic) scaling along the rotated axes, and a subsequent rotation back by $-\phi$ to the original orientation. This sequence may be further followed by an additional rotation by an angle θ . The most important invariants under affine transformations are parallel lines, the ratio of lengths of parallel line segments and the ratio of areas [13].

The fundamental distinction between affine and projective transformations lies in the presence of the additional vector $\mathbf{v}$ in the projective case. This term introduces nonlinear behavior absent from affinities [13, 12]. An important nonlinear characteristic of projective transformations can be seen in the following theorem.

Theorem 2.1.10. *Projective transformations may map ideal points to finite points, while affine transformations cannot [13].*

Proof. Consider the vector $(\mathbf{x}^T, 0)^T$ representing an ideal point, with $\mathbf{x}^T$ a vector of size $(n-1) \times (n-1)$. This vector is transformed by an affinity as follows,

$$\begin{bmatrix} A & \mathbf{t} \\ \mathbf{0}^\top & v \end{bmatrix} \begin{pmatrix} \mathbf{x} \\ 0 \end{pmatrix} = \begin{pmatrix} A\mathbf{x} \\ 0 \end{pmatrix}.$$

In contrast, transformation by a projectivity is as follows,

$$\begin{bmatrix} A & \mathbf{t} \\ \mathbf{v}^T & v \end{bmatrix} \begin{pmatrix} \mathbf{x} \\ 0 \end{pmatrix} = \begin{pmatrix} A\mathbf{x} \\ \mathbf{v}^T \mathbf{x} \end{pmatrix}.$$

□

Another important difference concerns the homogeneity of their action. Affine transformations act uniformly across the plane: the scalings and rotations they induce depend only on the initial size and orientation of objects, not on their position. In contrast, projective transformations may act inhomogeneously. For instance, perspective mappings exhibit position-dependent scalings—distant objects appear smaller—and the orientation of transformed lines may vary with their location relative to the line at infinity [13].

2.2 Fundamentals of the Multivariate Normal Distribution

The primitives employed in the Gaussian Splatting process, three-dimensional Gaussians, are theoretically grounded in the properties of the multivariate normal distribution. Mardia et al. [2] describe this distribution as a natural extension of the univariate normal function to higher dimensions. The geometric interpretation and intrinsic characteristics of this distribution are fundamental to the reconstruction of scenes from Gaussians.

2.2.1 Mahalanobis Distance

The multinormal distribution depends on the Mahalanobis distance rather than the usual Euclidean distance in $\mathbb{R}^n$. This distance is described by [2] as

$$\Delta(x, y) = \sqrt{(x - y)^T \Sigma^{-1} (x - y)},$$

where Σ is a symmetric positive-definite matrix, usually corresponding to the covariance matrix. This distance is invariant under affine transformations. Given two random vectors x, y such that $E(x) = \mu_1$ and $E(y) = \mu_2$ with covariance matrix Σ , an affine transformation is of the form

$$\begin{aligned} x' &= Ax + b, \\ y' &= Ay + b, \\ \Sigma' &= A\Sigma A^T, \end{aligned} \tag{2-1}$$

with A nonsingular. Moreover, if x is a random vector with expected value μ and covariance matrix Σ , then $\Delta(x, \mu)$ is a random variable [2].

2.2.2 Multivariate Normal Distributions

According to [14], a random vector has a p -dimensional multivariate normal distribution, $N_p(\mu, \Sigma)$, if it has the following probability density function:

$$\begin{aligned} f(x) &= |2\pi\Sigma|^{-1/2} \cdot \exp\left(-\frac{1}{2}\Delta(x, \mu)^2\right) \\ &= |2\pi\Sigma|^{-1/2} \cdot \exp\left(-\frac{1}{2}(x - \mu)^T \Sigma^{-1} (x - \mu)\right). \end{aligned}$$

where the expected value and covariance matrix are $E(x) = \mu$ and $\text{Var}(x) = \Sigma$.

It is emphasized in [14] that the probability density function of the multivariate normal distribution is normalized by the factor $|2\pi\Sigma|^{-1/2}$. However, in the Gaussian Splatting process, this function is often used in an unnormalized form [11].

2.2.3 Geometric Interpretation

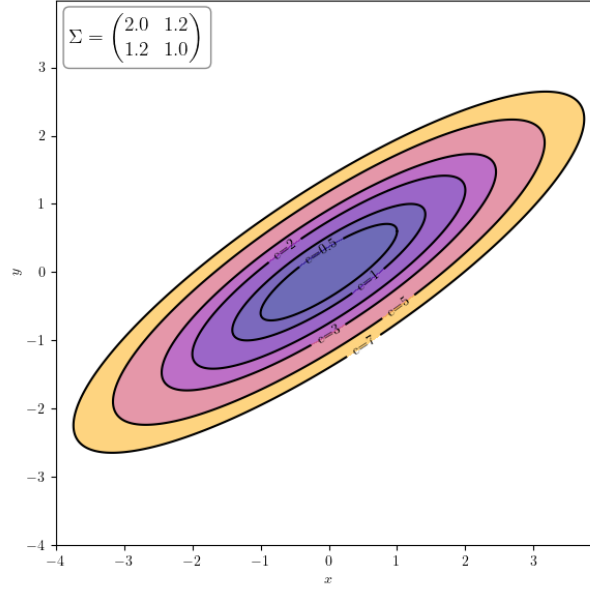


Figure 2-1: Isocontours of an example Multinormal Function. Based on [2].

Multivariate normal functions are interpreted geometrically through their contours. According to [2], these contours correspond to points where the distribution has a constant value. Specifically, they are the points for which there exists a fixed $c \in \mathbb{R}$ such that

$$\Delta(x, \mu)^2 = (x - \mu)^\top \Sigma^{-1} (x - \mu) = c^2.$$

Since Σ is a symmetric matrix, it can be decomposed as $\Sigma = \Gamma \Lambda \Gamma^\top$, where Λ is a diagonal matrix whose entries are the eigenvalues of Σ and Γ is an orthogonal matrix whose columns are the eigenvectors of Σ [2]. Consider the linear transformation $y = \Gamma^\top (x - \mu)$, then we have:

$$\begin{aligned} c^2 &= (x - \mu)^\top \Sigma^{-1} (x - \mu) \\ &= (x - \mu)^\top (\Gamma \Lambda \Gamma^\top)^{-1} (x - \mu) \\ &= (x - \mu)^\top (\Gamma^\top)^{-1} \Lambda^{-1} \Gamma^{-1} (x - \mu) \\ &= (x - \mu)^\top (\Gamma^{-1})^{-1} \Lambda^{-1} \Gamma^\top (x - \mu) \\ &= (x - \mu)^\top \Gamma \Lambda^{-1} \Gamma^\top (x - \mu) \\ &= y^\top \Lambda^{-1} y. \end{aligned}$$

Since Γ is orthogonal, note that Λ^{-1} is also a diagonal matrix, with diagonal entries given by $\lambda_{ii}^{-1} = \frac{1}{\lambda_{ii}}$, where λ_{ii} denotes the diagonal entries of Λ . Therefore, we have:

$$c^2 = \sum_{i=1}^p \frac{y_i^2}{\lambda_i} = \sum_{i=1}^p \frac{(\gamma_i^\top (x_i - \mu_i))^2}{\lambda_i}$$

where γ_i are the columns of Γ . It is clear that this equation corresponds to an ellipsoid whose axes are rotated according to $\gamma_i^\top$ [2].

2.3 Spherical Harmonics

Spherical harmonics are a basis over the sphere S in $\mathbb{R}^3$ that is analogous to the Fourier transform over the unit circle and can be used to model lightning in 3D rendering [15, 5, 4]. Using the standard parameterization in spherical coordinates,

$$\mathbf{p} = (x, y, z)^\top = (\sin \theta \cos \phi, \sin \theta \sin \phi, \cos \theta)^\top.$$

The basis functions are defined as

$$Y_l^m(\theta, \phi) = K_l^m e^{im\phi} P_l^{|m|}(\cos \theta),$$

where $l \in \mathbb{N}$, $-l \leq m \leq l$, $P_l^{|m|}$ are the associated Legendre polynomials, and K_l^m are normalization constants of the form

$$K_l^m = \sqrt{\frac{(2l+1)(l-|m|)!}{4\pi(l+|m|)!}}.$$

This definition forms an orthonormal basis in $\mathbb{C}$, that is given two SH functions Y_l^m and $Z_{l'}^{m'}$,

$$\int_S Y_l^m Z_{l'}^{m'} ds = \begin{cases} 0, & Y_l^m \neq Z_{l'}^{m'} \\ 1, & Y_l^m = Z_{l'}^{m'} \end{cases}.$$

However, practical applications of SH in computer graphics often use a real-valued basis obtained through the transformation

$$y_l^m = \begin{cases} \sqrt{2} \Re(Y_l^m), & m > 0, \\ \sqrt{2} \Im(Y_l^m), & m < 0, \\ Y_l^0, & m = 0, \end{cases}$$

which can be expressed explicitly as

$$y_l^m = \begin{cases} \sqrt{2} K_l^m \cos(m\phi) P_l^m(\cos \theta), & m > 0, \\ \sqrt{2} K_l^m \sin(-m\phi) P_l^{-m}(\cos \theta), & m < 0, \\ K_l^0 P_l^0(\cos \theta), & m = 0. \end{cases}$$

The parameter l is called the *band index* and represents low-frequency basis functions over the sphere. The functions within a specific band l can be reduced to polynomials of order l in $\mathbb{R}^3$ [15, 5, 4].

2.3.1 Recursive Evaluation of SH Basis Functions

In practice, evaluating the SH basis functions requires efficiently computing the associated Legendre polynomials $P_l^m(z)$ that define their polynomial structure. Direct computation from their analytical definition involves derivatives of complex expressions, which are numerically unstable and computationally expensive. Instead, recurrence relations are used to generate higher-order terms from lower ones, ensuring stability and efficiency in the evaluation of the SH basis [5].

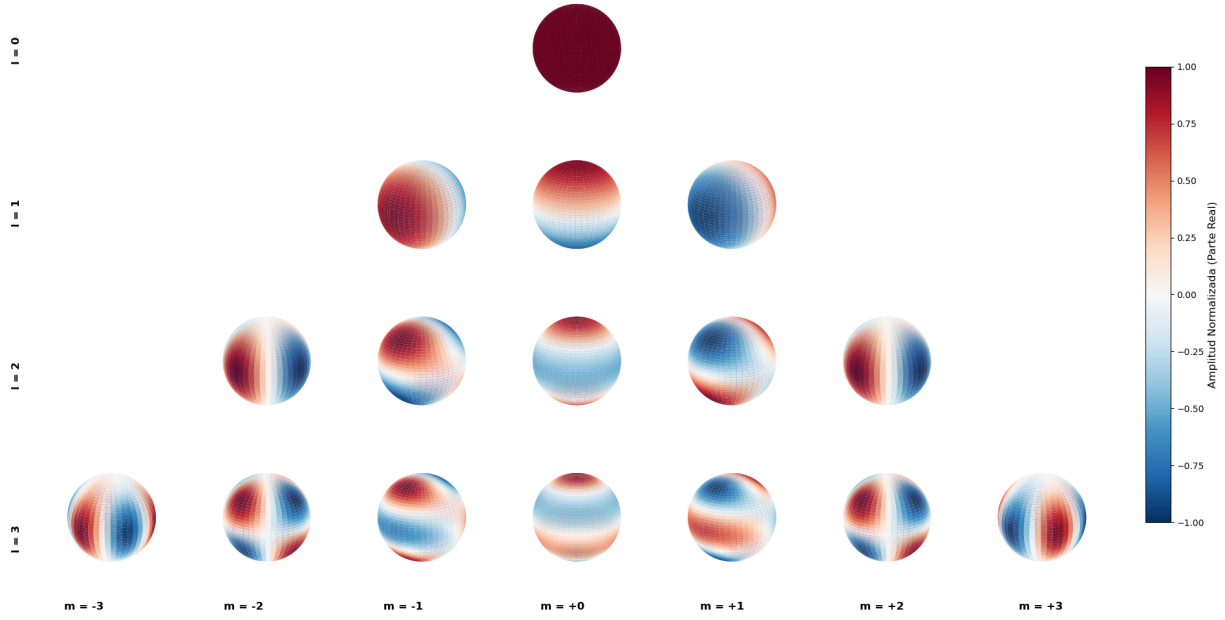


Figure 2-2: Representation of the first 4 SH bands on a unit sphere. Based on [3, 4, 5]

Starting from $P_0^0 = 1$, the recurrence relations for the associated Legendre polynomials are given by

$$\begin{aligned} P_m^m &= (1 - 2m) P_{m-1}^{m-1} z, \\ P_{m+1}^m &= (2m + 1) z P_m^m, \\ P_l^m &= \frac{(2l - 1) z P_{l-1}^m - (l + m - 1) P_{l-2}^m}{l - m}. \end{aligned}$$

Given a point (x, y, z) on the unit sphere, the φ -dependent trigonometric terms of the SH functions (multiplied by $\sin^m \theta$) can also be computed recursively using

$$\begin{aligned} S(m) &= x S(m - 1) + y C(m - 1), \\ C(m) &= x C(m - 1) - y S(m - 1), \end{aligned}$$

with initialization $S(0) = 0$ and $C(0) = 1$. Replacing $\sin |m|\varphi$ and $\cos m\varphi$ by $S(m)$ and $C(m)$ respectively allows for more efficient evaluation of the SH basis functions, as it reuses common subexpressions and minimizes redundant trigonometric computations [5].

2.3.2 Projection and Reconstruction

As the SH basis is orthonormal, any scalar function f over the unit sphere $S \subset \mathbb{R}^3$ can be projected into SH coefficients by the integral

$$f_l^m = \int_S f(s) y_l^m(s) ds,$$

where y_l^m denotes the SH basis functions [15, 5].

A projection of order n produces n^2 coefficients. When a high-frequency function is reconstructed from a low-order projection, the result is a smoothed approximation that avoids aliasing [15]. The function f can

be approximated from its SH coefficients as

$$\tilde{f}(s) = \sum_{l=0}^{n-1} \sum_{m=-l}^l f_l^m y_l^m(s),$$

where $\tilde{f}$ denotes an approximation of f whose accuracy increases with the number of bands. This reconstruction can also be expressed in compact vector form as

$$\tilde{f}(s) = \sum_{i=1}^{n^2} f_i y_i(s),$$

where the index mapping is given by $i = l(l+1) + m + 1$ [15, 5].

2.3.3 Properties of SH Projections

Rotational invariance. The SH projection is invariant under rotations. Given a rotation Q on S and defining $g(s) = f(Q(s))$, it follows that

$$\tilde{g}(s) = \tilde{f}(Q(s)).$$

This means that SH projection does not introduce aliasing when the set of sample points is rotated [15, 4].

Orthogonality of projections. Due to the orthogonality of the SH basis, the projections of any two functions a and b satisfy

$$\int_S \tilde{a}(s) \tilde{b}(s) ds = \sum_{i=1}^{n^2} a_i b_i.$$

This property directly follows from the orthonormality of the SH basis functions [15, 4].

Convolution property. Let h be a circularly symmetric kernel function around the z -axis, $h(z)$, and let $h * f$ denote its convolution with f . The SH coefficients of the convolution satisfy

$$(h * f)_l^m = \sqrt{\frac{4\pi}{2l+1}} h_l^0 f_l^m = \alpha_l^0 h_l^0 f_l^m,$$

where, since h is circularly symmetric, only coefficients with $m = 0$ are nonzero [15].

2.4 Gradient Descent

Gradient descent is an iterative optimization method that generates a sequence $\{x^{(k)}\}_{k=0}^{\infty}$ according to

$$x^{(k+1)} = x^{(k)} + t^{(k)} \Delta x^{(k)},$$

where $t^{(k)} \geq 0$ is the step size and $\Delta x^{(k)}$ is the search direction at iteration k . According to [16], a method is said to be a descent method if it generates a sequence $\{x^{(k)}\}$ such that $f(x^{(k+1)}) < f(x^{(k)})$. From standard results in convex analysis, if

$$\nabla f(x^{(k)})^\top (x^{(k+1)} - x^{(k)}) \geq 0,$$

then $f(x^{(k+1)}) \geq f(x^{(k)})$ for a continuously differentiable function f . By the contrapositive, if the descent condition holds, then

$$\nabla f(x^{(k)})^\top \Delta x^{(k)} < 0. \quad (2-2)$$

Condition (2-2) is therefore necessary for $\Delta x^{(k)}$ to be a valid descent direction. In the GD method, the search direction is given by the negative gradient, which satisfies (2-2) as

$$\nabla f(x^{(k)})^\top (-\nabla f(x^{(k)})) = -\|\nabla f(x^{(k)})\|_2^2 < 0,$$

whenever $\nabla f(x^{(k)}) \neq 0$. Hence, GD is guaranteed to be a descent method [16].

The efficiency of GD depends on the choice of step size. A practical strategy described by [16] is line search, which determines how far along the search direction to move. In exact line search, the step size is chosen to minimize f along the ray $\{x^{(k)} - t\nabla f(x^{(k)}) \mid t \in \mathbb{R}^+\}$. Since this one-dimensional minimization can be expensive, backtracking line search is often preferred. This method introduces parameters $\alpha \in (0, 0.5)$ and $\beta \in (0, 1)$, and starts with $t = 1$, repeatedly reducing t by the factor β until the condition

$$f(x^{(k)} - t\nabla f(x^{(k)})) \leq f(x^{(k)}) - \alpha t \|\nabla f(x^{(k)})\|_2^2$$

is satisfied. This criterion is guaranteed to hold. Indeed, for sufficiently small $t > 0$, the first-order Taylor expansion yields

$$f(x^{(k)} - t\nabla f(x^{(k)})) \approx f(x^{(k)}) - t \|\nabla f(x^{(k)})\|_2^2,$$

and since $\|\nabla f(x^{(k)})\|_2^2 > 0$ unless at a stationary point, the decrease is strictly larger in magnitude than $\alpha t \|\nabla f(x^{(k)})\|_2^2$ for $\alpha \in (0, 0.5)$. Thus, backtracking always finds a valid step size that ensures sufficient decrease [16].

A common stopping rule for this method is $\|\nabla f(x^{(k)})\|_2 \leq \eta$, where $\eta > 0$ is a small tolerance [16] or after a number of predefined epochs [17]. Both exact and backtracking line search guarantee the convergence of gradient descent when f is strongly convex and has Lipschitz continuous gradients. Under these conditions, the iterates converge linearly to the unique global minimizer [16].

2.4.1 Stochastic Gradient Descent

Variants of Gradient Descent (GD) differ primarily in how they handle the amount of data used to compute the gradient at each iteration. The standard approach, known as Batch Gradient Descent (BGD), computes the gradient of the loss function with respect to the entire dataset. While this ensures a stable and consistent descent direction, it can become computationally expensive for large datasets that do not fit into memory. As a result, alternative variants that use smaller subsets of data per update are often preferred in practice [17].

One of the most widely used alternatives is Stochastic Gradient Descent (SGD). This method avoids redundant computations that arise in BGD when data samples are sufficiently similar, since in such cases successive gradients contribute little new information. SGD updates the model parameters after processing each individual training example and its corresponding label (ground truth). This approach enables faster updates and supports online learning. However, it can exhibit oscillations around the minimum unless the learning rate is decreased gradually. To improve convergence behavior, the examples are typically selected in a random order or in randomly selected batches from the dataset during training [17]. The pseudocode for this method is shown in Algorithm 1.

Algorithm 1 Stochastic Gradient Descent (SGD)

Require: Initial parameters $z^{(0)}$, learning rate t , number of epochs N

```
1: for  $k = 1$  to  $N$  do
2:   Randomly shuffle the training data
3:   for each example  $(x_i, y_i)$  in the dataset do
4:      $g := \nabla f_i(z^{(k)}, x_i, y_i)$  ▷ Compute the gradient for each example
5:      $z^{(k)} := z^{(k-1)} - tg$  ▷ Update parameters
6:   end for
7: end for
```

3 Foundations of 3D Rendering

Rendering is described by [3] as the process through which an image is generated on the screen. Although there are different implementations to carry out the rendering process, most of them are based on the abstraction known as the rendering pipeline. This term refers to a process that involves several sequential stages executed in parallel. Each stage depends on the output of the immediately preceding one, although parallel computations may still be executed at any given moment [18]. Foley et al. [18] describe that, in its early implementations, the pipeline was implemented through fixed functions, meaning the stages were configurable but not fully programmable. However, the current paradigm relies on the generalization of these stages through the use of shaders, that is, “small programs that specify how the duties of certain portions of the pipeline are to be carried out” [18, p. 928]. In general, it is argued that since the implementations of the pipeline are in constant evolution, it is more relevant in the long term to understand the overall purpose of the process rather than its technical details [18].

3.1 Rasterization-based Rendering

One of the most common types of pipeline for real-time rendering is the rasterization-based pipeline, which consists of four main stages [3] as described in Figure **3-1**. The first stage involves the execution of an application that provides most of the data to be used in the subsequent stages, along with other programmable features. The second stage focuses on the processing of per-vertex properties, mainly the necessary geometric transformations for the rendering process. The third stage performs the rasterization process, described by [18] as the approximation of a continuous representation by a discrete collection of pixels. The fourth and final stage consists of fragment or pixel processing, where shading (the “operation of determining the effect of a light on a material” [3, p. 16]) is typically applied, and the fragments that will be visible in the final image are determined.

3.1.1 Application Stage

The first stage consists of the design and execution of specialized software that provides the next stage with rendering primitives, camera parameters, lighting information, textures, and other data [3, 18]. This software is executed on CPUs or on GPUs (through compute shaders), which makes it a highly programmable stage. Moreover, it is executed in parallel despite not being divided into sub-stages [3].

Rendering primitives are defined as “points, lines, and triangles, that might eventually end up on the screen (or whatever output device is being used)” [3, p. 14]. The most widely used type of primitive today is the polygonal mesh, formed multiple polygons through their edges to generate surfaces. These meshes are characterized by their geometry and topology, that is, by the positions of their vertices and the edges that

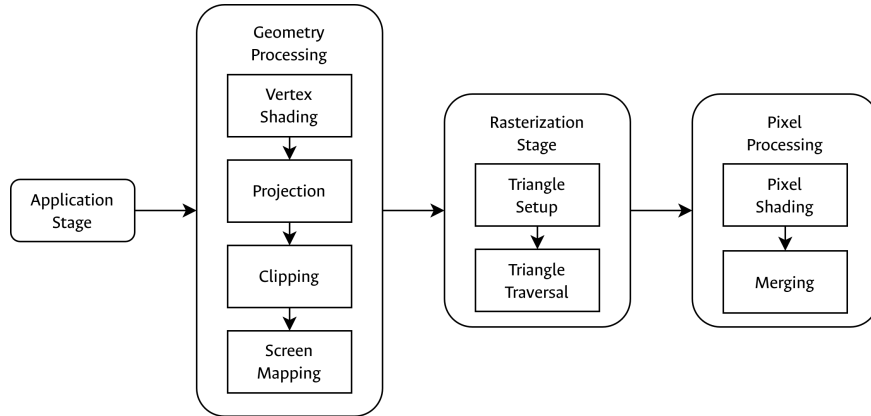


Figure 3-1: Diagram of the Rasterization Rendering Pipeline. Based on [3].

connect them. Such characteristics are commonly represented by two tables: one of vertices and one of faces. The first stores the index, position, and any other attributes of a vertex, whereas the second stores the vertices that compose each face along with optional information about their edges [18].

Triangles are the most commonly used polygons in mesh construction, as they guarantee convexity, their vertices are coplanar, and they allow interpolation through barycentric coordinates. Furthermore, triangular meshes enable efficient operations such as surface subdivision, i.e replacing triangles with smaller ones, and mesh simplification, where a representation is substituted with a geometrically or topologically simpler version. These processes make it possible to adjust an object’s level of detail according to its distance from the camera. However, some elements are not well suited to mesh-based representation, either because of excessive geometric detail or because they possess structures that are difficult to model, such as hair or grass [18].

According to Akenine-Möller et al. [3], this stage also serves additional functions. Among the most notable are the handling of input devices and the actions derived from the signals they transmit. In addition, it carries out computations related to collision detection, which involves modeling both the collision itself and the corresponding response of the objects involved. Finally, this stage implements various optimization algorithms, such as those that perform primitive culling, i.e removing elements that are not visible or necessary for rendering the final image.

3.1.2 Geometry Processing

This is the first stage that is executed entirely on the GPU [3], and it is responsible for most per-vertex and per-triangle operations. It is divided into four sub-stages in which the position and projection of vertices are determined, additional vertices and triangles are generated, and the primitives visible to the camera are identified. The output of this stage is an array containing data associated with each vertex, such as position, color, texture coordinates, UV coordinates, among others.

Vertex Shading

According to Akenine-Möller et al. [3], this process is referred to as vertex shading, since in its early implementations most of the data related to the appearance of models was determined at this stage. This was implemented by “applying lights to each vertex’s location and normal and storing only the resulting

color at the vertex” [3, p.15], with these colors later interpolated. However, in modern GPU architectures, this stage has been extended to the initialization of any data associated with vertices.

The primary task of per-vertex operations is to determine the final positions of vertices. These positions are established through a series of geometric transformations from one coordinate system, or space, to another [3]. Projection space, as described in Definition 2.1.2, is used to represent both coordinates and transformations, which correspond to series of projectivities applied in succession. This representation allows for the concise representation of affine transformations instead of the representation present in Equation 2-1, and the modeling of perspective points as per Theorem 2.1.10.

Initially, each individual model resides in its own space and has not yet been integrated with the rest of the scene [3], that is, “a model [consisting of] a collection of objects and light sources” [18, p. 21]. The coordinates of each model are transformed into a single reference system called world space. This is accomplished through the model transformation, where models are positioned, rotated, and scaled with respect to a common frame of reference. Since only the objects visible to the camera should appear in the final image, the camera serves as the reference frame for the subsequent stages. To simplify certain calculations, the view transformation is applied, in which the entire coordinate system is shifted so that the camera’s position coincides with the origin and its viewing direction is aligned with the z-axis (its precise orientation depends on the implementation of each API). The transformed coordinates of each vertex constitute the first output of this sub-process [3]. It is important to note that the majority of the transformations at this point of the pipeline are affine, thus they can be decomposed into a series of elementary transformations as per Theorem 2.1.9. The transformations performed during the vertex shader stage are represented in Figure 3-2.

The second type of output values corresponds to additional per-vertex data, often related to the modeling of object appearance. Among these data are color, normal vectors, texture coordinates, UV coordinates, among others. These values may be used in computations carried out during this stage or simply passed on to the next one, depending on the criteria defined by the programmer [3].

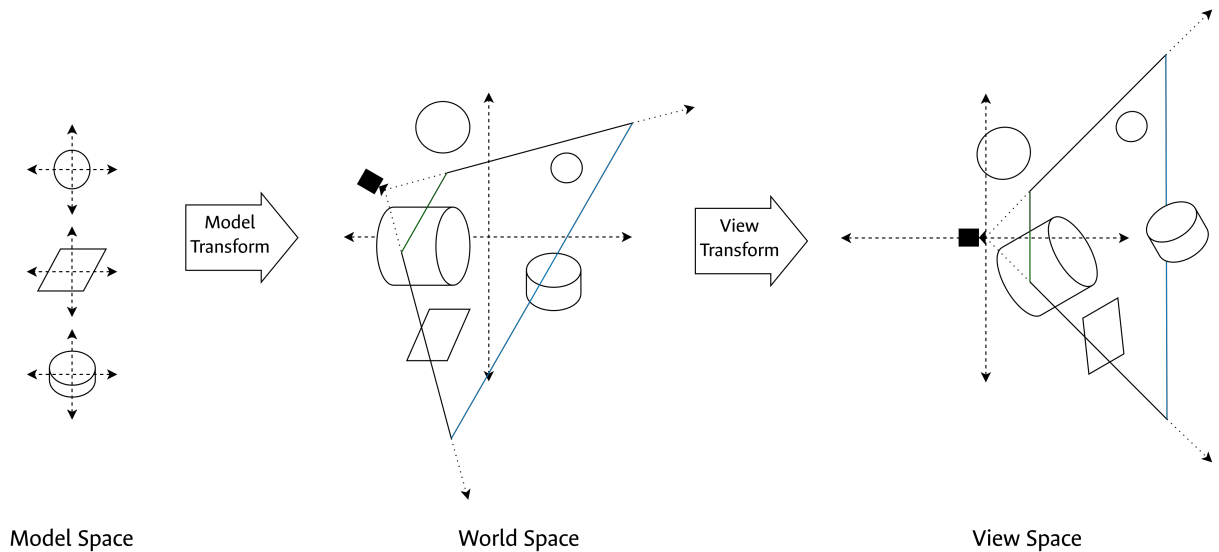


Figure 3-2: 2D Representation of the ModelView Transform. Based on [3].

Projection

The projection stage is described by [3] as the transformation that maps objects from the view volume to the canonical view volume, facilitating the rasterization process. The view volume is an implicit three-dimensional

shape, meaning it is not physically stored, and represents the camera’s field of view. Only objects within this volume are considered visible and continue through the rendering process. This volume is transformed into the canonical view volume, a unit cube with corners at the coordinates $(-1, -1, -1)$ and $(1, 1, 1)$ (which may vary depending on the API used) [3]. The same transformation applied to the corners of the original view volume is also applied to all vertices of the objects it contains, effectively deforming space to achieve the desired projection.

Akenine-Möller et al. [3] point out that this process does not exactly match its mathematical counterpart. It is called projection because, once performed, “the z coordinate is not stored in the generated image but in a z -buffer [...]. In this way, models are projected from three to two dimensions” [3, p. 18]. After this transformation, the coordinates are considered to lie in clip space, ready for the next stage.

The most commonly used types of projection are orthographic and perspective. In orthographic projection, the view volume takes the shape of a rectangular prism, with the key property that parallel lines remain parallel after the transformation. Perspective projection simulates human vision, such that the perceived size of an object decreases inversely with its distance from the viewpoint. Accordingly, the view volume in this case is a truncated rectangular pyramid, known as a frustum [3]. Figure 3-3 illustrates these transformations on view volumes.

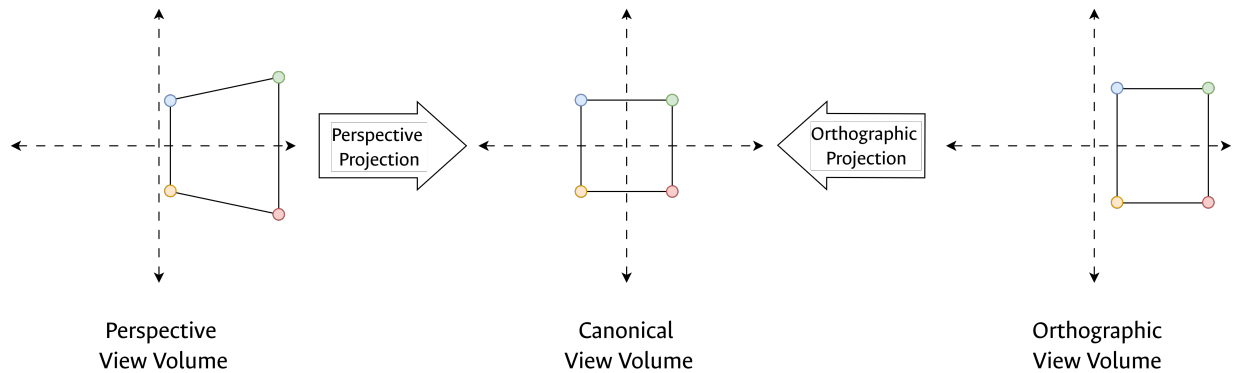


Figure 3-3: 2D Representation of View Volume Transforms of Perspective Transformations.

Optional Vertex Processing

This stage is described by [3] as an optional and configurable module, with its components determined by the computer’s instructions and the capabilities of the hardware. It begins with the tessellation stage, in which sets of vertices are expanded based on their proximity to the camera. The new vertices are used to construct additional sets of triangles, selectively increasing the level of detail. Next, the geometry shader can generate additional vertices on a smaller scale than in tessellation, producing simpler primitives (small sets of vertices) for effects such as particle generation. Finally, the stream output stage enables the storage of vertices in a buffer, allowing them to be reused in subsequent computations either on the CPU or the GPU [3].

Clipping and Screen Mapping

The clipping stage manages vertices located outside the canonical view volume. Primitives entirely outside this volume are discarded, while those fully inside are passed to the next stage. Primitives intersecting the boundaries are clipped by generating new vertices at their intersections with the clipping planes, which include the faces of the canonical view volume and any additional programmer-defined planes. Using these new vertices, fully visible primitives are then reconstructed according to the specified criteria [3].

Finally, a last transformation is applied to the scene coordinates, known as screen mapping. This operation consists of a translation followed by a scaling operation, ensuring that the limits of the canonical view volume coincide with the boundaries of the window where the image will be displayed. In addition, the z coordinate is normalized to the range $[0, 1]$. The specifics of this transformation are implementation-dependent, due to variations in window handling and related characteristics [3].

3.1.3 Rasterization

Rasterization is described by [18] as the process of converting the continuous representation of rendering primitives, such as triangles, points, or Gaussians, into discrete image elements known as fragments. Each fragment corresponds to a complete or partial pixel contained within the primitive's projection. This discretization reflects the physical design of display devices, in which a finite set of rectangular pixels is illuminated to form an image.

The rasterization stage begins with the setup of primitives. In this step, quantities such as edge equations, differentials, and interpolation coefficients are computed to prepare for fragment generation. These operations are typically performed by dedicated hardware units, which provide efficiency but limit the programmer's direct control over the process. Once setup is complete, primitives are traversed, and fragments are produced at the pixel locations where their projections overlap the image grid as shown in Figure 3-4. Each fragment represents a potential contribution to the final image and stores the information required for subsequent shading and visibility tests. To determine fragment coverage, sampling points are evaluated within each pixel to test whether they lie inside the primitive's boundaries. Attributes for the covered samples are then computed through interpolation or related reconstruction techniques [3].

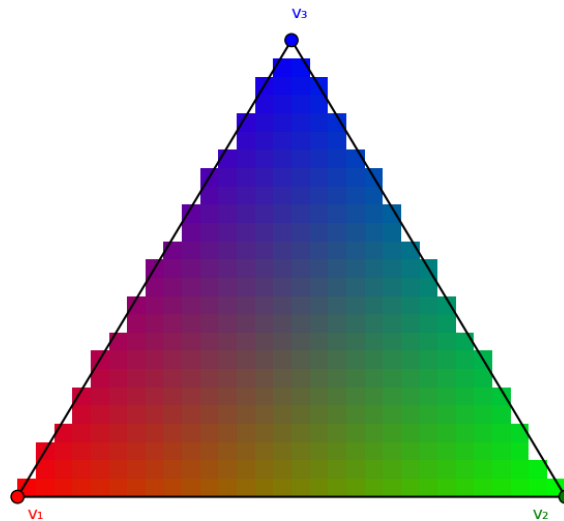


Figure 3-4: Fragments generated from a triangle. Based on [3].

Different sampling strategies can be employed. The simplest approach samples the pixel center, while higher-quality methods take multiple samples per pixel (multisampling) or render at higher resolutions followed by resampling (supersampling). Each fragment carries a depth value, derived from the primitive's z coordinate, as well as interpolated shading attributes from the geometry stage. For triangular meshes, these attributes are typically obtained using barycentric interpolation of the vertex data [3].

After rasterization, the resulting fragments are passed to the fragment shading stage, where material properties, lighting, and texture information are evaluated to determine the final pixel color [3].

3.1.4 Pixel Processing

The final stage of the graphics pipeline focuses on per-fragment computations, including color assignment, blending, and visibility operations. Its objective is to produce the values of the color buffer, a rectangular array of RGB (or another color representation) values, which is subsequently used to render colors in the pixel raster of the output screen. This process consists of two components [3].

The first component is fragment shading (also referred to as pixel shading), which determines the final color of each fragment. This color is computed from interpolated attributes such as per-vertex color, texture coordinates, and UV coordinates, in combination with a lighting model. The computations are executed on the programmable cores of the GPU, enabling the use of user-defined shaders. The output of this stage is the computed color of each fragment, which is then forwarded to the subsequent stage. Because primitives are not necessarily sorted, their colors must either be computed directly (the most common approach), or a depth pass or sorting step must be performed. The second component is the merging stage, which resolves the final values stored in the color buffer by integrating information from all fragments. This step is critical for producing a coherent image from multiple overlapping primitives. Although highly configurable, it is not fully programmable [3].

Visibility is commonly determined using the z-buffer algorithm, where an auxiliary buffer of the same size as the color buffer is used to store depth values. Initially, this buffer is filled with infinity (or a sufficiently large number) in all entries. As fragments are processed, their z -value is compared against the value stored in the z-buffer. If the fragment's depth is smaller, both the depth and color entries in their respective buffers are overwritten with the fragment's information. In this way, the primitive closest to the viewer is rendered. However, this algorithm is not well suited to handle semi-transparent primitives, which must instead be processed separately using blending techniques [3].

Other types of buffers are also employed in the merging stage as described by [3]. The color buffer can be extended to include an alpha channel that stores opacity values for each pixel, which are applied in pixel discarding or blending operations. The stencil buffer is an off-screen buffer used to draw primitives that serve as masks, affecting both the color buffer and the z-buffer. The framebuffer encompasses all the buffers in the system.

Finally, the screen displays the contents of the color buffer at each pixel, thereby generating the final image. Because delays may occur in displaying primitives, techniques such as double buffering are employed. In this approach, a back buffer renders the next frame off-screen, and its contents are swapped with the front buffer during the vertical retrace [3].

3.2 Rendering beyond Polygonal Meshes

In rendering, the fundamental objective is to generate images that faithfully depict a scene within the constraints of computational resources. Achieving this goal requires balancing visual fidelity, efficiency, and flexibility, often motivating the development of techniques that reduce runtime and memory consumption. Triangular meshes provide a robust geometric representation that enables accurate visualization from arbitrary viewpoints without the need for on-the-fly reconstruction. However, this representation entails significant storage requirements and can limit rendering performance. Depending on the application, different priorities may take precedence: some emphasize geometric accuracy, others visual realism, interactivity, or compactness

of representation. In contexts where polygonal models are not optimal, alternative representations such as volumetric or point-based methods offer more efficient or flexible solutions [3].

3.2.1 Volumetric Rendering

Volumetric rendering creates an image by integrating radiance along rays cast from a virtual camera into a scene. Each ray is parameterized as

$$\mathbf{r}(t) = \mathbf{o} + t\mathbf{d},$$

where $\mathbf{o}$ is the camera position, $\mathbf{d}$ is the ray direction, and $t \in [t_{\text{near}}, t_{\text{far}}]$ is a parameter that specifies the distance along the ray between the near and far planes [11].

The color observed along a ray is computed as

$$C(\mathbf{r}) = \int_{t_{\text{near}}}^{t_{\text{far}}} T(t) \sigma(\mathbf{r}(t)) c(\mathbf{r}(t), \mathbf{d}) dt,$$

where $T(t)$ denotes the transmittance, i.e., the fraction of light that reaches the point $\mathbf{r}(t)$ from the ray origin without being absorbed, $\sigma(\mathbf{r}(t))$ is the volume density that controls the opacity of the medium, and $c(\mathbf{r}(t), \mathbf{d})$ is the emitted color at that point, which may depend on the viewing direction $\mathbf{d}$. A common numerical approach to approximate this integral is ray marching, which samples the volume at discrete intervals along the ray [11].

3.2.2 Point-Based Rendering

Point-based rendering converts a set of discrete samples (a point cloud) into pixels on a raster image. Each point contributes to the final image according to its position, normal, and radiance information. However, this discrete representation introduces certain limitations, such as the possible appearance of holes when the point density is insufficient in parts of the scene, and rendering artifacts can arise from the discrete sampling and blending of points [11].

Several approaches to point-based rendering have been proposed, differing primarily in how the space between points is managed, the data structures used to represent them, and the strategies employed for traversal and filtering. Because point-based datasets can contain a large number of samples, hierarchical structures are generally preferred for storage and visualization, as they enable efficient traversal, culling, and level-of-detail refinement. Examples of such approaches include QSplat, which introduced the concept of splats as circular or elliptical surface elements with an associated radius to achieve smooth reconstruction; surfels, where points represent small surface patches of an object; and layered point clouds, which extend the representation to capture visibility and transparency effects across multiple depth layers [3].

3.2.3 Radiance Fields and Image-Based Scene Reconstruction

Image-based three-dimensional scene reconstruction aims to transform a collection of images or video sequences capturing a real-world environment into a digital 3D representation that can be computationally analyzed, manipulated, and rendered. This paradigm underlies modern view synthesis techniques, where the goal is not only to recover scene geometry but to learn how light interacts with it across viewpoints [11].

Early approaches primarily focused on light field representations, attempting to reproduce the fundamental visual characteristics of a scene. These methods relied on densely sampled input views and strictly controlled

capture configurations. While they achieved convincing results under constrained conditions, their dependence on regular sampling patterns and fixed illumination limited their generalization to complex geometries and dynamic lighting environments [11].

Radiance fields formalize the distribution of light within a scene as a continuous function of position and viewing direction. They serve as the conceptual bridge between traditional image-based rendering and modern volumetric or point-based scene representations. Radiance fields can be broadly categorized into *implicit* and *explicit* formulations, depending on whether scene properties are learned through continuous neural mappings or stored in discrete spatial data structures [11].

Implicit Representations. Implicit methods do not explicitly store geometry. Instead, they approximate a continuous function that maps spatial coordinates and viewing direction to color and density:

$$(c, \sigma) \leftarrow \text{MLP}(x, y, z, \theta, \phi).$$

A notable example is the Neural Radiance Field (NeRF), which achieves high-fidelity view synthesis by training a multilayer perceptron to regress color and density from position and direction. Although highly accurate, NeRF incurs substantial computational cost during both training and rendering, and its implicit nature complicates direct editing or compositional manipulation. NeRF employs volumetric ray marching, where color depends on the viewing direction while density remains view-independent [11].

Explicit Representations. Explicit methods, in contrast, represent radiance fields through discrete data structures such as voxel grids, surfels, or point clouds:

$$(c, \sigma) \leftarrow \text{DataStructure}(x, y, z, \theta, \phi).$$

These structures store either direction-dependent features decoded by a lightweight neural network or coefficients (e.g., spherical harmonics) directly used to compute radiance as a function of viewing direction. Explicit encodings provide improved rendering efficiency and allow more direct control over spatial organization, forming the foundation of modern point-based and Gaussian Splatting techniques [11].

4 Gaussian Splatting

Gaussian Splatting is an “explicit scene representation that models a scene using millions of learnable 3D Gaussians in space” [11, p.2]. The goal of this representation is to reconstruct a 3D scene from a limited set of 2D images, enabling the generation of novel views or perspectives beyond those provided by the input data [11].

The Gaussian Splatting training process begins with the generation of a point cloud from the set of input images. Gaussians are then centered at each of these points. Initially, the covariance matrix Σ is set according to the distribution of the nearest points to a gaussian center, while the color and opacity of the generated primitives correspond to those of their respective points. Once this initialization is complete, the Gaussians are projected (a procedure known as splatting) onto a plane oriented according to the camera position. These projections are rendered through an α -blending operation, which determines the contribution of each primitive to a specific pixel. Finally, training is performed by means of back-propagation with the generated images, in order to optimize both the characteristics of each Gaussian and the number of Gaussians in the scene [1].

One limitation of Gaussians is their lack of normals, which are required by most illumination models. Consequently, the lighting of the generated scene is static, meaning it does not change with respect to the input images. However, [1] note that computing normals from a point cloud is computationally expensive and restricts the speed at which the model can be trained.

4.1 SfM Point Clouds

Point clouds, i.e., sets of point samples in 3D space, can be obtained through different techniques such as LiDAR scanning or photogrammetry. A common approach in computer vision is to use a Structure-from-Motion (SfM) pipeline, which reconstructs 3D structure from its projections into a series of images taken from different viewpoints [6].

The most common SfM technique employed to reconstruct unordered photo collections is incremental SfM, a sequential two-stage pipeline. The first stage, correspondence search, aims to identify overlapping regions across the input image set $I = \{I_i \mid i = 1 \dots N_I\}$ and to detect projections of the same scene points in multiple images. To achieve this, distinctive features are extracted from each image, designed to remain invariant under geometric and radiometric changes. These features serve as the basis for determining which image pairs observe the same scene content without exhaustively comparing every possible pair, which would be computationally infeasible for large collections. The result is a set of candidate image pairs with tentative matches, though these correspondences are not yet guaranteed to represent the same physical points in the scene [6].

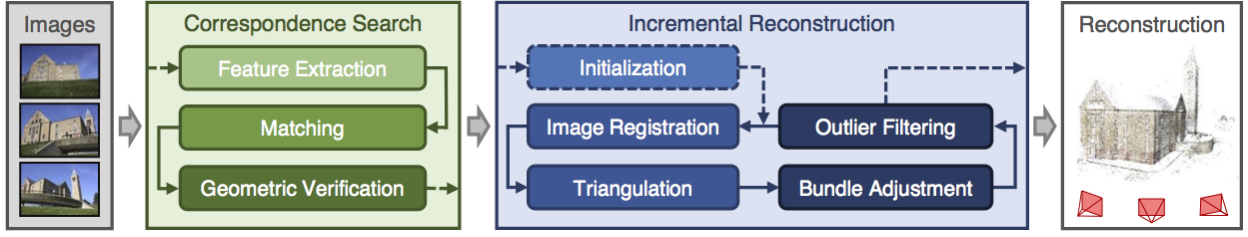


Figure 4-1: Overview of the COLMAP pipeline implementation of Incremental SfM. Figure from [6].

Because purely appearance-based matches are often unreliable, an additional geometric verification step is performed. This stage estimates the perspective transformation that aligns shared features between candidate image pairs, thereby filtering out spurious correspondences and ensuring geometric consistency. From this process, a scene graph is constructed, where vertices represent images and edges represent geometrically verified pairs. This graph provides the structural backbone for the subsequent reconstruction stage [6].

The second stage, incremental reconstruction, begins with a carefully chosen image pair used to initialize a two-view reconstruction. The choice of this initial pair is critical, as errors introduced at this step can propagate through the remainder of the process. New images are then integrated sequentially by estimating their camera pose and, when necessary, intrinsic parameters through the Perspective-n-Point problem. Correspondences between the new images and the existing reconstruction are identified, and new 3D points are added via triangulation. To improve the accuracy of both the camera and point parameters, the model is iteratively refined through bundle adjustment, which minimizes the reprojection error between the reconstructed 3D model and the original image observations [6].

The outcome of the incremental SfM pipeline is a sparse 3D reconstruction of the scene in the form of a point cloud. Each point in this cloud represents a scene point that has been consistently observed across multiple images and localized through triangulation. This representation captures both the geometry of the scene and the camera poses from which it was observed, providing a robust foundation for downstream tasks [6]. In the context of this thesis, these point clouds constitute the primary input to Gaussian Splatting, serving as the raw data from which continuous and photorealistic scene representations are constructed.

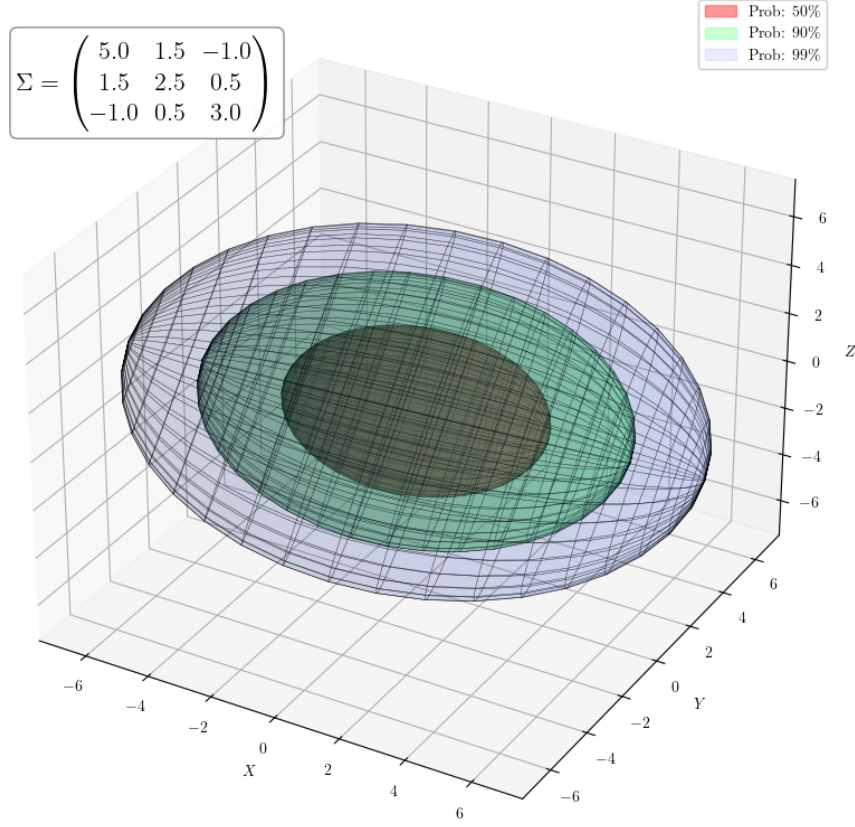
4.2 Definition of the Rendering Primitive

A Gaussian is a primitive based on an unnormalized multivariate normal function, characterized by a center μ , an opacity α , a covariance matrix Σ , and a color c , which is typically represented using spherical harmonics [11]. Formally, it is defined in [1] as

$$G(x) = \exp\left(-\frac{1}{2}(x - \mu)^\top \Sigma^{-1}(x - \mu)\right),$$

together with the aforementioned attributes. Although the domain of a Gaussian is $\mathbb{R}$, in practice computations are effectively restricted to a finite domain as seen in Figure 4-2, since $G(x)$ rapidly approaches zero for sufficiently large $\|x\|$.

Kerbl et al. [1] was the first work to implement real-time rendering of these primitives in high definition. This work argues that the use of Gaussians “results in a reasonably compact representation of the 3D scene, in part because highly anisotropic volumetric splats can be used to represent fine structures compactly” [1, p.4]. This means that if the projected Gaussians (splats) are highly optimized, they can represent complex structures with a smaller number of primitives in the scene.

Figure 4-2: Isosurfaces of a 3D Gaussian with covariance matrix Σ .

4.2.1 Color Representation via Spherical Harmonics

In GS, the radiance observed along a viewing ray can be approximated by the outgoing radiance of each Gaussian primitive [1, 11]. Assuming negligible participation of the surrounding medium, the radiance reaching the camera satisfies

$$L_i(\mathbf{c}, -\mathbf{v}) = L_o(\mathbf{p}, \mathbf{v}),$$

where $\mathbf{c}$ denotes the camera position, $\mathbf{v}$ the viewing direction, and $\mathbf{p}$ the point associated with the Gaussian [3].

Outgoing radiance depends on the incident illumination integrated over the hemisphere Ω oriented along the surface normal $\mathbf{n}$:

$$L_o(\mathbf{p}, \mathbf{v}) = \int_{\mathbf{l} \in \Omega} f(\mathbf{l}, \mathbf{v}) L_i(\mathbf{p}, \mathbf{l}) (\mathbf{n} \cdot \mathbf{l}) d\mathbf{l},$$

where f is the BRDF. The factor $(\mathbf{n} \cdot \mathbf{l})$ accounts for **foreshortening**, reducing the contribution of light arriving at grazing angles relative to the surface normal [3].

To represent the directional dependence of incident illumination efficiently, Gaussian Splatting employs Spherical Harmonics. The incident radiance is projected onto a finite set of SH coefficients, providing a compact approximation of hemispherical lighting at each Gaussian. This formulation captures local illumination effects while remaining computationally tractable, as contributions from directions opposite to the surface normal are negligible [3].

4.2.2 Gaussian Projection

After the first step of the process, the Gaussians are generated in world-space, so they must be transformed into camera-space and projection-space coordinates to continue with the procedure. According to [10], the camera transformation can be expressed as $\phi(t) = Wt + d$, where W is a rotation matrix and d is a displacement vector. Being an affine transformation, the center and covariance matrix are transformed as

$$\begin{aligned}\mu'' &= W\mu + d, \\ \Sigma'' &= W\Sigma W^\top,\end{aligned}$$

in accordance with the properties of the multivariate normal function. The projection transformation $m(x)$ is not affine. Therefore, the covariance matrix cannot be transformed in the same manner. Consequently, a first-order Taylor approximation around μ is used:

$$m(x) \approx m(\mu) + J_\mu(x - \mu),$$

where J_μ is the Jacobian of the function $m(x)$ evaluated at the point μ [1, 10]. Being an affine transformation, the transformed center and covariance matrix are

$$\begin{aligned}\mu' &= m(\mu) + J_\mu(\mu'' - \mu) = m(\mu) + J_\mu(W\mu + d - \mu), \\ \Sigma' &= J_\mu \Sigma'' J_\mu^\top = J_\mu W \Sigma W^\top J_\mu^\top.\end{aligned}$$

This projection is applied to all primitives, with their transformed parameters and z coordinates, i.e., depth values, stored in a buffer for subsequent rasterization and rendering. Projected Gaussians are referred to as Gaussians splats.

4.3 Splat Rendering

The Gaussian rendering process uses approximate α -blending, as only a single sorting of primitives is performed rather than one per pixel [1]. Primitives are first culled and sorted to skip computations for those that will not be rendered. A forward rendering pass then applies α -blending in parallel, followed by a backward pass that computes the gradients of the primitives used. To parallelize rendering, a tile-based rasterizer subdivides the pixel grid into disjoint 16×16 tiles, or patches. [1, 11].

4.3.1 Splat Culling and Sorting

First, the frustum culling process is performed to determine which primitives will be considered for rendering a specific frame. The implementation by Kerbl et al. [1] only includes Gaussians with 99% of their probability density contained within the bounds of the view frustum. Additionally, Gaussians in extreme positions—such as too close to the near plane or too far from the culling volume—are excluded, since computing their parameters may introduce numerical instability.

After the culling process, the Gaussians intersecting each patch are determined and instantiated once per patch. Instantiation refers to creating a pointer-like structure rather than duplicating the primitive. Each instance is associated with a single patch using a 64-bit key, where the first 32 bits store the patch identifier and the remaining 32 bits store the depth value of the Gaussian. These keys are used to perform sorting via a single iteration of the fast GPU radix sort algorithm, which is highly parallelized. The final result is a list of all primitive instances, sorted first by their associated patch and then by depth [1].

4.3.2 Forward Rendering Pass

The approximate alpha composition process begins with the initialization of a thread block for each patch. A thread block consists of multiple threads that share a common memory space. Within this shared memory, a sublist is created for each block, containing only the instances associated with the pixels of the corresponding patch. Each pixel then traverses this sublist sequentially, processing the instances in order of increasing depth, from front to back. During this traversal, all pixels evaluate the same instance in parallel, applying alpha composition to their values when the Gaussian intersects the pixel. If a pixel is not intersected by the current Gaussian, the instance is skipped for that pixel. This process continues iteratively until every pixel in the patch reaches a predefined target saturation value (α), ensuring that the contribution of all relevant Gaussians has been accounted for [1].

Per-pixel alpha composition is described by [11] as a modification of the process used in *NeRFs* and point-based rasterization. Given an ordered list of N Gaussians, the color of a pixel is computed as:

$$C = \sum_{i=1}^N T_i \cdot \alpha'_i \cdot c_i$$

where α'_i and T_i represent the adjusted transparency and the accumulated transmittance of the pixel, respectively. These quantities are defined as:

$$\alpha'_i = \alpha_i \cdot \exp\left(-\frac{1}{2}(x' - \mu'_i)^\top (\Sigma'_i)^{-1} (x' - \mu'_i)\right), \quad T_i = \prod_{j=1}^{i-1} (1 - \alpha'_j).$$

Here, α_i denotes the intrinsic transparency of each Gaussian. Following the description of radial interpolation in [19], the influence of a Gaussian on the final pixel color decreases with distance from its center, which is the basis for the attenuation behavior described here.

4.4 Training and Optimization

The training phase in Gaussian Splatting consists of iteratively refining the parameters of the primitives so that the rendered views match randomly selected input images as closely as possible. This process alternates between two stages: first, computing the gradients of each Gaussian with respect to the photometric loss via backpropagation; and second, updating and restructuring the primitives using stochastic gradient descent (SGD) [1, 11]. The overall training procedure is summarized in Algorithm 2, which outlines the main loop of rendering, loss computation, gradient backpropagation, and parameter refinement.

Algorithm 2 Gaussian Splatting Training Pipeline**Require:** Training dataset $\mathcal{D} = \{(I_i, C_i)\}$; maximum iterations T ; learning rate η **Ensure:** Optimized Gaussian set $\mathcal{G}$

```

1:  $\mathcal{G} := \text{InitializeGaussians}(\mathcal{D})$  ▷ Initialize Gaussians from point cloud
2: Initialize optimizer (e.g., Adam) on  $\mathcal{G}$ 
3: for  $t = 1$  to  $T$  do
4:    $(I, C) := \text{random view from } \mathcal{D}$  ▷ Stochastic step: select one view for this iteration
5:    $\hat{Y} := \text{Render}(\mathcal{G}, C)$  ▷ Compute predicted image from current Gaussians
6:    $\mathcal{L} := (1 - \lambda) \mathcal{L}_1(\hat{Y}, I) + \lambda (1 - \text{SSIM}(\hat{Y}, I))$  ▷ Compute loss for this view
7:    $\nabla_{\mathcal{G}} \mathcal{L} := \text{Backprop}(\mathcal{L}, \mathcal{G})$  ▷ Compute gradients w.r.t Gaussian parameters
8:    $\mathcal{G} = \mathcal{G} - \eta \nabla_{\mathcal{G}} \mathcal{L}$  ▷ Gradient descent update of Gaussian parameters
9:    $\text{PrimitiveRestructuring}(\mathcal{G})$  ▷ Densify/prune Gaussians
10: end for
11: return  $\mathcal{G}$ 

```

4.4.1 Gradient Computation and Backpropagation

After the forward rendering pass, gradients are computed for each Gaussian’s parameters—namely, the covariance matrix, the spherical harmonics coefficients, and the opacity value α . The photometric loss function is defined as

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

where $\mathcal{L}_1$ denotes the mean absolute error (MAE) and $\mathcal{L}_{\text{D-SSIM}}$ represents the structural dissimilarity [11, 1]. Both terms measure the per-pixel discrepancy between the rendered color and the reference color in the input views. Given a Gaussian indexed by i and a parameter θ_i , the gradient of the loss function can be written as

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial \theta_i} &= (1 - \lambda) \frac{\partial \mathcal{L}_1}{\partial \theta_i} + \lambda \frac{\partial \mathcal{L}_{\text{D-SSIM}}}{\partial \theta_i}, \\ \frac{\partial \mathcal{L}}{\partial \theta_i} &= (1 - \lambda) \sum_{p \in P} \frac{\partial \mathcal{L}_1}{\partial I(p)} \frac{\partial I(p)}{\partial \theta_i} + \lambda \sum_{p \in P} \frac{\partial \mathcal{L}_{\text{D-SSIM}}}{\partial I(p)} \frac{\partial I(p)}{\partial \theta_i}, \end{aligned}$$

where $I(p)$ denotes the rendered color at pixel p [1].

Since the influence of each Gaussian extends over multiple pixels, computing its gradients requires accessing the pixel gradients accumulated during rendering. To enable this efficiently, the lists of Gaussian instances corresponding to each patch are reused. Thread blocks are launched per patch, and these lists are loaded into shared memory to allow parallel traversal in reverse depth order, following the same structure as in the forward pass [1].

During backpropagation, it is necessary to recover the intermediate opacities computed in the forward stage. Directly storing or recomputing these values is expensive, so a weight term representing each Gaussian’s contribution to the final opacity is introduced:

$$w_i = \frac{\alpha_{\text{final}}}{\alpha_i}.$$

The error for each Gaussian is then explicitly computed and stored, reducing the performance cost associated with automatic differentiation [11, 1].

4.4.2 Primitive Restructuring

During training, the gradients obtained from the loss function are used not only to adjust existing primitives but also to adapt the structure of the representation itself. Regions exhibiting large positional gradients in view space often indicate inconsistencies in the reconstruction: areas with missing geometry (“under-reconstruction”) or regions where large Gaussians redundantly cover the scene (“over-reconstruction”). In such cases, new primitives are created or existing ones are subdivided to improve coverage and accuracy. Training proceeds until a convergence criterion is met—either a fixed number of iterations or a sufficiently low loss value.

To correct under-reconstructed regions, copies of nearby Gaussians are generated and displaced along the direction of their positional gradients. These are independent primitives rather than instances used for rendering. Conversely, in over-reconstructed areas, large Gaussians are split into two smaller ones, each scaled by a factor of 1.6, determined empirically [1]. The new primitives are incorporated into subsequent optimization iterations, and those with opacity values below a threshold are pruned [11].

Even with these refinements, over-reconstruction can persist near the input camera positions, leading to redundant geometry and unnecessary gradient computation. Additionally, Gaussians with small α values in empty regions may consume resources without contributing to the rendering. To mitigate these effects, after several iterations, the α value of each pixel is reset to zero, allowing the optimization process to reduce the transparency of relevant primitives while discarding irrelevant ones [1].

Finally, [11] note that direct optimization of the covariance matrix Σ via SGD can produce non-positive-definite results, which are physically meaningless. To avoid this, the covariance is reparameterized as

$$\Sigma = R R S^T R^T,$$

where R is a rotation matrix derived from a quaternion q , and S is a scaling matrix reconstructed from a 3D vector s . Optimizing q and s instead of Σ ensures valid, interpretable covariance matrices while maintaining stable training dynamics.

5 Methodology

This thesis quantitatively analyzes the performance and reconstruction fidelity of the original 3DGS method [1]. Our evaluation benchmark includes the standard scenes from the original work, augmented with a custom dataset featuring diverse geometry, materials, and lighting to probe the model’s limitations.

All experiments are executed using the official C++/CUDA implementation [20]. The core 3DGS algorithm remains unaltered; however, minor peripheral modifications were made to adapt the framework to our testing infrastructure and ensure reproducibility. We conduct an ablation study focused on the optimization process, systematically varying a discrete set of hyperparameters. These include the total number of training iterations, the learning rate schedule (initial and final values), and the iteration at which the adaptive Gaussian densification routine is terminated.

5.1 Experimental Setup

This section details the hardware and software configurations used for all experiments. We first describe the primary workstation used for model optimization (training) and then detail the different systems used for performance benchmarking.

5.1.1 Model Optimization Environment

All Gaussian Splatting models were trained on the dedicated workstation specified in Table 5-1.

Feature	Specification
Form Factor	Desktop Workstation
OS	Ubuntu 24.04.2 LTS (x86_64)
CPU	AMD Ryzen 7 9700X
System RAM	32 GB DDR5
GPU	NVIDIA RTX 4090
VRAM	24 GB GDDR6X
FP32 TFLOPS	≈ 82.6

Table 5-1: Hardware Configuration for Model Optimization.

The software environment relies on COLMAP for camera pose estimation and OpenGL for the real-time viewer. The core CUDA 12.1 implementation is compiled with G++ 12 via the Ninja build system. The associated Python 3.10 environment utilizes libraries including PyTorch (for visualization utilities), OpenCV (for image I/O), and plyfile.

5.1.2 Performance Benchmark Environments

To measure real-time rendering metrics (e.g., FPS) at various resolutions, the final optimized models were deployed to the workstation described previously as well as to the systems detailed in Table 5-2.

Feature	System A	System B	System C	System D
Form Factor	Laptop	Laptop	Desktop	Desktop
OS	Linux Mint 22.2	MacOS Tahoe 26	Linux Mint 22.2	Windows 10 Pro
CPU	Intel i7-8750H	Apple M3 Pro	Ryzen 5 5600X	Intel i5-12600K
System RAM	16 GB DDR3	18 GB Unified	16 GB DDR4	16 GB DDR4
GPU	GTX 1060 Mobile	Apple M3 Pro (18c)	Radeon RX 6600	RTX 3060 Ti FE
VRAM	6 GB GDDR5	Shared (Unified)	8 GB GDDR6	8 GB GDDR6
FP32 TFLOPS	≈ 4.4	≈ 7.3	≈ 8.9	≈ 16.2

Table 5-2: Hardware configurations sorted by theoretical GPU compute performance (FP32 TFLOPS).

This specification is provided to facilitate reproducibility and to contextualize the reported performance benchmarks, such as optimization time and rendering frame rates.

5.2 Data Used for Evaluation

The experimental validation of this work relies on a combination of standard benchmark datasets and custom-captured scenes. This approach allows for direct comparison with existing state-of-the-art methods while also testing the proposed techniques against real-world constraints, particularly lesser capture equipment quality and the absence of Light Detection and Ranger (LiDaR) data.

5.2.1 3DGS Dataset

The first part of the input data is sourced from a refined selection of the Gaussian Splatting dataset [20]. The provenance of this data trace back to established computer vision benchmarks, specifically the DeepBlending [8] and Tanks & Temples collections [7], ensuring the use of high-quality, widely-validated scene captures. These scenes provide reference benchmarks that allow comparison with the quantitative and qualitative results reported by [1] (hereafter referred to as the 3DGS Dataset).

5.2.2 Proprietary Scene Collection

A collection of four distinct real-world scenes was meticulously recorded and processed for reconstruction. These scenes were acquired from publicly accessible areas within the Universidad Nacional de Colombia, Bogotá campus (Sede Bogotá) and the Central Cemetery of Bogotá, and include:

- The Professors’ Room located in Building 401.
- The UNAL signage situated in the rear garden of Building 454.
- A statue prominently featured in the lobby of Building 301.
- A statue in the innermost area (President’s alley) of the Central Cemetery.

These captures were specifically designed to introduce challenging conditions often underrepresented in standard public datasets. Such conditions include: low-texture areas (e.g., planar, uniform surfaces), complex specular and reflective materials (e.g., highly polished floors), and large-scale outdoor environments featuring vast, distant backgrounds.

The raw video data is processed through a custom pipeline (detailed in the next section) to generate a final set of approximately 200 to 800 high-quality images. These images are then processed using COLMAP [6] to estimate the camera intrinsics and extrinsics, producing a final dataset ready for 3DGS training.

Dataset Video Capture

Camera configuration is a crucial determinant of success and quality in image-based 3D reconstruction pipelines. Automatic camera settings, common in commercial devices, frequently introduce artifacts such as noise, undesired brightness adjustments, and motion blur. While these features are intended to improve visual appearance or performance under challenging lighting, they are fundamentally detrimental to 3D reconstruction, as the underlying images must serve as precise, undistorted sources for data comparison [21].

To mitigate these detrimental factors and ensure photometric consistency across the capture sequence, the following camera settings were manually locked on the acquisition device (iPhone 17 Pro, capturing at 4K resolution and 60 frames per second):

- ISO (Sensor Sensitivity): Locked to 200. This low setting minimizes digital noise and maintains image fidelity [21].
- Shutter Speed (Exposure Time): Locked to 1/100 second. This relatively fast speed minimizes motion blur caused by camera movement or scene dynamics [21].
- Aperture (Iris Diaphragm): Locked to f/3.5. This fixed aperture controls the depth of field and ensures a consistent light intensity reaching the sensor, which is critical for accurate color and brightness matching during reconstruction [21].

Video Processing Pipeline

Processing video for 3D reconstruction introduces significant challenges, namely image quality, motion blur, and extreme data redundancy. A static selection of frames is often insufficient, as it may select blurry images or miss crucial viewpoints.

To address this, a multi-stage adaptive pipeline was developed to systematically determine and apply the optimal Frame Rate (FPS) and image quality threshold (P) for each scene through two controlled optimization phases.

I. The Core Processing Loop This procedure defines the sequential technical steps executed for every unique (FPS, P) parameter combination tested.

1. **Data Subsampling/Augmentation:** The initial 10 FPS frame pool (or the adapted frame pool from temporal escalation) is temporally subsampled or augmented to achieve the FPS target for the current test iteration.
2. **Adaptive Sharpness Filtering:** The percentile threshold (P) is applied using the Variance of the Laplacian to retain only the sharpest $100 - P\%$ of frames, controlling input image quality [22].
3. **Visual Deduplication:** Perceptual hashing (pHash) is applied with a defined Hamming distance of 10 to remove frames that are visually too similar, controlling input data redundancy [23].
4. **COLMAP Reconstruction:** The resulting non-redundant, quality-filtered frame set forms the input for the COLMAP pipeline (Feature Extraction, Matching, Mapping) to establish the scene’s geometric foundation.
5. **Performance Evaluation:** If reconstruction is successful (single component), the resulting sparse model is used to train a 3DGS model, recording final quantitative metrics (PSNR, SSIM, LPIPS, Training Time) and geometric stability metrics (Components, Registered Images).

II. Initial Parameter Search The goal of this phase is to determine the minimum stable parameters (FPS_{Min} , P_{Min}) required to establish a stable geometric reconstruction (i.e., a single COLMAP component) for the most challenging scene.

1. **Baseline P Search:** The Core Loop is executed with a baseline of **10 FPS**. The percentile P is incrementally tested across a defined range (**50, 60, 70, 80**).
2. **Temporal Escalation (FPS_{Min} Determination):** If a scene fails to reconstruct successfully at all tested P values at 10 FPS, the Core Loop is repeated at an escalated rate (15 FPS, then 20 FPS) until stability is achieved. This identifies the scene’s Minimum Stable Temporal Density (FPS_{Min}).
3. **Outcome Selection (P_{Min}):** The final Minimum Quality Percentile (P_{Min}) is selected as the highest P value that guarantees a single, unified component across the entire dataset, using each scene’s respective FPS_{Min} .

III. Systematic Parameter Search With stable geometry guaranteed by the parameters derived in Phase 1, this phase conducts a fine-grained search to maximize the final rendering quality and computational efficiency.

1. **Frame Rate Optimization (FPS_{Optimal}):** The P_{Min} value is fixed. The resulting frame set is systematically tested across neighboring FPS values (**5, 10, 15, 20 FPS**). The FPS that yields the highest quantitative metrics (PSNR, SSIM, LPIPS) is selected as the Optimal Frame Rate (FPS_{Optimal}).
2. **Final Quality Optimization (P_{Optimal}):** The FPS_{Optimal} is locked. A final systematic search for the optimal P value is conducted (testing 50, 60, 70, 80 percentiles).

This iterative, two-phase process ensures the final input data set is the most computationally efficient and geometrically stable foundation for high-fidelity 3DGS training.

5.3 Training Evaluation

Two main evaluation metrics are utilized: visual quality and computational performance. The performance metrics measured for visual quality are Peak Signal-to-Noise Ratio (PSNR), Structural Similarity Index (SSIM), and Learned Perceptual Image Patch Similarity (LPIPS). Each metric is computed between rendered novel views and ground-truth images withheld from training. The metrics for computational performance include the training time per iteration, total convergence time, rendering speed (measured in frames per second, FPS) and Gaussian count growth over time.

The experimental procedure is as follows:

1. **Scene preparation:** Estimate camera poses using COLMAP and convert them into the Gaussian Splatting input format. This step is only performed for the additionally captured scenes as the dataset of Kerbl et. al [20] is already in this format.
2. **Training:** Run the unmodified implementation [20] using the parameters described in Table 5-3 and Table 5-4, logging the duration of the training process.
3. **Rendering and evaluation:** Generate novel views from unseen camera positions and compute quality metrics (PSNR, SSIM, LPIPS) against ground-truth frames. This is performed via a script present in the original implementation.

To further analyze the behavior of the optimization process, a controlled series of experiments is performed varying a minimal subset of hyperparameters as detailed in Tables 5-3 and 5-4. The objective is to measure the influence of training duration and learning rate on convergence speed, reconstruction quality, and computational cost. All other parameters remain fixed to their default values to ensure comparability.

ID	Description	Iter.	LR_0^{pos}	LR_t^{pos}	Densify
exp1_base	Default (baseline)	30k	0.001	0.0001	15k
exp2_high_lr	Higher position LR	30k	0.005	0.0005	15k
exp3_short_train	Short training (efficiency)	10k	0.001	0.0001	5k
exp4_long_train	Extended training	60k	0.001	0.0001	30k

Table 5-3: Baseline Gaussian Splatting training configurations.

ID	Description	LR^{scale}	LR^{rot}
exp5_no_scaling	Baseline, no scaling	0	0.001
exp6_no_rotation	Baseline, no rotation	0.005	0
exp7_no_scaling_rotation	Baseline, no scaling/rotation	0	0

Table 5-4: Ablation configurations disabling scaling and/or rotation.

5.4 Performance Evaluation

To quantitatively evaluate the rendering performance of the trained models, the Supersplat web visualizer [24] was employed. The evaluation environment consisted of the latest stable version of the Firefox web browser.

The MangoHud utility [25], a C-based performance profiling overlay, was utilized to capture a set of hardware metrics. The metrics logged for analysis included: mean CPU utilization, mean GPU utilization, mean FPS, and the 1% (P1) and 97% (P97) frame rate percentiles.

The models resulting from the base training (experiment 1), higher position learning rate (experiment 2), and extended training (experiment 4) phases were selected for this analysis, as these models present higher variations in size and quality metrics. To ensure a robust evaluation encompassing diverse viewpoints, a set of keyframes was manually selected prior to the evaluation. Although the keyframe sequence is unique to each scene, all animations were configured for a target playback rate of 60 FPS.

The benchmarking protocol consisted of executing each scene’s animation in isolation for a 60-second (1 minute) period in a fixed 1600×900 resolution. During this interval, performance data were logged continuously. This evaluation procedure was replicated once per scene for each of the specified hardware architectures. The logged data were subsequently aggregated to compute the arithmetic mean for utilization and FPS. The 1% low percentile metric serves to quantify performance stability and characterize worst-case framerate consistency.

5.5 Image Quality Metrics

To provide a comprehensive quantitative assessment of the Gaussian Splatting reconstructions, we utilize a suite of three complementary metrics: PSNR, SSIM, and LPIPS.

We begin with the Peak Signal-to-Noise Ratio (PSNR) to measure absolute pixel-wise fidelity. Derived from the Mean Squared Error (MSE) between the render and the ground truth, this metric quantifies signal purity in decibels [26]. However, because PSNR relies strictly on pixel differences, it often fails to correlate with human visual perception.

To bridge this gap, we employ the Structural Similarity Index (SSIM) [27]. Unlike PSNR, SSIM evaluates the structural information, luminance, and contrast of the image, assessing the inter-dependencies of spatially close pixels to better approximate the perceived structural integrity of the scene.

Finally, to capture nuances that traditional signal processing metrics miss, we utilize the Learned Perceptual Image Patch Similarity (LPIPS) [28]. By calculating the distance between feature maps extracted from a pre-trained deep network (e.g., VGG), LPIPS aligns closely with human perception, making it particularly effective at quantifying high-frequency details and blurriness in neural rendering outputs.

6 Results

This chapter presents the experimental results obtained from the evaluation of the Gaussian Splatting pipeline on three custom-captured datasets. The findings are organized into three distinct sections to address perceptual quality, numerical fidelity, and computational efficiency. First, we conduct a qualitative visual analysis, presenting side-by-side render comparisons across the seven experimental configurations to highlight geometric consistency and high-frequency details. Second, we report the quantitative reconstruction metrics, summarizing the quality metrics described previously in tabular form. Finally, we present the performance benchmarks, detailing training convergence times and real-time rendering frame rates (FPS) across the hardware specifications defined in the methodology.

6.1 Qualitative Visual Analysis

Figures **6-1** through **6-4** display the rendering fidelity across the captured datasets. We contrast the Ground Truth and Baseline (Exp 1) against two distinct optimization paths. First, we examine Experiment 4 (Long Train), which recorded the highest weighted metric scores as seen in Table **6-4**. Visual evidence supports these numbers, especially regarding the preservation of complex, high-frequency color textures that appear smoothed out in the baseline. Second, we analyze Experiment 7 (No Scaling/Rotation). While its numerical performance remains close to the baseline, a qualitative inspection exposes a critical deficiency where the model struggles to converge on sharp geometric lines, causing visible blurring artifacts not captured by global metrics. For completeness, validation results on the standard 3DGS datasets are provided in Appendix A, while the full repository of high-resolution images and video demonstrations is available online¹.

6.2 Quantitative Reconstruction Results

Table **6-1** summarizes the experimental results derived from the initial FPS and percentage threshold configurations. For successful Structure-from-Motion (SfM) reconstruction via COLMAP, a minimum of three registered images is strictly required; consequently, any attempt yielding fewer than three images is categorized as an Insufficient Registration [6]. Furthermore, reconstructions exhibiting a scene coverage below 70% are also deemed insufficient, as they fail to provide adequate spatial data for robust scene representation.

Tables **6-2** and **6-3** present the results of the systematic search for optimal FPS and percentile threshold values. To facilitate a direct comparison across heterogeneous metrics, we normalized the Peak Signal-to-

¹Supplementary material and high-resolution comparisons: <https://nikoduke-doo.github.io/Gaussian-Splatting-Thesis-Materials/>



GT (Full)



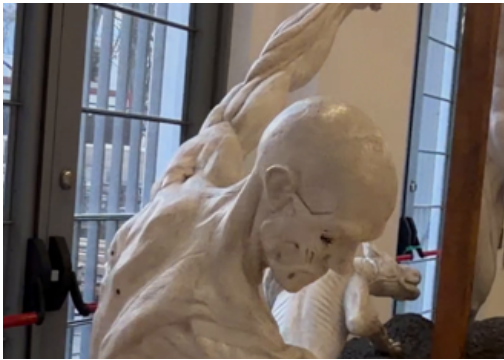
Exp. 1 Baseline (Full)



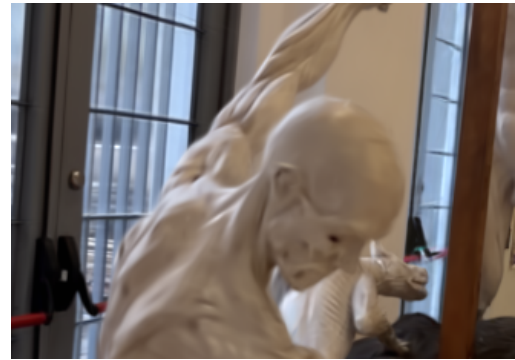
Exp. 4 Long Train (Full)



Exp. 7 No Scaling/Rotation (Full)



(a) GT Detail



(b) Exp. 1 Baseline Detail



(c) Exp. 4 Long Train Detail



(d) Exp. 7 No Scaling/Rotation Detail

Figure 6-1: Visual progression of Estatuas Scene. **Top:** Full scene geometry. **Bottom:** High-frequency texture details.



GT (Full)



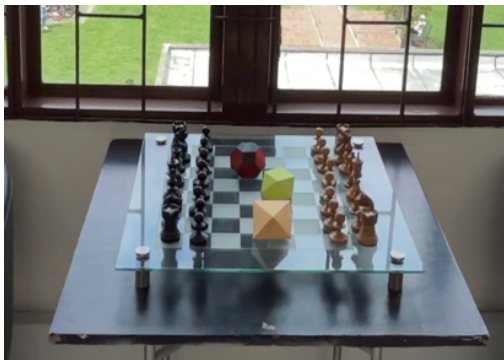
Exp. 1 Baseline (Full)



Exp. 4 Long Train (Full)



Exp. 7 No Scaling/Rotation (Full)



(a) GT Detail



(b) Exp. 1 Baseline Detail



(c) Exp. 4 Long Train Detail



(d) Exp. 7 No Scaling/Rotation Detail

Figure 6-2: Visual progression of Sala Profes Scene. **Top:** Full scene geometry. **Bottom:** High-frequency texture details.



GT (Full)



Exp. 1 Baseline (Full)



Exp. 4 Long Train (Full)



Exp. 7 No Scaling/Rotation (Full)



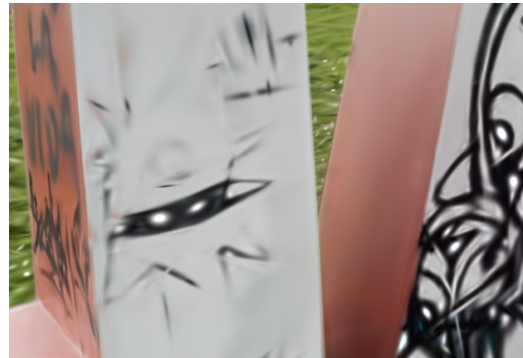
(a) GT Detail



(b) Exp. 1 Baseline Detail



(c) Exp. 4 Long Train Detail



(d) Exp. 7 No Scaling/Rotation Detail

Figure 6-3: Visual progression of UNAL Scene. **Top:** Full scene geometry. **Bottom:** High-frequency texture details.



GT (Full)



Exp. 1 Baseline (Full)



Exp. 4 Long Train (Full)



Exp. 7 No Scaling/Rotation (Full)



(a) GT Detail



(b) Exp. 1 Baseline Detail



(c) Exp. 4 Long Train Detail



(d) Exp. 7 No Scaling/Rotation Detail

Figure 6-4: Visual progression of Virgin Scene. **Top:** Full scene geometry. **Bottom:** High-frequency texture details.

Scene	FPS	P-Thr.	COLMAP Result	Reg. Imgs.	Inp. Imgs.	% Cov.
Estatuas	10	50	Minimal Registration	4	421	0.95%
		60	Minimal Registration	3	343	0.87%
		70	Minimal Registration	6	254	2.36%
		80	Insufficient Registration	2	168	1.19%
	15	50	Success	450	450	100.00%
SalaProfe2	10	50	Success	724	724	100.00%
UNAL	10	50	Success	422	422	100.00%
Virgen	10	50	Success	363	363	100.00%

Table 6-1: Results of the initial P value search for the captured dataset.

Noise Ratio (PSNR) to a $[0, 1]$ range. Since observed PSNR values fell strictly between 10 and 30, we applied Min-Max normalization as follows:

$$PSNR_N = \frac{PSNR - 10}{30 - 10}.$$

Furthermore, we harmonized the directionality of the metrics to ensure that higher values consistently represent superior reconstruction quality. Consequently, the Learned Perceptual Image Patch Similarity (LPIPS) was inverted:

$$LPIPS_I = 1 - LPIPS.$$

Finally, we computed a Weighted Average (W.A) to aggregate these metrics into a single score. Weights were assigned based on the metric’s alignment with human visual perception, prioritizing LPIPS and SSIM over pixel-wise accuracy:

$$W.A = 0.6 \times LPIPS_I + 0.25 \times SSIM + 0.15 \times PSNR_N.$$

Scene	FPS	P-Thr.	Rec. Imgs.	Inp. Imgs.	% Cov.	PSNR	SSIM	LPIPS	PSNR_N	LPIPS_I	W.A
Estatuas	5	50	3	366	0.82%	12.7366314	0.5560441	0.4606386	0.1368316	0.5393614	2.3731226
	10	50	4	421	0.95%	13.9805021	0.5065180	0.5960733	0.1990251	0.4039267	2.4660608
	15	50	450	450	100.00%	27.4168167	0.9201944	0.1523298	0.8708408	0.8476702	4.8511732
	20	50	460	460	100.00%	27.6024742	0.9212830	0.1512245	0.8801237	0.8487755	4.8799572
SalaProfe2	5	50	587	587	100.00%	25.4259319	0.8783633	0.2165233	0.7712966	0.7834767	4.5035666
	10	50	724	724	100.00%	25.6444321	0.8790168	0.2185305	0.7822216	0.7814695	4.5353007
	15	50	783	783	100.00%	25.4590034	0.8786860	0.2166200	0.7729502	0.7833800	4.5085500
	20	50	802	802	100.00%	25.3966675	0.8779916	0.2166468	0.7698334	0.7833532	4.4990099
UNAL	5	50	337	337	100.00%	20.4952965	0.5683851	0.4020253	0.5247648	0.5979747	3.5751756
	10	50	422	422	100.00%	20.6263771	0.5797650	0.3948427	0.5313189	0.6051573	3.6019922
	15	50	459	459	100.00%	20.8600693	0.5919678	0.3887753	0.5430035	0.6112247	3.6437372
	20	50	2	475	0.42%	None	None	None	None	None	None
Virgen	5	50	298	298	100.00%	20.9613686	0.6820502	0.3170133	0.5480684	0.6829867	0.6625148
	10	50	363	363	100.00%	20.6586647	0.6821167	0.3225407	0.5329332	0.6774593	0.6569447
	15	50	388	388	100.00%	21.032692	0.6867616	0.3215096	0.5516346	0.6784904	0.6615298
	20	50	447	447	100.00%	20.7362099	0.680894	0.3293996	0.5368105	0.6706004	0.6531053

Table 6-2: Systematic Search for Optimal Frame Rate ($FPS_{Optimal}$) Using Baseline Quality

Finally, Tables 6-4 and 6-5 present the outcomes of the hyperparameter sensitivity analysis and ablation studies for both datasets. These tests aim to explicitly map the relationship between specific configuration adjustments and overall model behavior. Beyond the previously defined W.A score, we record the final model size in Megabytes (MB) and the training duration (T.D.) in minutes to provide a holistic view of system performance. Ultimately, these auxiliary metrics allow for a rigorous characterization of the inherent trade-offs between reconstruction fidelity and computational resource consumption.

Scene	FPS	P-Thr.	Rec. Imgs.	Imp. Imgs.	% Cov.	PSNR	SSIM	LPIPS	PSNR_N	LPIPS_I	W.A.
Estatuas	20	50	460	460	100.00%	27.6024742	0.9212830	0.1512245	0.8801237	0.8487755	4.8799572
		60	377	377	100.00%	27.1740417	0.9179310	0.1506978	0.8587021	0.8493022	4.8151703
		70	287	287	100.00%	27.1082916	0.9201243	0.1450628	0.8554146	0.8549372	4.8092371
		80	6	184	3.26%	22.0181122	0.8360435	0.1969829	0.6009056	0.8030171	3.9935380
SalaProfe2	10	50	724	724	100.00%	25.6444321	0.8790168	0.2185305	0.7822216	0.7814695	4.5353007
		60	574	574	100.00%	25.5171375	0.8796084	0.2120057	0.7758569	0.7879943	4.5202693
		70	432	432	100.00%	25.3724136	0.8802330	0.2038534	0.7686207	0.7961466	4.5036083
		80	2	280	0.71%	None	None	None	None	None	None
UNAL	15	50	459	459	100.00%	20.8600693	0.5919678	0.3887753	0.5430035	0.6112247	3.6437372
		60	371	371	100.00%	20.4873142	0.5649992	0.4039285	0.5243657	0.5960715	3.5719898
		70	303	303	100.00%	20.3172455	0.5543801	0.4089879	0.5158623	0.5910121	3.5407891
		80	209	209	100.00%	20.0889797	0.5287286	0.4184182	0.5044490	0.5815818	3.4944782
Virgen	5	50	298	298	100.00%	20.9613686	0.6820502	0.3170133	0.5480684	0.6829867	0.6625148
		60	235	235	100.00%	20.337038	0.6599216	0.3266151	0.5168519	0.6733849	0.6465391
		70	184	184	100.00%	20.8210526	0.6826481	0.3053774	0.5410526	0.6946226	0.6685935
		80	122	122	100.00%	18.4858475	0.6151936	0.3533717	0.4242924	0.6466283	0.6054192

Table 6-3: Systematic Search for Optimal Threshold ($P_{Optimal}$) Using $FPS_{Optimal}$

Model	Experiment	Size (Mb)	T. D. (min.)	PSNR	SSIM	LPIPS	PSNR_N	LPIPS_I	W.A.
Estatuas	exp1_base	309	22.74	27.6024742	0.9212830	0.1512245	0.8801237	0.8487755	0.8716046
	exp2_high_position_lr	129	17.69	25.1917114	0.8593675	0.2597074	0.7595856	0.7402926	0.7729553
	exp3_short_train	145	6.94	25.3653488	0.8703191	0.2408982	0.7682674	0.7591018	0.7882810
	exp4_long_train	407	55.92	28.0459976	0.9259167	0.1382958	0.9022999	0.8617042	0.8838467
	exp5_no_scaling	310	22.86	27.5680103	0.9213473	0.1503018	0.8784005	0.8496982	0.8719158
	exp6_no_rotation	310	22.78	27.5355816	0.9210359	0.1503054	0.8767791	0.8496946	0.8715926
	exp7_no_scaling_rotation	311	22.71	27.5875130	0.9210785	0.1503517	0.8793757	0.8496483	0.8719650
SalaProfe2	exp1_base	290	23.19	25.3724136	0.8802330	0.2038534	0.7686207	0.7961466	0.8130393
	exp2_high_position_lr	171	19.27	21.3251953	0.7951664	0.3393732	0.5662598	0.6606268	0.6801066
	exp3_short_train	123	7.9	21.5399189	0.7943671	0.3368604	0.5769959	0.6631396	0.6830249
	exp4_long_train	458	55.47	25.7390289	0.8848816	0.1855632	0.7869514	0.8144368	0.8279252
	exp5_no_scaling	291	23.21	25.3596954	0.8798267	0.205739	0.7679848	0.7942610	0.8117110
	exp6_no_rotation	293	23.1	25.3926754	0.8798519	0.2053699	0.7696338	0.7946301	0.8121861
	exp7_no_scaling_rotation	290	23.19	25.4240761	0.8798907	0.2055793	0.7712038	0.7944207	0.8123057
UNAL	exp1_base	687	25.15	20.8600693	0.5919678	0.3887753	0.5430035	0.6112247	0.5961773
	exp2_high_position_lr	576	21.79	18.5045223	0.4501466	0.5549939	0.4252261	0.4450061	0.4433242
	exp3_short_train	357	7.34	18.4989414	0.4525481	0.5526867	0.4249471	0.4473133	0.4452671
	exp4_long_train	991	64.03	21.4844913	0.6590187	0.3174144	0.5742246	0.6825856	0.6604397
	exp5_no_scaling	685	25.08	20.8013973	0.5909807	0.3896118	0.5400699	0.6103882	0.5949886
	exp6_no_rotation	688	25.02	20.8484097	0.5917671	0.3891175	0.5424205	0.6108825	0.5958343
	exp7_no_scaling_rotation	684	25.13	20.7583275	0.5903484	0.3894634	0.5379164	0.6105366	0.5945965
Virgen	exp1_base	682	22.69	20.8210526	0.6826481	0.3053774	0.5410526	0.6946226	0.6685935
	exp2_high_position_lr	367	16.68	18.515173	0.5314608	0.48054	0.4257587	0.5194600	0.5084050
	exp3_short_train	239	5.77	18.7655067	0.5463341	0.4657934	0.4382753	0.5342066	0.5228488
	exp4_long_train	1136	63.47	21.4233761	0.7168692	0.2578963	0.5711688	0.7421037	0.7101548
	exp5_no_scaling	661	22.42	20.625351	0.6784014	0.3090041	0.5312676	0.6909959	0.6638880
	exp6_no_rotation	694	22.81	20.8236446	0.6831425	0.3048784	0.5411822	0.6951216	0.6690359
	exp7_no_scaling_rotation	687	22.74	20.7613506	0.681228	0.3063647	0.5380675	0.6936353	0.6671983

Table 6-4: Performance Metrics, Size and Training Duration of the Models generated from the Custom Dataset.

Model	Experiment	Training Duration	PSNR	SSIM	LPIPS	PSNR_N	LPIPS_I	W.A.
drjohnson	exp1_base	16.79	27.0498276	0.8472588	0.3501876	0.8524914	0.6498124	0.7295758
	exp2_high_position_lr	12.36	17.0750141	0.7000058	0.5333327	0.3537507	0.4666673	0.5080644
	exp3_short_train	4.29	23.2986412	0.7743300	0.4843340	0.6649321	0.5156660	0.6027219
	exp4_long_train	48.42	27.3497639	0.8591927	0.3216452	0.8674882	0.6783548	0.7519343
	exp5_no_scaling	16.77	27.0470619	0.8464864	0.3492797	0.8523531	0.6507203	0.7299067
	exp6_no_rotation	16.73	27.0889549	0.8473652	0.3492811	0.8544477	0.6507189	0.7304398
	exp7_no_scaling_rotation	16.7	27.1338329	0.8470345	0.3494340	0.8566916	0.6505660	0.7306020
playroom	exp1_base	18.52	28.8757401	0.8874397	0.2897951	0.9437870	0.7102049	0.7895509
	exp2_high_position_lr	14.37	25.0161629	0.8185983	0.4328181	0.7508081	0.5671819	0.6575799
	exp3_short_train	3.82	25.3871632	0.8253890	0.4213734	0.7693582	0.5786266	0.6689269
	exp4_long_train	54.6	28.9628677	0.8917471	0.2675386	0.9481434	0.7324614	0.8046351
	exp5_no_scaling	18.57	28.9075069	0.8876138	0.2894638	0.9453753	0.7105362	0.7900315
	exp6_no_rotation	18.5	28.8840256	0.8864236	0.2892924	0.9442013	0.7107076	0.7896607
	exp7_no_scaling_rotation	18.52	28.9058590	0.8882113	0.2892547	0.9452930	0.7107453	0.7902939
train	exp1_base	7.88	20.0800629	0.6910232	0.3560724	0.5040031	0.6439276	0.6347128
	exp2_high_position_lr	6.47	17.4200363	0.5447433	0.5416659	0.3710018	0.4583341	0.4668366
	exp3_short_train	2.3	17.3909760	0.5513140	0.5319946	0.3695488	0.4680054	0.4740641
	exp4_long_train	21.01	20.7808781	0.7311215	0.2976150	0.5390439	0.7023850	0.6850680
	exp5_no_scaling	7.87	19.9642696	0.6914065	0.3552035	0.4982135	0.6447965	0.6344615
	exp6_no_rotation	7.89	19.9264336	0.6915606	0.3540717	0.4963217	0.6459283	0.6348954
	exp7_no_scaling_rotation	7.87	19.9791336	0.6905081	0.3559531	0.4989567	0.6440469	0.6338987
truck	exp1_base	15.26	24.5138988	0.8452547	0.2039191	0.7256949	0.7960809	0.7978165
	exp2_high_position_lr	13.46	21.4738770	0.7030769	0.4201930	0.5736939	0.5798070	0.6097075
	exp3_short_train	3.46	21.6481285	0.7215211	0.3960764	0.5824064	0.6039236	0.6300954
	exp4_long_train	36.38	24.9573212	0.8635048	0.1618823	0.7478661	0.8381177	0.8309267
	exp5_no_scaling	15.35	24.5239258	0.8458745	0.2040820	0.7261963	0.7959180	0.7979489
	exp6_no_rotation	15.27	24.4674110	0.8448001	0.2051749	0.7233706	0.7948251	0.7966007
	exp7_no_scaling_rotation	15.35	24.5249138	0.8458001	0.2037885	0.7262457	0.7962115	0.7981138

Table 6-5: Performance Metrics, Size and Training Duration of the Models generated from the 3DGS Dataset, Sourced from Tanks and Temples [7] and Deepblending [8].

6.3 Performance Benchmarks

Tables 6-6 and 6-7 detail the quantitative results of the performance benchmarks. While the Average FPS indicates the model’s overall rendering throughput, the 1% Low FPS serves as a critical metric for assessing frame rate stability. This latter metric captures transient drops in performance that lead to visible micro-stuttering; consequently, a higher 1% Low value corresponds to better frame pacing and a perceptually smoother rendering experience.

Hardware	Experiment	Estatuas		SalaProfe2		UNAL		Virgen	
		Avg. FPS	1% Low FPS	Avg. FPS	1% Low FPS	Avg. FPS	1% Low FPS	Avg. FPS	1% Low FPS
GTX 1060 (Laptop)	1	32.23	20.25	43.06	28.37	39.45	24.66	48.94	24.41
	2	44.74	31.33	44.73	27.42	35.93	22.11	53.25	28.7
	4	28.53	18.55	36.12	22.88	21.53	15.29	26.96	14.44
M3 Pro (Laptop)	1	93.98	27.59	110.61	37.94	89.88	27.9	59.99	59.31
	2	113.84	39.56	110.53	33.67	98.04	30.01	60	58.98
	4	83.47	23.99	97.67	25.96	57.04	17.07	54.03	20
RX 6600 (Desktop)	1	234.11	57.34	322.36	62.91	225.87	45.95	281	64.42
	2	250.66	80.52	306.36	58.88	234.2	48.54	334.65	75.4
	4	206.34	45.5	263.57	47.62	159.61	31.1	223.41	36.13
RTX 3060ti (Desktop)	1	177.08	89.93	176.79	60.02	177.34	89.93	176.53	60.02
	2	177.4	89.93	177.29	89.93	177.94	89.93	176.64	60.02
	4	176.45	60.02	175.48	60.02	177	89.93	175.37	60.02
RTX 4090 (Desktop)	1	275.31	65.19	252.5	66.23	208.26	68.59	392.89	144.93
	2	346.94	63.29	286.66	64.6	199.67	67.44	423.6	181.6
	4	251.83	65.79	240.15	63.89	233.74	58.95	367.37	121.95

Table 6-6: Performance Metrics for Models generated from the Custom Dataset.

To ensure these values accurately reflect real-world performance rather than outliers, the benchmarks were conducted over a standardized camera trajectory that weights both geometry-heavy and empty regions of the scene equally. Furthermore, significant disparities between the Average and the 1% Low metrics often reveal underlying bottlenecks, such as memory bandwidth saturation or Gaussian sorting overheads. These specific performance penalties typically manifest only during high-complexity view directions which a simple average frame rate would effectively obscure.

Hardware	Experiment	drjohnson		playroom		train		truck	
		Avg. FPS	1% Low FPS	Avg. FPS	1% Low FPS	Avg. FPS	1% Low FPS	Avg. FPS	1% Low FPS
GTX 1060 (Laptop)	1	45.06	28.37	34.37	22.94	47.85	32.78	39.52	25.07
	2	34.79	18.75	35.22	26.46	61.73	42.98	32.69	21.52
	4	30.19	14	26.04	17.81	33.36	25.78	35.27	22.8
M3 Pro (Laptop)	1	106.56	32.17	99.16	30.62	115.63	59.95	86.4	26.61
	2	93.36	19.31	110.25	38.76	115.46	59.95	97.69	29.99
	4	86.64	16.18	77.54	22.69	100.18	30.24	76	23.99
RX 6600 (Desktop)	1	314.51	55.99	243.28	57.8	268.62	75.99	255.47	53.58
	2	285.24	34.75	281.97	67.02	279.75	109.53	222.27	55.43
	4	189.88	29.94	206.61	41.56	247.1	59.74	238.55	47.83
RTX 3060ti (Desktop)	1	137.95	36	126.97	25.72	146	36	153.97	36
	2	112.45	18	123.32	29.99	166.43	59.95	132.46	29.99
	4	110.32	22.5	108.4	25.71	118.88	29.99	141.41	36
RTX 4090 (Desktop)	1	247.22	66.49	227.86	64.77	301	64.27	252.27	63.86
	2	302.97	63.94	290.86	64.6	340.9	63.21	219.54	67.57
	4	206.22	68.97	233.72	64.43	260.56	65.79	212.83	65.32

Table 6-7: Performance Metrics for Models generated from the 3DGS Dataset.

7 Discussion and Final Comments

This chapter provides a detailed interpretation of the experimental results. We begin by analyzing the impact of input sampling and filtering strategies on the fidelity of the GS reconstruction. Subsequently, we conduct a sensitivity analysis to determine how parameter variations influence the final output quality. We also evaluate the efficacy of exhaustive parameter optimization for enhancing performance. Finally, we discuss the limitations of the current study, outline potential avenues for future research, and present our concluding remarks.

7.1 Dynamics of Input Sampling and Pre-processing

The results presented in Tables 6-2 and 6-3, and further visualized in Figures 7-1 and 7-2, suggest that the quality of GS reconstruction is heavily dependent on the initialization of the SfM point cloud. Intuitively, one might assume that increasing the input sampling rate, that is providing more images to the SfM pipeline, would linearly result in a more complete reconstruction. The expectation is that a higher density of views facilitates better triangulation, provided the computational cost remains within reasonable limits.

However, the experimental data refutes this assumption of linear correlation. Instead, we observe a non-linear, non-monotonic relationship (see Figure 7-1) governed by two opposing constraints. On the lower end, the results from the *Estatuas* scene, where the quality score doubled between 10 and 15 FPS, highlight a critical connectivity threshold. Below this level, insufficient overlap prevents the feature matching process from establishing a coherent graph, resulting in sparse Gaussian initialization.

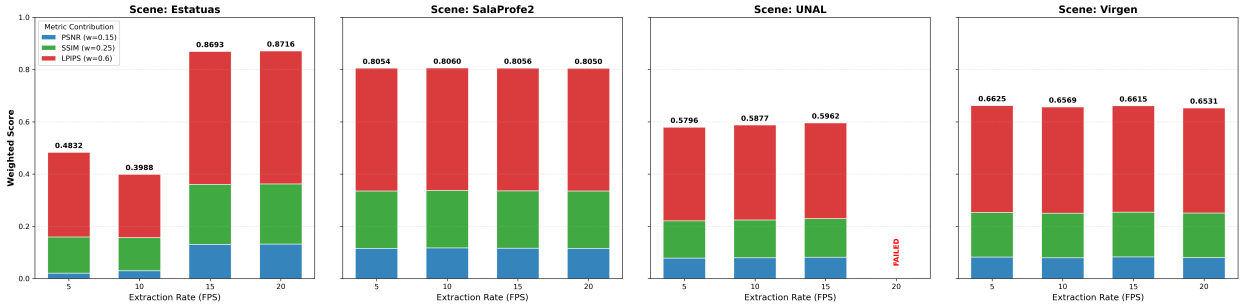


Figure 7-1: Impact of Extraction Rate on quality metrics at P_{Base} .

Conversely, the upper limit is defined by data saturation, exemplified by the pipeline failure in the *UNAL* scene at 20 FPS. In this regime, excessive sampling introduces redundancy that, rather than improving geometry, overwhelms the Bundle Adjustment process, causing registration drift or system collapse. Between these extremes, as seen in scenes like *SalaProfe2* and *Virgen*, the impact of higher sampling is negligible,

7. Discussion and Final Comments 7.2. Sensitivity of Reconstruction Fidelity to Hyperparameter Tuning confirming that once the connectivity threshold is met, additional data yields diminishing returns until it eventually becomes detrimental.

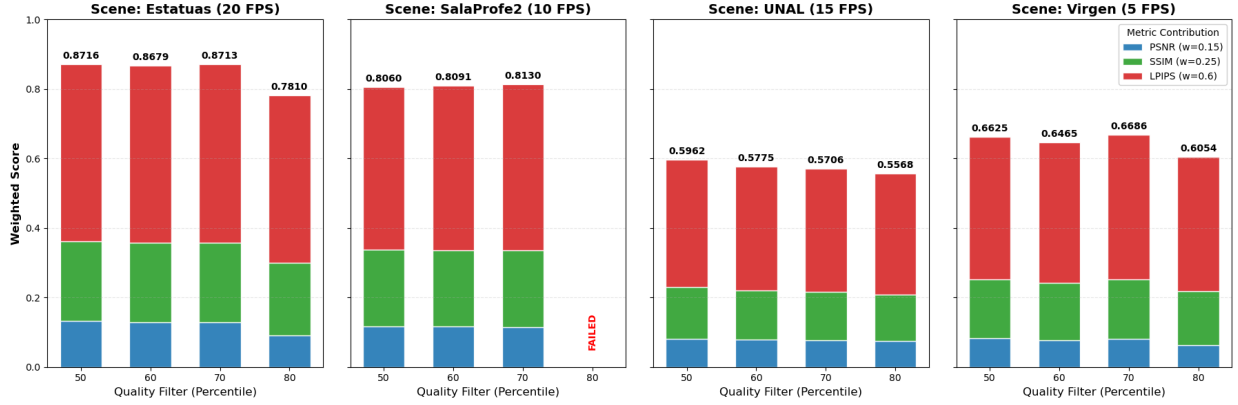


Figure 7-2: Impact of P-Threshold on quality metrics at the optimal FPS extraction rate.

Another counter-intuitive finding of this work is the negative impact of aggressive quality filtering. While standard photogrammetry pipelines often discard images with motion blur to reduce noise, our experiments with the P_{80} threshold (retaining only the sharpest 20% of frames) consistently degraded the reconstruction. This implies that blurrier frames act as essential *connective tissue* within the visual graph. The removal of these frames creates gaps that the densification heuristics of 3DGS cannot bridge, confirming that graph connectivity is a stronger predictor of success than individual pixel sharpness.

7.2 Sensitivity of Reconstruction Fidelity to Hyperparameter Tuning

The analysis of Figures 7-3 and 7-4 (based on Tables 6-6 and 6-7) reveal a clear trend: aggressive training strategies correlate with reduced reconstruction fidelity. Specifically, increasing the positional learning rate (Experiment 2) or truncating the training schedule (Experiment 3) consistently degraded quality metrics, a decline corroborated by the qualitative visual inspection in Figure B-1, which exhibits noticeable artifacts.

In contrast, the quantitative difference between the remaining configurations appears marginal. This is particularly striking regarding Experiment 4, which doubled both the training duration and memory consumption compared to the baseline. Solely based on metrics, one might conclude that extended training is inefficient, yielding diminishing returns for a significant increase in resource usage. However, this case illustrates a limitation of standard quality metrics; they fail to capture the full nuance of the reconstruction. Visual comparisons (Figures 6-1 through 6-4) reveal that Experiment 4 actually yields superior definition in complex textures and color distributions.

This discrepancy stems from the interplay between the extended training duration and the adaptive Gaussian control strategy. In the standard implementation [20], densification occurs during the first 50% of the training schedule. In designing Experiment 4, we deliberately chose to adhere to this proportional schedule rather than fixing the densification phase to the default duration. Consequently, doubling the total iterations resulted in a proportional extension of the densification window. This design choice led to a significantly higher number of Gaussians, theoretically allowing for a finer representation of the scene. However, a higher number of Gaussians increases the difficulty of the optimization. While this improves scene detail, the results suggest that the optimal training duration lies somewhere in between: requiring more iterations than the base configuration to capture fine detail, but potentially fewer than the extensive schedule employed in Experiment 4.

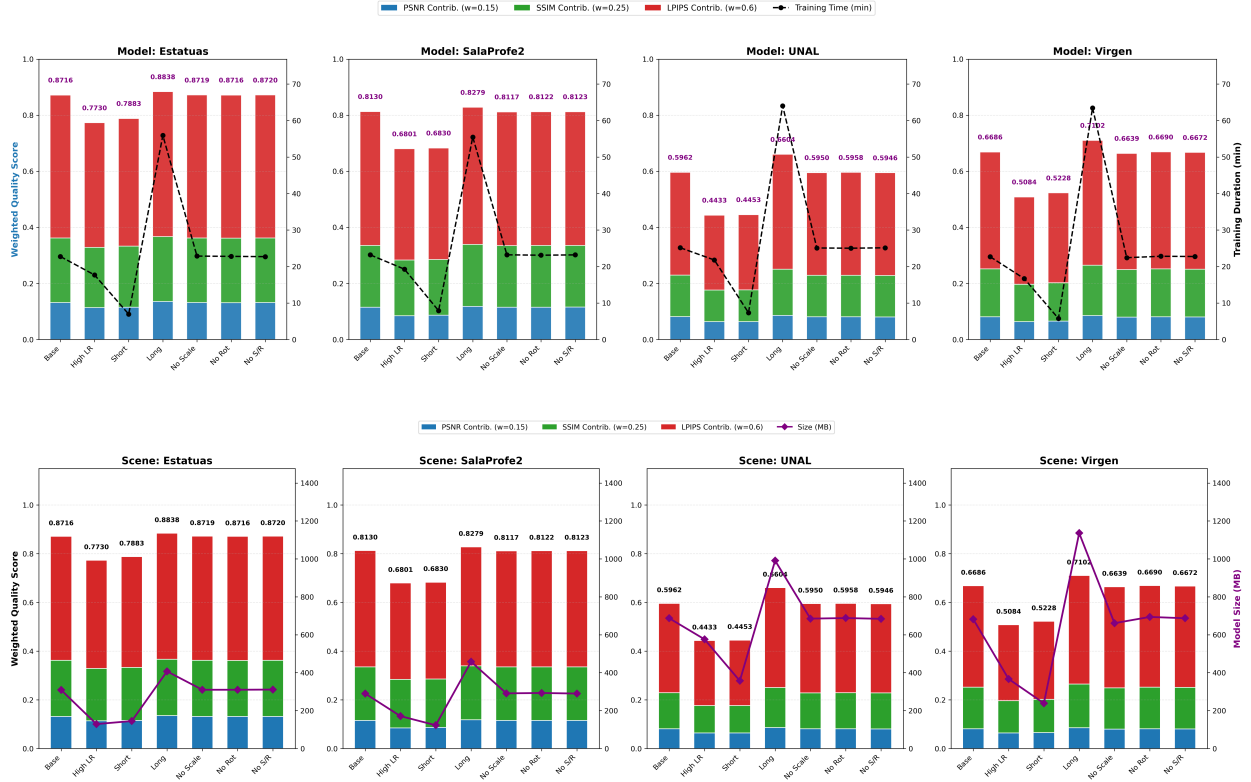


Figure 7-3: Ablation results for Custom Dataset Experiments. The top figure shows the trade-off between Weighted Quality Score and Training Duration. The bottom figure shows the trade-off between Weighted Quality Score and Model Size (Memory).

Experiments 5 through 8, which restrict primitive anisotropy, produced quality metrics surprisingly similar to the baseline (Experiment 1). On the surface, this suggests that the anisotropic parameters, considered fundamental to the 3DGS framework [1] for modeling surface alignment, might be redundant.

However, visual evidence proves otherwise. A close examination of Figures 6-1 through 6-4 highlights a loss of sharpness in scenes such as *Estatuas* and *SalaProfe2* when anisotropy is limited. The isotropic gaussians struggle to conform to sharp geometric boundaries. Thus, despite the misleading similarity in metrics, our work confirms the original premise: anisotropic training is indispensable for the accurate reconstruction of fine details.

7.3 Performance Benchmarks

The rendering performance across different hardware configurations is summarized in Figures 7-5 and 7-6. The primary metric, Mean FPS, indicates that the system achieves real-time performance thresholds (> 30 FPS) across all tested devices, including the older GTX 1060. Generally, performance improves as hardware becomes more powerful, with desktop environments consistently outperforming laptop setups.

However, this relationship is not strictly linear. Notably, the RTX 3060 Ti yielded lower average frame rates compared to the RX 6600, despite being theoretically more powerful. This result suggests that the bottleneck in this specific deployment is likely not raw GPU compute (TFLOPS), but rather the overhead introduced by the web-based viewer (WebGL) or browser-specific drivers, which may not fully utilize the

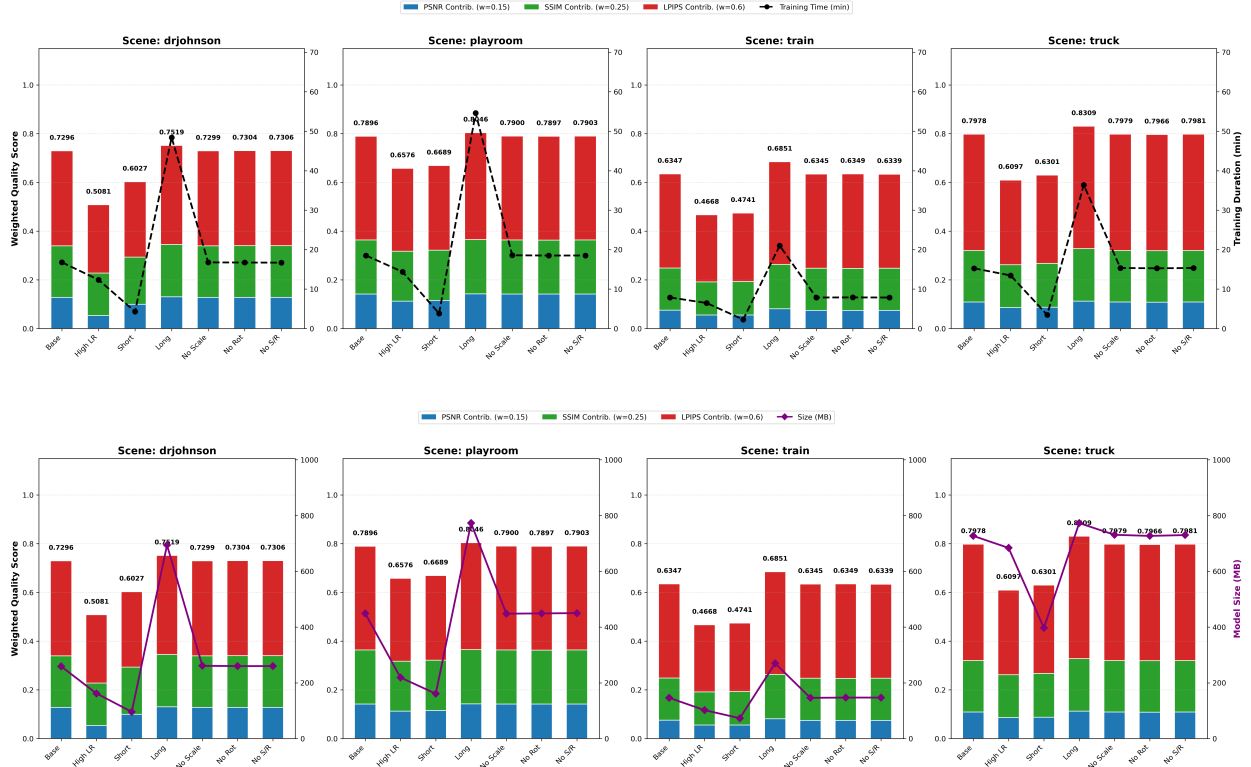


Figure 7-4: Ablation results for the 3DGS Dataset Experiments. The top figure shows the trade-off between Weighted Quality Score and Training Duration. The bottom figure shows the trade-off between Weighted Quality Score and Model Size (Memory).

architectural advantages of the NVIDIA card in this context.

Regarding the specific experimental configurations, the results demonstrate that model complexity directly impacts performance. Experiment 2, which resulted in the lowest primitive count, achieved the highest throughput. Conversely, Experiment 4, characterized by the “geometric bloat” from extended training, suffered the most significant performance drop.

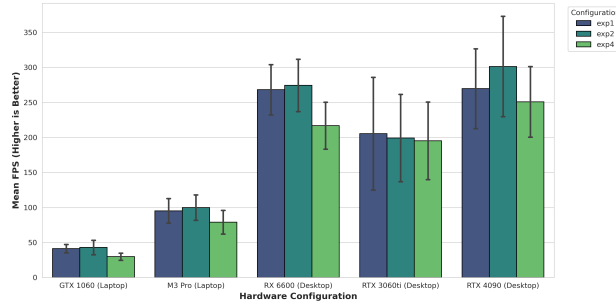


Figure 7-5: Mean FPS per testing platform, averaged across all scenes.

While average throughput varies, the analysis of rendering stability (measured via 1% Low FPS) reveals a more linear relationship with raw processing power. As shown in Figure 7-6, modern desktop and high-end laptop environments maintain fluid experiences. In contrast, older hardware struggles to maintain 20 FPS in this metric. This drop indicates noticeable lag, suggesting that while the average frame rate is acceptable, the consistency of the experience on older hardware is compromised.

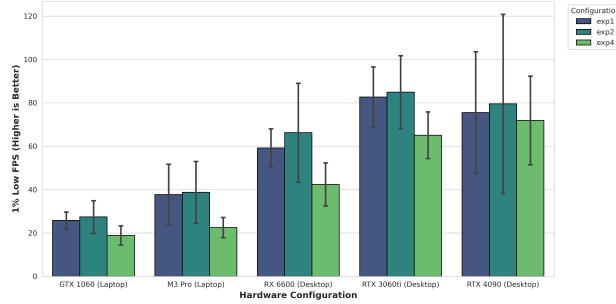


Figure 7-6: 1% Low FPS per testing platform, averaged across all scenes.

While the web-based viewer introduces overhead, the implementation remains viable for deployment. The data suggests that for lower-end systems, moving from a web-based solution to a native hardware-based renderer could likely alleviate the bottlenecks observed in the 1% low metrics, improving the user experience from functional to smooth.

7.4 Limitations

It is important to acknowledge that these findings are contextualized by the specific software framework employed. Crucially, our experimental design relies on the original implementation of 3D Gaussian Splatting [1]. Consequently, the analysis is subject to the inherent limitations of this foundational codebase and does not benefit from the regularization techniques, anti-aliasing filters, or densification improvements introduced in subsequent state-of-the-art literature.

Furthermore, the ambiguity observed in Experiment 4 highlights the necessity for more granular ablation studies. By scaling the densification schedule proportionally with training time, we introduced confounding variables: the extended optimization duration and the increased total primitive count. Future research should decouple these factors to definitively isolate whether the perceptual gains stem from extended convergence time or the increased geometric density.

Finally, our performance benchmarks must be interpreted through the lens of the deployment environment. The use of a web-based renderer imposes inherent overhead compared to native implementations. This abstraction layer does not fully utilize the underlying hardware resources. Consequently, the reported frame rates likely represent a conservative lower bound; it is plausible that lower-end devices, which struggled in our web benchmarks, could achieve viable performance thresholds if the viewer were optimized for native execution.

7.5 Future Work

Building on the findings and limitations of this study, several avenues for future research emerge. These proposals aim to refine the experimental methodology, expand the technological scope, and broaden the applicability of Gaussian Splatting.

As highlighted in the analysis of Experiment 4, future investigations should prioritize the rigorous decoupling of hyperparameters to isolate their individual contributions to reconstruction quality. Specifically, distinct experiments should be designed to separate the effects of training duration from the total number of primitives.

By fixing the densification schedule while varying the convergence time (and vice versa), researchers can definitively determine whether perceptual improvements stem from the optimization of the loss function or simply from the increased density of the point cloud.

Since this study utilized the foundational 3DGS codebase, significant value lies in evaluating state-of-the-art derivatives that have emerged subsequently. Future work should benchmark newer frameworks that introduce regularization terms, hierarchical training, or anti-aliasing filters. Comparing these advanced techniques against the baseline established here would help quantify the progress of the field and identify which algorithmic refinements yield the most tangible benefits.

To address the performance limitations observed in web-based environments, future work must benchmark native rendering pipelines (e.g., C++ or CUDA implementations) against WebGL-based viewers. Validating the hypothesis that web overhead represents a performance bottleneck is crucial. If native implementations can unlock significantly higher frame rates on the same hardware, it would validate the feasibility of deploying high-fidelity 3DGS visualizations on lower-end consumer devices, such as standard smartphones or tablets, thereby democratizing access to 3D representations of real locations and objects.

Finally, expanding the scope of data acquisition is critical for understanding the robustness of the pipeline. While this study relied on standard handheld photography, future research should systematically evaluate the impact of diverse capture modalities. This includes testing high-altitude drone footage for roof coverage, LiDAR-assisted capture for improved scale accuracy, and variable camera sensors. Understanding how different data sources, and their fusion, affect the initialization and density of the Gaussian field will be essential for developing comprehensive guidelines for its use and deployment.

7.6 Conclusions

This work presented a systematic evaluation of 3D Gaussian Splatting (3DGS) applied to architectural reconstruction, focusing on the interplay between input data quality, hyperparameter tuning, and computational efficiency. By stressing the pipeline beyond standard benchmarks, we have identified critical operational boundaries that define the viability of this technology for real-world deployment.

First, our analysis refutes the intuitive assumption that increasing input sampling rates linearly improves reconstruction quality. Instead, we identified a non-monotonic relationship governed by two opposing constraints: a *connectivity threshold* at the lower end, where insufficient overlap leads to sparse initialization, and *data saturation* at the upper end, where excessive redundancy overwhelms the Bundle Adjustment process. This finding suggests that for architectural scenes, the focus of data acquisition should shift from raw quantity to optimal graph connectivity.

Second, the parameter sensitivity analysis demonstrated that exhaustive optimization, or “brute forcing” the training process, is an inefficient strategy. While the extended training schedule (Experiment 4) yielded marginal improvements in texture fidelity, it incurred disproportionate costs in training time and memory, resulting in “geometric bloat” that compromised real-time rendering on consumer hardware. Conversely, aggressive acceleration strategies (Experiments 2 and 3) degraded quality below acceptable limits. Consequently, we conclude that the Base Configuration represents the optimal engineering trade-off, balancing visual fidelity with the strict latency requirements of interactive web visualization.

Regarding hardware deployment, our benchmarks confirm that the pipeline achieves viable real-time throughput (> 30 FPS) across a heterogeneous range of devices, including legacy mid-range GPUs. However, performance scaling proved non-linear on high-end hardware, exposing the inherent overhead of the web-based rendering environment as a primary bottleneck over raw compute power. Furthermore, while average frame rates remained robust, the analysis of 1% low metrics revealed that excessive model density significantly

compromises rendering stability on lower-end systems, introducing perceptible stuttering. This indicates that while web deployment is functional, transitioning to native hardware-based renderers is essential to unlock the full fluidity required for immersive mobile experiences.

Methodologically, this study highlights a critical dissonance between quantitative metrics (PSNR, SSIM) and perceptual quality. This was most evident in the anisotropy experiments, where metrics failed to capture the degradation of sharp geometric edges, and in the extended training experiments, where metrics understated the improvement in texture resolution. These discrepancies reinforce the necessity of visual validation in photogrammetry research, particularly when the goal is preserving the structural and aesthetic integrity of the collected scenes.

In summary, while 3D Gaussian Splatting offers a powerful alternative to traditional mesh-based photogrammetry, it is not a “black box” solution. Its success requires a calibrated approach to data sampling and a disciplined rejection of diminishing returns in training density. Future work building upon these insights, specifically targeting native mobile implementation and decoupled ablation studies, will be instrumental in transitioning this technology from experimental validation to practical, widespread adoption.

A Comparisons of 3DGS Dataset Scenes

This section provides supplementary qualitative validation of the rendering pipeline. The following figures illustrate the reconstruction quality on standard benchmark scenes from the Tanks & Temples [7] and Deep-Blending [8] datasets. These results serve as a baseline verification, demonstrating correct convergence and geometric consistency on publicly available ground truth data.



Ground Truth



Exp. 1 Baseline



Exp. 4 Long Train



Exp. 7 No Scaling/Rotation

Figure A-1: Visual progression of the drjohnson scene from the 3DGS dataset.



Ground Truth



Exp. 1 Baseline



Exp. 4 Long Train



Exp. 7 No Scaling/Rotation

Figure A-2: Visual progression of the playroom scene from the 3DGS dataset.



Ground Truth



Exp. 1 Baseline



Exp. 4 Long Train



Exp. 7 No Scaling/Rotation

Figure A-3: Visual progression of the train scene from the 3DGS dataset.



Ground Truth



Exp. 1 Baseline



Exp. 4 Long Train



Exp. 7 No Scaling/Rotation

Figure A-4: Visual progression of the truck scene from the 3DGS dataset.

B Qualitative Analysis of Suboptimal Reconstructions

This section provides supplementary visual evidence of the artifacts observed in Experiments 2 and 3. These qualitative results serve to corroborate the findings discussed previously regarding the necessity of anisotropic covariance and the requirement for sufficient training duration.



Exp. 2 on Estatuas scene.



Exp. 2 on SalaProfe2 scene.



Exp. 3 on UNAL scene.



Exp. 3 on Virgen scene.

Figure B-1: Select Renders from Experiments 2 and 3 from the Custom Dataset.

Bibliography

- [1] B. Kerbl, G. Kopanas, T. Leimkühler, and G. Drettakis, “3D Gaussian Splatting for Real-Time Radiance Field Rendering,” Aug. 2023.
- [2] K. V. Mardia, J. T. Kent, and J. M. Bibby, *Multivariate Analysis*. London: Academic Press, 1979.
- [3] T. Akenine-Möller, E. Haines, and N. Hoffman, *Real-Time Rendering*, 4th ed. Boca Raton, FL: A K Peters/CRC Press, 2018.
- [4] R. Green, “Spherical Harmonic Lighting: The Gritty Details,” in *Game Developers Conference*, San Jose, CA, 2003.
- [5] P.-P. Sloan, “Stupid Spherical Harmonics (SH) Tricks,” in *Game Developers Conference*, San Francisco, CA, 2008.
- [6] J. L. Schonberger and J.-M. Frahm, “Structure-from-Motion Revisited,” in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Las Vegas, NV, USA: IEEE, Jun. 2016, pp. 4104–4113.
- [7] A. Knapitsch, J. Park, Q.-Y. Zhou, and V. Koltun, “Tanks and temples: Benchmarking large-scale scene reconstruction,” *ACM Transactions on Graphics*, vol. 36, no. 4, 2017.
- [8] P. Hedman, J. Philip, T. Price, J.-M. Frahm, G. Drettakis, and G. Brostow, “Deep blending for free-viewpoint image-based rendering,” vol. 37, no. 6, pp. 257:1–257:15, 2018.
- [9] M. Levoy and T. Whitted, “The Use of Points as a Display Primitive,” Computer Science Department, University of North Carolina at Chapel Hill, Tech. Rep. TR85-022, Jan. 1985.
- [10] M. Zwicker, H. Pfister, J. Van Baar, and M. Gross, “EWA volume splatting,” in *Proceedings Visualization, 2001. VIS '01*. San Diego, CA, USA: IEEE, 2001, pp. 29–538.
- [11] G. Chen and W. Wang, “A Survey on 3D Gaussian Splatting,” Mar. 2025.
- [12] M. N. Gevorkyan, M. Grevorkyan, A. V. Korolkova, A. Korol’kova, D. S. Kulyabov, D. Kulyabov, L. A. Sevastianov, and L. Sevast’yanov, “Analytic projective geometry for computer graphics,” *Discrete and Continuous Models and Applied Computational Science*, vol. 33, no. 1, pp. 74–102, Jun. 2025.
- [13] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2003.
- [14] J. E. Gentle, *Matrix Algebra: Theory, Computations, and Applications in Statistics*, ser. Springer Texts in Statistics. New York: Springer, 2007.
- [15] P.-P. Sloan, J. Kautz, and J. Snyder, “Precomputed Radiance Transfer for Real-Time Rendering in Dynamic, Low-Frequency Lighting Environments,” in *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*. San Antonio, TX: ACM, 2002, pp. 527–536.
- [16] S. P. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge New York Melbourne New Delhi Singapore: Cambridge University Press, 2004.
- [17] S. Ruder. An overview of gradient descent optimization algorithms. [Online]. Available: <http://arxiv.org/abs/1609.04747>
- [18] J. D. Foley, A. van Dam, S. K. Feiner, J. F. Hughes, D. Sklar, and M. McGuire, *Computer Graphics: Principles and Practice*, 3rd ed. Boston, MA: Addison-Wesley Professional, 2013.
- [19] W. du Toit, “Radial Basis Function Interpolation,” Master’s thesis, University of Stellenbosch, Stellenbosch, South Africa, Mar. 2008.
- [20] GraphDeco - INRIA, “Graphdeco-inria/gaussian-splatting,” GitHub Repository, Jul 2023, accessed: Oct. 10, 2025. [Online]. Available: <https://github.com/graphdeco-inria/gaussian-splatting>
- [21] D. Rangelov, S. Waanders, K. Waanders, M. v. Keulen, and R. Miltchev, “Impact of camera settings on 3d reconstruction quality: Insights from nerf and gaussian splatting,” vol. 24, no. 23, 2024. [Online]. Available: <https://www.mdpi.com/1424-8220/24/23/7594>
- [22] R. Bansal, G. Raj, and T. Choudhury, “Blur image detection using Laplacian operator and Open-CV,” in *2016 International Conference System Modeling & Advancement in Research Trends (SMART)*, 2016, pp. 63–67. [Online]. Available: <https://ieeexplore.ieee.org/document/7894491>

- [23] C. Zauner, “Implementation and benchmarking of perceptual image hash functions,” Master’s Thesis, Upper Austria University of Applied Sciences, Hagenberg, 2010.
- [24] PlayCanvas, “Playcanvas/supersplat,” GitHub Repository, Oct 2023, accessed: Nov. 05, 2025. [Online]. Available: <https://github.com/playcanvas/supersplat>
- [25] flightlessmango, “Flightlessmango/mangohud,” GitHub Repository, Jan 2020, accessed: Nov. 10, 2025. [Online]. Available: <https://github.com/flightlessmango/MangoHud>
- [26] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, 4th ed. Pearson, 2018.
- [27] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, “Image quality assessment: From error visibility to structural similarity,” *IEEE transactions on image processing*, vol. 13, no. 4, pp. 600–612, 2004.
- [28] R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang, “The unreasonable effectiveness of deep features as a perceptual metric,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 586–595.