h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES

Hochschule Darmstadt

– Faculty of Mathematics and Natural Sciences and Computer Science–

How to Build 3D Models Anywhere: Photogrammetry vs. Gaussian Splatting on Mobile, Desktop and Cloud Platforms

Submitted in Partial Fulfillment of the Requirements for the Degree of

Master of Science (M.Sc.)

Submitted by

Salime Faizi

Student ID: 731994

First Examiner : Prof. Dr. Elke Hergenröther
Second Examiner : Prof. Dr. Timo Schürg

DECLARATION

Ich versichere hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die im Literaturverzeichnis angegebenen Quellen benutzt habe.

Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder noch nicht veröffentlichten Quellen entnommen sind, sind als solche kenntlich gemacht.

Die Zeichnungen oder Abbildungen in dieser Arbeit sind von mir selbst erstellt worden oder mit einem entsprechenden Quellennachweis versehen.

Diese Arbeit ist in gleicher oder ähnlicher Form noch bei keiner anderen Prüfungsbehörde eingereicht worden.

Im Rahmen der sprachlichen Überarbeitung wurde Künstliche Intelligenz eingesetzt, insbesondere DeepL.com, um Formulierungen sowie Grammatik zu verbessern.

Darmstadt, 16.10.2025

Salime Faizi

ABSTRACT

This thesis investigates the feasibility and performance of classical photogrammetry and modern differentiable scene-representation methods, with a particular focus on *3D Gaussian Splatting* (3DGS). The study systematically compares mobile applications (*Kiri Engine*, *Polycam*), desktop software (*Meshroom*, *PostShot*), and the official 3DGS source-code implementation executed on Amazon AWS. The evaluation covers a diverse range of scenarios, including static objects, architectural structures, outdoor environments, and dynamic motion sequences, to analyze visual quality, reconstruction robustness, and hardware efficiency under realistic conditions.

Quantitative metrics (PSNR, SSIM, LPIPS) were applied to source-code models, while all other tools were assessed qualitatively through structured visual evaluation. Photogrammetry demonstrated strong performance for metrically consistent, watertight object reconstructions but degraded under motion, illumination changes, or limited parallax. In contrast, 3DGS achieved visually immersive, continuous scene representations with superior consistency in dynamic and large-scale environments, though at the cost of high GPU memory demands and reduced sharpness in peripheral regions.

Across all methods, data quality, particularly image stability, parallax, and lighting uniformity, proved to be the decisive factor for reconstruction success. Mobile applications provided the lowest entry barrier and fastest turnaround times, enabling even non-expert users to generate realistic 3D models directly from smartphone captures.

Overall, the results show that high-quality 3D reconstructions are achievable under limited hardware resources and without expert knowledge. This opens new opportunities for cost-efficient 3D digitization in cultural heritage, education, and other domains where accessibility and visual realism are prioritized.

ZUSAMMENFASSUNG

Diese Masterarbeit untersucht die Machbarkeit und Leistungsfähigkeit klassischer Photogrammetrie im Vergleich zu modernen, differenzierbaren Szenenrepräsentationen, insbesondere *3D Gaussian Splatting* (3DGS). Ziel war es, die Eignung beider Ansätze unter begrenzten Hardware-Ressourcen zu bewerten und ihre Stärken und Schwächen im Hinblick auf Rekonstruktionsqualität, Robustheit und Zugänglichkeit aufzuzeigen.

Verglichen wurden mobile Anwendungen (*Kiri Engine*, *Polycam*), Desktop-Software (*Meshroom*, *PostShot*) sowie die offizielle 3DGS-Source-Code- Implementierung auf Amazon AWS. Die Evaluation umfasste verschiedene Szenarien, darunter statische Objekte, architektonische Strukturen, Außenaufnahmen und dynamische Bewegungsszenen, um die visuelle Qualität, Robustheit und Hardwareeffizienz unter realistischen Bedingungen zu analysieren.

Die Ergebnisse zeigen, dass Photogrammetrie besonders bei statischen, strukturierten Szenen metrisch präzise und geschlossene 3D-Modelle erzeugt, jedoch unter Bewegung, wechselnden Lichtbedingungen oder geringer Parallaxe an Stabilität verliert. 3DGS hingegen erzielt visuell konsistente und immersive Szenenrekonstruktionen, ist aber stark speicherintensiv und zeigt in Randbereichen Qualitätsverluste.

Entscheidend für die Ergebnisqualität sind die Aufnahmedaten selbst – insbesondere Bildstabilität, Parallaxe und gleichmäßige Beleuchtung. Mobile Anwendungen bieten dabei die niedrigste Einstiegshürde und ermöglichen auch technisch unerfahrenen Nutzerinnen und Nutzern die Erstellung realistischer 3D-Modelle direkt vom Smartphone aus.

Insgesamt verdeutlicht die Arbeit, dass hochwertige 3D-Rekonstruktionen auch mit begrenzter Rechenleistung und ohne Expertenwissen möglich sind. Dies eröffnet neue Perspektiven für die kosteneffiziente 3D-Digitalisierung in Bereichen wie Bildung und Kulturerhalt.

CONTENTS

I Thesis

1 Introduction 2
- 1.1 Relevance of 3D Reconstruction . . . 2
  - 1.1.1 Aims of the Thesis . . . 2
  - 1.1.2 Contribution . . . 3
- 1.2 Structure of the Thesis . . . 3

2 Omitted 3D Reconstruction Methods 5
- 2.1 High-End Neural Rendering Methods . . . 5
- 2.2 Newer Photogrammetry Methods . . . 6

3 Fundamentals of 3D Reconstruction: From Pixels to Geometry 7
- 3.1 Flat vs. Spatial: Understanding 3D Data Representations . . . 7
  - 3.1.1 2D vs. 3D Representation . . . 7
  - 3.1.2 Forms of representation in the 3D reconstruction . . . 7
  - 3.1.3 Voxel . . . 8
- 3.2 Decoding the Blackbox: The Math Behind 3D Reconstruction . 9
  - 3.2.1 The Pinhole Camera Model . . . 9
  - 3.2.2 Feature Detection and Description . . . 11
  - 3.2.3 Epipolar Geometry . . . 14
  - 3.2.4 RANSAC (Random Sample Consensus) . . . 16
  - 3.2.5 Triangulation - Estimating 3D Points from Image Pairs . 17
  - 3.2.6 Perspective-n-Point (PnP) . . . 19
  - 3.2.7 Bundle Adjustment (BA) . . . 21

4 Photogrammetry 23
- 4.1 Photogrammetry-Pipeline . . . 23
  - 4.1.1 Structure-from-Motion (SfM) . . . 23
  - 4.1.2 Multi-View Stereo (MVS) . . . 24
  - 4.1.3 Meshing . . . 26
  - 4.1.4 Texturing . . . 27
- 4.2 Tools and Platforms for Photogrammetry . . . 29
  - 4.2.1 Open-source software as a baseline . . . 29
  - 4.2.2 Commercial software as high-end solutions . . . 29
  - 4.2.3 Mobile applications as a recent trend . . . 30
- 4.3 Photogrammetry Limitations . . . 30

5 3D Gaussian Splatting (3DGS) 32
- 5.1 Comparison with NeRF and Photogrammetry . . . 32
- 5.2 Foundations . . . 32
  - 5.2.1 Novel View Synthesis (NVS) . . . 33
  - 5.2.2 Radiance Fields . . . 33
  - 5.2.3 3D Gaussians . . . 34
  - 5.2.4 Real-Time Rendering . . . 35
- 5.3 Method: 3D Gaussian Splatting in Detail . . . 36

5.3.1 Parametrization of the 3D Gaussians . . . 36
5.3.2 Adaptive Density Control . . . 37
5.3.3 Differentiable Rasterization . . . 38
5.3.4 Rendering Pipeline . . . 39
5.4 Practical Implementation and Tools . . . 40
5.4.1 Pre-processing with COLMAP . . . 41
5.4.2 Training Pipeline (Original Code) . . . 41
5.4.3 From Code to Application: User-Oriented Tools and Platforms . . . 42
5.5 Technical Limitations . . . 42
6 Method Selection and Evaluation Setup 44
6.1 Overview of Tested Tools . . . 44
6.2 Experimental Setup . . . 45
6.2.1 Mobile Applications . . . 45
6.2.2 Desktop Software . . . 45
6.2.3 Source Code Implementation . . . 46
6.2.4 Drone-based Data Acquisition . . . 46
6.3 Evaluation Criteria . . . 46
6.3.1 Metric Evaluation . . . 46
6.3.2 Visual Evaluation . . . 48
6.3.3 Tool Evaluation . . . 48
6.4 Experimental Design . . . 48
6.4.1 Datasets and Evaluation Protocol . . . 49
6.4.2 Experimental Procedure . . . 50
7 Results and Discussion 51
7.1 Photogrammetry Models Results . . . 51
7.1.1 Feasibility Tests on Edge-Case Datasets (Photogrammetry) . . . 51
7.1.2 Photogrammetry Visual Evaluation of Everyday Scenarios . . . 55
7.1.3 Robustness Test for Photogrammetry Citadel Model . . . 57
7.2 3D Gaussian Splatting Models Results . . . 60
7.2.1 Feasibility Tests on Edge-Case Datasets (3DGS) . . . 60
7.2.2 3DGS Models from Everyday Scenarios Datasets . . . 67
7.2.3 Robustness Tests of 3DGS Citadel Models . . . 70
7.3 Cross-Method Comparison . . . 75
7.4 Tool Evaluation . . . 78
7.5 Limitations . . . 80
8 conclusion 82

II Appendix

A Appendix 84
.1 Mathematical Details of Spherical Harmonics in 3DGS . . . 84
.1.1 Use of SH in Neural Rendering . . . 84
.1.2 Application in 3D Gaussian Splatting . . . 84
.1.3 Limitations and Advances . . . 85

.1.4 Conclusion 85
.2 Dataset Collages 85
.2.1 Everyday Scenario Datasets 85
.2.2 Edge Case Datasets 85
.2.3 External Datasets 87
.3 Tool Specifications 88
.3.1 Mobile Applications 88
.3.2 Desktop and Server Solutions 88
.4 Feasibility Criteria 89

Bibliography 97

LIST OF FIGURES

Figure 3.1 Pinhole camera model: a 3D point is projected onto the 2D image plane through the camera center. [48] . . 10
Figure 3.2 Epipolar plane and point correspondence geometry . . 15
Figure 3.3 Epipolar geometry and epipolar pencil 15
Figure 3.4 Triangulation from two views 18
Figure 3.5 Reprojection error in triangulation 18
Figure 4.1 Incremental Structure-from-Motion pipeline. [56]. . . . 24
Figure 4.2 Delaunay-based meshing via restriction: a 3D Delaunay tetrahedralization is built on the samples, and a surface is extracted by restricting to the vicinity of the underlying surface. [9]) 27
Figure 4.3 The texturing process: the mesh with chart boundaries (left), the charts as separate UV patches in the 2D texture atlas (center), and the final textured model (right). [39] 28
Figure 5.1 Illustration of projected disc rendering: surfels are represented as discs in object space (right), which are projected onto the screen space as ellipses (left). [20]. . 34
Figure 5.2 Construction of a splat primitive: a colored point primitive c is multiplied with an alpha mask $w(x, y)$, often represented as a 2D Gaussian function, resulting in a smooth splat. [20]. 35
Figure 5.3 Visualization of the first spherical harmonics functions. Blue regions indicate positive values, yellow regions negative values, while the distance from the center corresponds to the magnitude of the function. [69] 37
Figure 5.4 Adaptive Gaussian densification scheme from Kerbl et al. [33]. Top row: under-reconstruction (clone). Bottom row: over-reconstruction (split). 38
Figure 5.5 Pipeline of 3D Gaussian Splatting: starting from an Structure from Motion (SfM) point cloud, Gaussians are optimized through adaptive density control and differentiable rasterization. The final stage produces real-time renderings suitable for interactive exploration. [33]. 40
Figure 7.1 Photogrammetry-based reconstruction of the *Owl* dataset using three different tools: Kiri Engine (left), Polycam (center), and Meshroom (right). A toy owl was placed on a rotating turntable and recorded from all sides to capture the dataset. 52

Figure 7.2 Photogrammetry-based reconstructions of the *Car Ride* dataset using the *Kiri Engine* and *Polycam* apps. Recordings were taken from the passenger seat at 30–50 km/h. Screenshots are arranged chronologically from left to right. . . . 53
Figure 7.3 Photogrammetry-based reconstructions of the *Forest 360-Degree Shot* dataset. The top row shows Kiri Engine; the bottom row shows Polycam. . . . 53
Figure 7.4 Photogrammetry-based reconstructions of the *Walkway* dataset, captured while walking along a path. Top: *Kiri Engine*; bottom: *Polycam*. . . . 54
Figure 7.5 Photogrammetry-based reconstructions of Schloss Philippsruhe (*Castle Compound* dataset). Top: Kiri Engine. Bottom: Polycam. . . . 55
Figure 7.6 Photogrammetry-based reconstructions of Schloss Philippsruhe (front side): top Kiri Engine, middle Meshroom, bottom Polycam. . . . 56
Figure 7.7 Photogrammetry-based reconstructions of the *Bush* dataset. Top: Kiri Engine; middle: Polycam; bottom: Meshroom. Multiple viewpoints are shown, including side and top views. . . . 57
Figure 7.8 Polycam photogrammetry-based reconstructions of the *Citadel* dataset. . . . 58
Figure 7.9 Kiri Engine photogrammetry-based reconstructions of the *Citadel* dataset. . . . 60
Figure 7.10 Meshroom photogrammetry-based reconstructions of the *Citadel* dataset. . . . 61
Figure 7.11 3D Gaussian Splatting (3DGS) reconstructions of the *Owl* (Eule100) dataset created with four pipelines (top to bottom): Kiri Engine, Polycam, PostShot, and the 3DGS source code. The toy owl was recorded on a rotating turntable to obtain 360° coverage. . . . 62
Figure 7.12 3DGS reconstructions of the *Car Ride (100 frames)* variant. From top to bottom: Kiri Engine, Polycam, and 3DGS source code. Frames were extracted from a short car ride around Schloss Philippsruhe and are shown in capture order. . . . 64
Figure 7.13 3DGS reconstruction of the *Car Ride (video)* variant with **Kiri Engine**. The video was recorded from the passenger seat at ∼30–50 km/h. Frames are shown in chronological order (left to right, top to bottom). . . . 64
Figure 7.14 3DGS reconstructions of the *Forest 360-Degree Shot* dataset. Rows: Kiri Engine (top), Polycam (middle), 3DGS source code (bottom). Within each row, frames are ordered left to right. . . . 65

Figure 7.15 3DGS reconstructions of the *Walkway* dataset. Recordings were taken while walking with mostly forward-facing camera direction. Rows: Kiri Engine (top), Polycam, PostShot, 3DGS source code (bottom). 66
Figure 7.16 3DGS reconstructions of the *Castle Compound* dataset with four tools: Kiri Engine, Polycam, PostShot, and the 3DGS source code. 68
Figure 7.17 3DGS reconstructions of Schloss Philippsruhe (front side) with matching top views for each tool. 69
Figure 7.18 3DGS reconstructions of the *Bush* dataset (top to bottom: Kiri Engine, Polycam, PostShot, 3DGS Source Code). All methods use the same 360° capture for direct visual comparison. 71
Figure 7.19 3DGS reconstructions of the *Citadel* dataset using the **Polycam** app. Left: full model (20, 50, 100, 200 images from top to bottom). Right: corresponding detail crop of the electricity pylon in front of the fortress wall. . . . 72
Figure 7.20 3DGS reconstructions of the *Citadel* dataset using the **Kiri Engine** app. From top to bottom: 20, 50, 100, and 200 input images. 73
Figure 7.21 3DGS reconstructions of the *Citadel* dataset using **PostShot**. From top to bottom: 20, 50, 100, and 200 input images. 75
Figure 7.22 3DGS reconstructions of the *Citadel* dataset using the **Source Code** implementation. From left to right: 20, 50, 100, and 200 input images. 76
Figure .1 All frames of the **Bush** dataset as a collage (thumbnails). 86
Figure .2 Overview of the captures in the **Castle Compound** dataset. 87
Figure .3 Sample overview of the captures in the **Owl** dataset. . 91
Figure .4 All frames of the **Walkway** dataset as a collage (thumbnails). 92
Figure .5 All frames of the **Forest 360-Degree Shot** dataset as a collage (thumbnails). 93
Figure .6 All frames of the **Car Ride** dataset as a collage (thumbnails). 94
Figure .7 All frames of the **Castle Compound** dataset as a collage (thumbnails). 95
Figure .8 All frames of the **Citadel** dataset as a collage (thumbnails). 96

LIST OF TABLES

Table 6.1 Workstation specifications for desktop software testing. 45
Table 6.2 AWS EC2 specifications for source code experiments. . 46
Table 6.3 Criteria for visual evaluation of reconstructed models. 48
Table 6.4 Criteria for tool evaluation. 49
Table 7.1 Feasibility results for the *Owl* dataset (photogrammetry). 52
Table 7.2 Feasibility results for the *Car Ride* dataset (photogrammetry). 52
Table 7.3 Feasibility results for the *Forest 360-Degree Shot* dataset (photogrammetry). 53
Table 7.4 Feasibility results for the *Walkway* dataset (photogrammetry). 54
Table 7.5 Feasibility results for the *Castle Compound* dataset (photogrammetry). 54
Table 7.6 Visual evaluation of photogrammetry-based reconstructions for the *Castle Frontside* dataset. 56
Table 7.7 Visual evaluation of photogrammetry-based reconstructions for the *Bush* dataset. 57
Table 7.8 Visual evaluation of *Polycam* photogrammetry reconstructions for the *Citadel* dataset (20–200 images). . . . 58
Table 7.9 Visual evaluation of Kiri Engine photogrammetry reconstructions for the *Citadel* dataset (20–200 images). . 59
Table 7.10 Visual evaluation of Meshroom photogrammetry reconstructions for the *Citadel* dataset (20–200 images). . 59
Table 7.11 Feasibility results for the *Owl* dataset (3DGS). 62
Table 7.12 Feasibility results for the *Car Ride* dataset variants (3DGS). 63
Table 7.13 Feasibility results for the *Forest 360-Degree* dataset (3DGS). 63
Table 7.14 Feasibility results for the *Walkway* dataset (3DGS). . . . 65
Table 7.15 Feasibility results for the *Castle Compound* dataset (3DGS). 67
Table 7.16 Quantitative evaluation of 3DGS models for *Edge-Case* datasets. 67
Table 7.17 Visual evaluation of Castle Frontside 3DGS reconstructions (Kiri Engine, Polycam, PostShot). 69
Table 7.18 Visual evaluation of 3DGS reconstructions of the *Bush* dataset. 70
Table 7.19 Quantitative evaluation of 3DGS Source Code models for *Everyday Scenario* datasets. 70
Table 7.20 Visual evaluation of Polycam 3DGS reconstructions for the *Citadel* dataset (20–200 images). 71
Table 7.21 Visual evaluation of Kiri Engine 3DGS reconstructions for the *Citadel* dataset (20–200 images). 73

Table 7.22 Visual evaluation of PostShot 3DGS reconstructions for the *Citadel* dataset (20–200 images). 74
Table 7.23 Quantitative evaluation of 3DGS Source Codemodels for the *Citadel* reference series. 74
Table .1 Comparison of mobile 3D reconstruction apps [27–31, 49–52]. 88
Table .2 Comparison of selected desktop/server solutions for 3D reconstruction [16, 24, 44]. 89

LISTINGS

LIST OF ABBREVIATIONS

SIFT Scale-Invariant Feature Transform

SfM Structure from Motion

MVS Multi-View Stereo

NeRF Neural Radiance Fields

3DGS 3D Gaussian Splatting

FPS Frames per Second

SMERF Streamable Memory Efficient Radiance Fields

MipNeRF360 Multi-scale Neural Radiance Fields 360

InstantNGP Instant Neural Graphics Primitives

Plenoxels Sparse Voxel-based Radiance Fields

ZipNeRF Compressed Neural Radiance Fields

MVSNet End-to-End Deep Learning Architecture for Multi-View Stereo

CasMVSNet Cascade Cost Volume Multi-View Stereo Network

SURF Speeded-Up Robust Features

ORB Oriented FAST and Rotated BRIEF

BRISK Binary Robust Invariant Scalable Keypoints

AKAZE Accelerated KAZE Features

FED Fast Explicit Diffusion

M-LDB Modified-Local Difference Binary

DoG Difference of Gaussians

PnP Perspective-n-Point

EPnP Efficient Perspective-n-Point

BA Bundle Adjustment

NVS Novel View Synthesis

SH Spherical Harmonics

MLP Multi-Layer Perceptron

VRAM Video Random Access Memory

VOXEL Volumetric Pixel

RANSAC Random Sample Consensus

SGM Semi-Global Matching

PSNR Peak Signal-to-Noise Ratio

SSIM Structural Similarity Index

LPIPS Learned Perceptual Image Patch Similarity

AWS Amazon Web Services

Part I

THESIS

1 INTRODUCTION

1.1 RELEVANCE OF 3D RECONSTRUCTION

The accurate representation of objects and environments has always been essential for human understanding. Historical examples illustrate the consequences of missing references: medieval European artists, lacking first-hand knowledge of exotic animals such as elephants, often produced depictions that appear distorted to today's viewers [34]. This underlines the importance of authentic modeling to avoid misconceptions and to provide reliable visual knowledge. In modern contexts, realistic three-dimensional reconstructions fulfill a similar role by enabling precise documentation and interpretation of real-world structures.

The digital transformation of images or videos into 3D models has therefore become a central component of contemporary science and society. Applications span robotics [46], augmented and virtual reality [65], medical imaging [66], and the preservation of cultural heritage [18]. In particular, the digitization of monuments in conflict zones or remote regions highlights the societal value of these technologies: they help to safeguard cultural identity while offering new opportunities for analysis and education.

Education and communication represent another domain where 3D reconstruction demonstrates its relevance. Digital learning environments increasingly combine physical and virtual spaces, allowing learners to explore complex content through immersive interaction. Realistic 3D models support comprehension and engagement, thereby contributing to future-oriented learning concepts such as Future Learning Spaces [15].

At the same time, accessibility remains a critical issue. While research prototypes and commercial systems often rely on expensive hardware or specialized software, mobile and cloud-based solutions aim to broaden availability. Yet these simplified tools frequently sacrifice transparency, controllability, or visual fidelity [11, 33]. It is imperative to strike a balance between accuracy, efficiency, and accessibility. This is the fundamental challenge that underlies the investigation conducted in this thesis.

1.1.1 *Aims of the Thesis*

The aims of this thesis are:

- to identify 3D reconstruction approaches that provide a balance between model quality, user-friendliness, and cost-efficiency,
- to evaluate how modern methods such as 3DGS can be made applicable under limited hardware resources,

- to explore how different usage contexts (mobile, desktop, cloud) influence accessibility and scalability of 3D reconstruction,
- to provide orientation for cultural heritage, education, and everyday documentation by determining which methods are most suitable for non-expert users.

1.1.2 *Contribution*

This thesis contributes by:

- presenting a structured comparison between photogrammetry and 3DGS across different platforms,
- evaluating the feasibility of mobile, desktop, and cloud-based solutions under constrained hardware conditions,
- applying both quantitative image-similarity metrics and qualitative tool- and model-based criteria,
- highlighting pathways to make advanced reconstruction methods accessible and scalable for non-expert users.

In doing so, the thesis connects state-of-the-art research with practical applicability, demonstrating how complex reconstruction techniques can be translated into usable solutions in resource-limited scenarios.

1.2 STRUCTURE OF THE THESIS

The remainder of this thesis is structured as follows. Each chapter is briefly introduced below.

Chapter 2: Omitted 3D Reconstruction Methods provides an overview of 3D reconstruction approaches that are mentioned for completeness but not examined in detail in this work. It briefly introduces representative neural rendering lines (Neural Radiance Fields (NeRF) family, Streamable Memory Efficient Radiance Fields (SMERF) and newer deep-learning Multi-View Stereo (MVS) methods.

Chapter 3: Fundamentals of 3D Reconstruction: From Pixels to Geometry. This chapter introduces the mathematical and algorithmic foundations of 3D reconstruction, covering data representations (point clouds, meshes, Volumetric Pixel (VOXEL)s), camera projection, epipolar geometry, feature detection/description, and robust estimation. It concludes with Perspective-n-Point (PnP) and Bundle Adjustment (BA) as optimization methods that link image data to consistent 3D structure.

Chapter 4: Photogrammetry. This chapter outlines the classical photogrammetry pipeline: SfM, MVS, meshing, and texturing, as well as its practical realization in AliceVision/Meshroom, which is used in later experiments. The chapter also establishes the scope in relation to other tools. It summarizes limitations that commonly affect reconstruction quality.

Chapter 5: 3D Gaussian Splatting (3DGS). This chapter introduces 3DGS, beginning with its conceptual foundations, such as Novel View Synthesis (NVS), radiance fields, 3D Gaussians, and real-time rendering, and detailing its internal pipeline, including parametrization, adaptive density control, differentiable rasterization, and rendering. The chapter also introduces practical tools (*COLMAP* preprocessing, training code, and user-oriented platforms) and summarizes technical limitations relevant for later comparison.
Chapter 6: Method Selection and Evaluation Setup. This chapter defines the evaluation framework and introduces the tested tools, including mobile applications, desktop software, and the 3DGS source code implementation executed on Amazon Web Services (AWS). It outlines the dataset categories (*Everyday Scenarios* and *Edge Cases*), describes the experimental procedure, and specifies both qualitative (visual and feasibility) and quantitative (Peak Signal-to-Noise Ratio (PSNR), Structural Similarity Index (SSIM), Learned Perceptual Image Patch Similarity (LPIPS)) evaluation criteria used throughout the study.
Chapter 7: Results and Discussion. This chapter presents and interprets the experimental results, systematically comparing photogrammetry and 3D Gaussian Splatting across all dataset categories. The analysis follows a structured evaluation framework, reporting feasibility tests on Edge Cases and visual assessments of Everyday Scenarios for both methodologies. A comprehensive robustness analysis examines reconstruction stability across varying input sizes, followed by a cross-method comparison and an extensive tool evaluation covering performance, usability, and scalability. The chapter concludes with a critical discussion of methodological limitations and study constraints that contextualize the overall findings.
Chapter 8: Conclusion. This chapter concludes the thesis with a synthesis of the main findings, evaluating photogrammetry and 3D Gaussian Splatting under practical and hardware-limited conditions. It highlights that coherent 3D models can be created even by non-expert users using accessible mobile or cloud-based tools, and that image quality and recording design are decisive for reconstruction success. The chapter closes with perspectives for future research, addressing challenges such as merging independently captured segments and mitigating motion-related artifacts to improve robustness in real-world scenarios.

2 OMITTED 3D RECONSTRUCTION METHODS

The variety of available 3D reconstruction tools and pipelines is extensive. This thesis focuses on a selected set of methods implemented in both open-source and commercially available applications, provided that their cost remains within a reasonable range and they are suitable for comparative analysis within the study's scope.

2.1 HIGH-END NEURAL RENDERING METHODS

In recent years, significant advances have been made in neural rendering methods for 3D reconstruction. Methods like NeRF and its successors such as Multi-scale Neural Radiance Fields 360 (MipNeRF360), Instant Neural Graphics Primitives (InstantNGP), and Sparse Voxel-based Radiance Fields (Plenoxels) achieve impressive results but exhibit a fundamental trade-off between image quality and efficiency. MipNeRF360 can reach state-of-the-art fidelity, yet requires up to 48 hours of training time and several seconds per frame for rendering, making them impractical for interactive or everyday use [33]. Faster variants such as InstantNGP and Plenoxels provide real-time training or interactive frame rates, but typically at the expense of reduced visual fidelity [33].

3D Gaussian Splatting 3DGS overcomes this limitation by combining high-quality results with real-time rendering performance. In the original evaluation, fully converged models were obtained in 35-45 minutes on a single RTX A6000 GPU, while maintaining frame rates well above 30 Frames per Second (FPS) at 1080p resolution [33]. However, the method is not without drawbacks: its GPU memory footprint can exceed 20 GB for large-scale scenes in the current prototype implementation [33].

Another relevant approach is SMERF, which addresses the challenge of rendering for large outdoor or indoor scenes - up to $300m^2$. SMERF introduces a hierarchical scene partitioning strategy and distillation from high-quality teacher models (e.g. Compressed Neural Radiance Fields (ZipNeRF)), enabling real-time rendering even on commodity devices such as web browsers or smartphones. However, this runtime efficiency comes at the cost of extremely high training requirements. The reported models were trained for 100k-200k steps in clusters of 8× V100 or 16× A100 GPU [12]. Consequently, while SMERF is highly promising for large-scale scenarios, its training cost makes it unsuitable in the context of this thesis, which focuses on hardware-efficient and broadly accessible approaches to 3D reconstruction.

2.2 NEWER PHOTOGRAMMETRY METHODS

This thesis focuses on selected classical photogrammetry methods, namely those implemented in the open-source tools *COLMAP* [56] and *AliceVision/Meshroom*; hereafter *Meshroom* [3]. These approaches are based on SfM and MVS and are integrated into user-friendly pipelines, making them suitable for the experiments conducted in this work. Note that not all classical photogrammetry methods are covered here, but only those relevant to the chosen tools.

In addition, there are more recent and promising approaches, such as deep-learning-based methods, like End-to-End Deep Learning Architecture for Multi-View Stereo (MVSNet) [74] and Cascade Cost Volume Multi-View Stereo Network (CasMVSNet) [73]. Although these methods have demonstrated significant scientific progress, they are not discussed in this thesis because a detailed analysis would exceed the intended scope. Furthermore, they have not yet been included in the tested, user-friendly open-source pipelines.

It is important to note that photogrammetry has many applications, including geodesy, aerial mapping, architecture, cultural heritage, medicine, and forensics [54]. However, this thesis deliberately restricts its focus **on the 3D reconstruction of ordinary photographs**, as these can be taken by non-expert users with standard cameras or mobile devices. Specialized applications, such as industrial metrology, are therefore not part of the analysis.

3

FUNDAMENTALS OF 3D RECONSTRUCTION: FROM PIXELS TO GEOMETRY

This chapter introduces the mathematical and algorithmic foundations of 3D reconstruction. It explains the basic principles that connect 2D images to 3D geometry and prepares the ground for the methods presented in Chapters 4 and 5. It focuses on common representations, camera models, geometric relationships, and optimization techniques that are essential for both classical photogrammetry and modern approaches such as 3DGS.

3.1 FLAT VS. SPATIAL: UNDERSTANDING 3D DATA REPRESENTATIONS

Understanding the difference between 2D and 3D representations is the starting point for any reconstruction method. This section introduces the two most common data structures: point clouds, polygonal meshes and VOXEL grids. Each format has its own advantages and limitations. Together, they form the basis for storing and processing reconstructed geometry.

3.1.1 *2D vs. 3D Representation*

In order to reconstruct 3D models from images, it is necessary to understand the fundamental difference between 2D and 3D. A 2D representation, such as a photograph or drawing, projects a scene onto a flat plane using only width (x) and height (y). Although this captures the appearance of a scene, it lacks geometric depth, which prevents true spatial interaction [43].

In contrast, a 3D representation incorporates a depth dimension - the z-axis - enabling a complete spatial model that can be rotated, measured, and navigated. The 3D reconstruction process uses visual cues in 2D images, such as texture, shading, and perspective, to infer missing spatial information through geometric analysis [43]. This makes 3D reconstruction applicable to fields such as robotics, cultural heritage, and architecture, where an understanding of spatial properties is essential.

3.1.2 *Forms of representation in the 3D reconstruction*

The resulting 3D structure can be represented in various shapes, depending on the chosen reconstruction method. In traditional workflows, the three most commonly used formats are point clouds, polygonal meshes and VOXEL grids. These formats form the basis of many classical reconstruction systems and are introduced in the following sections.

3.1.2.1 *Point Cloud*

A point cloud is a set of discrete points in 3D space. Each point is defined by its spatial coordinates (x, y, z). A point cloud captures the geometry of an object or scene. This is done in an unstructured format. This means that it does not include explicit information about surfaces or connectivity.

In many cases, additional attributes such as color, intensity, or surface normals can be stored for each point. This makes point clouds flexible representations for both visualization and further processing.[77].

Point clouds are typically generated using image-based methods, such as SfM 4.1.1, or depth estimation techniques 4.1.2. Depending on the method used, the resulting clouds may be sparse or dense and their accuracy and resolution may vary.[56]

Although point clouds offer flexible geometric abstraction, they are not directly suitable for rendering or simulation. In traditional workflows, they are typically converted into polygonal meshes and textured to generate visually appealing and functional 3D models (see Sections 4.1.3) and 4.1.4).

3.1.2.2 *Polygonal Meshes*

A polygonal mesh is a surface representation created by connecting points (vertices) with edges to form polygons, which are typically triangles or quadrilaterals. Meshes are a compact, structured format that is widely used for rendering, simulation, and geometric analysis.

There are three main components of a mesh:

VERTICES: Points in 3D space that are usually defined by coordinates (x, y, z). They act as the structural nodes of the mesh.

EDGES: Line segments that connect pairs of vertices.

FACES: Enclosed polygons (usually triangles) formed by three or more edges.[72]

Generally, meshes are created from point clouds using a process called surface reconstruction. (see Section 4.1.3).

3.1.3 *Voxel*

A VOXEL is the smallest element of a regular 3D grid. Each VOXEL represents a specific volume in space. It can store attributes such as occupancy, density, or color. Similar to pixels in a 2D image, VOXELs divide space into discrete units, but in three dimensions.

VOXEL grids provide a simple and intuitive way to represent volumetric data. Their regular structure makes them compatible with 3D convolutional neural networks. They are also suitable for tasks such as scene representation, semantic labeling, or shape completion.

However, VOXEL representations consume a lot of memory, especially at high resolutions. According to Xiao et al. (2025), this cubic memory growth restricts their applicability to large-scale or high-detail scenes. Despite these

limitations, VOXEL-based approaches laid the foundation for many early volumetric neural rendering methods. [71]

3.2 DECODING THE BLACKBOX: THE MATH BEHIND 3D RECONSTRUCTION

Creating reliable 3D geometry from image data requires solid mathematical and algorithmic tools. This section introduces the key concepts that form the basis of many reconstruction pipeline. First, we explore the pinhole camera model, which illustrates how 3D points are projected onto a 2D image plane. Second, we examine epipolar geometry, which defines the geometric relationship between multiple views of the same scene. Following that, we introduce feature detection and description methods, such as Scale-Invariant Feature Transform (SIFT) and Accelerated KAZE Features (AKAZE), which enables robust matching across images under varying conditions. Next, Random Sample Consensus (RANSAC) is discussed as a robust estimation technique that removes outliers and ensures geometric consistency.

Finally, two key optimization methods are presented: PnP and BA. PnP reliably estimates camera poses from 2D-3D correspondences. BA refines poses and 3D structures by minimizing reprojection error across all images. Together, these methods provide the mathematical and algorithmic basis necessary for accurate, consistent 3D reconstruction. They also serve as the link between raw image data and the reconstruction pipelines that will be examined in the following chapters.

3.2.1 *The Pinhole Camera Model*

In order to understand how a 3D scene is reconstructed from 2D images, it is essential to understand how a camera projects 3D points from the world coordinate system onto the 2D image plane. The mathematical basis of this process is defined by the pinhole camera model, which is applied in both traditional methods like photogrammetry and modern techniques such as 3DGS.

In the pinhole model, a camera is abstracted as a sealed box with a small aperture, or pinhole, through which light travels in straight lines to form an image on the opposite plane. This produces a perspective projection, in which distant objects appear smaller. [48].

Figure 3.1 illustrates the geometric setup of perspective projection. A 3D point P projects through the center of the camera into the image plane. The following equations, (X, Y, Z) denotes the world coordinates of this point P, while (X_c, Y_c, Z_c) denote its coordinates in the camera frame as given by Equation (3.5).

PROJECTION EQUATION OF THE PINHOLE MODEL

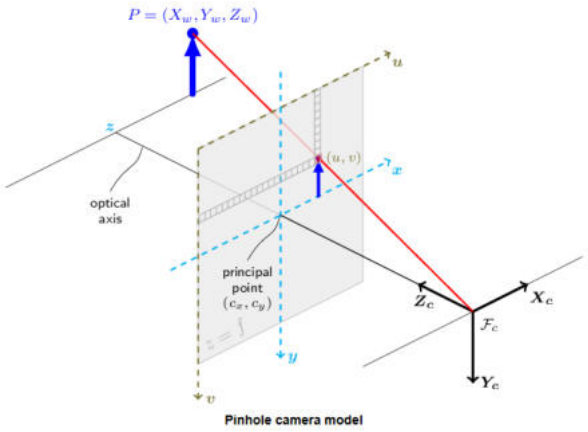


Figure 3.1: Pinhole camera model: a 3D point is projected onto the 2D image plane through the camera center. [48]

The projection from a 3D world point onto a 2D image point is written in homogeneous coordinates as:

$$w \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \mathbf{K} \begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}. \tag{3.1}$$

The result of this multiplication explicitly is:

$$\begin{bmatrix} u' \\ v' \\ w \end{bmatrix} = \mathbf{K} \begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}. \tag{3.2}$$

The final pixel coordinates (u, v) are obtained by normalization:

$$u = \frac{u'}{w}, \qquad v = \frac{v'}{w}. \tag{3.3}$$

Here, w is the homogeneous scale factor and corresponds to the depth Z_c of the 3D point in the camera coordinate system.[21, 48]

INTRINSIC PARAMETERS

The intrinsic matrix $\mathbf{K}$ encodes the internal characteristics of the camera:

$$\mathbf{K} = \begin{bmatrix} f_x & s_{\text{skew}} & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}. \tag{3.4}$$

- f_x, f_y: focal lengths along the x- and y-axes (in pixels),

- c_x, c_y: principal point (typically near the image center),
- s_{skew}: skew parameter, usually close to zero and generally insignificant [48].

EXTRINSIC PARAMETERS

The extrinsic parameters $(\mathbf{R}, \mathbf{t})$ define the camera's orientation and position relative to the world frame. A world point is first transformed into the camera frame:

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} = \mathbf{R} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} + \mathbf{t}. \tag{3.5}$$

Relative poses between multiple views are fundamental to 3D reconstruction methods such as SfM [21, 48].

HOMOGENEOUS COORDINATES AND NORMALIZATION

The use of homogeneous coordinates is essential to express the pinhole projection in a compact linear form. In this representation, a 2D point (x, y) is written as $(x : y : w)$, where all nonzero scalar multiples describe the same point. The actual Cartesian coordinates are obtained by normalization: $(x/w,\ y/w)$. This formulation makes perspective projection a linear operation and allows for the representation of points at infinity when $w = 0$ [21]. This normalization step corresponds to Equation (3.3) introduced in the pinhole projection.

CALIBRATED VS. UNCALIBRATED CAMERAS

A camera is calibrated if its parameters, particularly the intrinsic matrix $\mathbf{K}$ (see Equation (3.4)) and lens distortion, are known or estimated. In contrast, an uncalibrated camera lacks this information, so its intrinsic parameters are treated as unknown and must be inferred along with the scene geometry. In practice, these parameters are typically estimated directly from the image data during the reconstruction process. Despite such deviations from the ideal pinhole assumption, the model remains a robust approximation and the mathematical backbone of methods like SfM.

3.2.2 *Feature Detection and Description*

Feature detection and description are fundamental steps in many computer vision and photogrammetry pipelines, including image matching, SfM, and 3D reconstruction. The overall goal is twofold: First, identify salient and repeatable keypoints *feature detection*). Second, compute descriptors that capture the local image structure around these keypoints in a transformation-invariant manner (*feature description*). Effective features should be robust against geometric transformations such as scale changes, rotation, and mod-

erate viewpoint variation, as well as photometric changes such as illumination differences.

Over the years, numerous algorithms have been proposed and widely adopted in practice. Among the most prominent are the Scale-Invariant Feature Transform (SIFT) [41], Speeded-Up Robust Features (SURF) [7], Oriented FAST and Rotated BRIEF (ORB) [55], Binary Robust Invariant Scalable Keypoints (BRISK) [38], as well as the KAZE and AKAZE family of detectors[2]. Each method makes different trade-offs between robustness, invariance, computational cost, and memory usage. Classical algorithms such as SIFT and SURF prioritize robustness and accuracy, while methods such as ORB and BRISK emphasize efficiency and suitability for real-time or embedded systems.

In the following, two representative and widely used methods are discussed in more detail: the classical SIFT algorithm, which introduced the concept of a scale-invariant linear scale space, and the more recent AKAZE algorithm, which employs a nonlinear scale space with efficient binary descriptors.

3.2.2.1 *SIFT: Scale-Invariant Feature Transform*

Introduced by Lowe in 2004, SIFT is one of the most influential methods for detecting and describing distinctive local image features. SIFT descriptors are invariant to scale and rotation, and are robust against affine transformations, viewpoint changes, illumination changes, and image noise. [41]. Consequently, the method is highly effective for tasks such as object recognition, image stitching, and particularly in SfM pipelines. In SfM, SIFT provides the reliable feature correspondences that are essential for accurately estimating camera pose.

The SIFT algorithm follows a structured four-stage process:

1. **Scale-space extrema detection:** A Difference-of-Gaussian (Difference of Gaussians (DoG)) pyramid is constructed to detect potential keypoints invariant to scale. Keypoints are identified as local extrema in this 3D scale space.

2. **Keypoint refinement:** Candidate keypoints are adjusted to sub-pixel precision, and those with insufficient contrast or strong edge dominance are excluded to ensure stability.

3. **Orientation assignment:** Every keypoint is assigned a dominant orientation by analyzing the gradient directions in its local neighborhood. This crucial step gives the descriptor rotational invariance.

4. **Descriptor generation:** For every keypoint, a 128-dimensional vector of floating-point values is created to represent the distribution of gradient magnitudes and orientations within its surrounding region.

The resulting descriptors achieve excellent matching performance across a wide range of transformations, but their computation and storage are relatively expensive compared to more recent methods.

3.2.2.2 *AKAZE: Accelerated KAZE Features*

The AKAZE algorithm extends the earlier KAZE method by introducing a more computationally efficient nonlinear scale space. Unlike traditional linear scale spaces that rely on Gaussian smoothing, AKAZE builds its representation using edge-preserving Fast Explicit Diffusion (FED). This allows fine details and object boundaries to be preserved while reducing noise in homogeneous regions [2].

Key aspects of AKAZE include:

- **Nonlinear scale space:** Built using FED to adaptively smooth regions while retaining edge structures.
- **Feature detection:** Relies on the determinant of the Hessian, which exhibits strong responses at blob-like image regions.
- **Feature description:** Utilizes a Modified-Local Difference Binary (M-LDB) descriptor that combines intensity and gradient information into a compact binary format. This ensures invariance to scale and rotation.

The binary M-LDB descriptors can be matched efficiently using Hamming distance, which makes AKAZE particularly suitable for real-time or resource-constrained applications. Compared to SIFT, AKAZE typically offers better computational efficiency while still achieving competitive feature quality, especially in images with pronounced edges and fine structures. The output of both SIFT and AKAZE is a list of keypoints and their descriptors. These features are then matched across different images by identifying correspondences in a process called feature matching.

COMPARISON

SIFT and AKAZE represent different paradigms in feature detection and description. SIFT employs a linear Gaussian scale space and produces high-dimensional floating-point descriptors that are invariant with respect to scale and rotation, and are designed to be robust against affine transformations, lighting changes, and image noise.[41]. In contrast, AKAZE uses a nonlinear scale space based on FED and generates binary descriptors M-LDB that are highly efficient, scale- and rotation-invariant, and require less storage [2]. In comparative studies, SIFT is consistently recognized for its robustness and matching accuracy, while AKAZE provides a favorable trade-off by achieving competitive feature quality at significantly lower computational cost [2, 61].

3.2.2.3 *Feature Matching*

The outcome of feature detection and description is a collection of keypoints, each associated with a descriptor. The purpose of feature matching is to identify corresponding features across multiple images, forming the foundation for estimating epipolar geometry and performing 3D triangulation. The similarity between the descriptors is quantified using different distance metrics depending on their type. For floating-point descriptors like SIFT, the

Euclidean distance serves as the standard measure. In contrast, binary descriptors, such as those generated by AKAZE rely on the *Hamming distance*.

To obtain reliable correspondences despite noise, repetitive textures, or occlusions, several matching strategies are commonly applied:

- **Nearest-Neighbor Search (NN):** For each feature in the first image, the most similar feature in the second image is chosen based on the smallest descriptor distance.
- **Ratio Test (Lowe):** To eliminate uncertain correspondences, the algorithm compares the distance of the best match (d_1) with that of the second-best candidate (d_2). A match is accepted only if the ratio $\frac{d_1}{d_2} < \tau$, where τ is typically set to 0.75 [41].
- **Cross-Checking:** A match is kept only if it is consistent in both directions, i.e., feature x in image A matches feature x' in image B and vice versa. This helps eliminate asymmetric matches caused by noise or repetitive patterns.

The result of this step is a list of putative point correspondences $\{(x_i, x_i')\}_{i=1}^{N}$. Since this set still contains outliers, a subsequent geometric verification step is required. In SfM pipelines, robust estimation methods such as RANSAC are typically applied to simultaneously estimate a consistent epipolar geometry (e.g. the fundamental matrix) and reject outliers [13].

3.2.3 *Epipolar Geometry*

Epipolar geometry characterizes how two images of the same 3D scene, taken from distinct camera positions, are geometrically related. It is a central concept in projective geometry and forms the mathematical foundation of stereo vision (see Section 4.1.2) and SfM pipelines [21]. In such pipelines, local features (e.g. SIFT) are first detected and matched between images. Epipolar geometry is then applied to verify whether these correspondences are geometrically consistent.

When a 3D point $\mathbf{X}$ is projected into two images, it appears as $\mathbf{x}$ in the first image and $\mathbf{x}'$ in the second. From the perspective of the first camera, the image point $\mathbf{x}$ defines a ray in the 3D space that passes through the center of the camera C. Together with the center C' of the second camera, this ray forms the *epipolar plane*. The intersection of this plane with the second camera image plane defines the epipolar line $\mathbf{l}'$, where the corresponding image point $\mathbf{x}'$ must be located (see Fig. 3.2). The connecting line between the two camera centers is known as the *baseline*. The epipoles e and e are located where the baseline intersects the respective image planes. Furthermore, all epipolar lines within a single image converge at that image's epipole. As the 3D point $\mathbf{X}$ moves through space, the corresponding epipolar plane rotates around the baseline, forming a family of planes referred to as the *epipolar pencil*. (Fig. 3.3) [21].

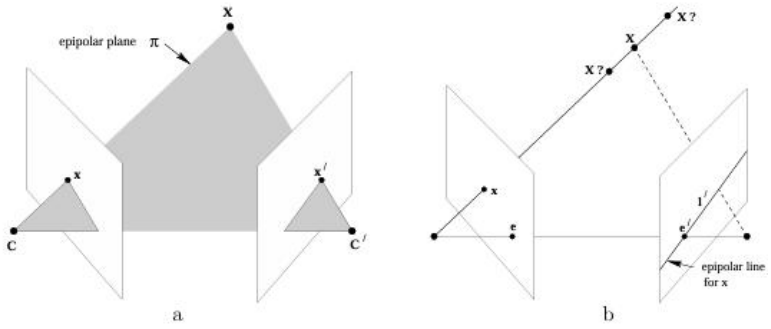


Figure 3.2: Illustration of point correspondence geometry between two camera views. A 3D point **X** is projected into the two image planes as **x** and **x**′. The camera centers C and C' and these projections together define the epipolar plane P. The viewing ray passing through C and **x** meets the second image plane along the epipolar line **l**′, which constrains the potential position of **x**′. [21]

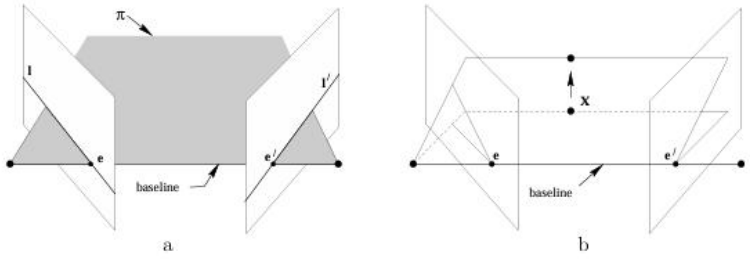


Figure 3.3: This is the epipolar geometry between two camera views. The baseline connecting the two camera centers intersects each image plane at its respective epipole (e and e). Any plane that contains this baseline is an *epipolar plane*. These planes intersect the image planes at corresponding *epipolar lines*, **l** and **l**′. As the 3D point **X** moves, its associated epipolar plane swivels around the baseline, creating the *epipolar pencil*, which is a family of planes. [21]

Based on this geometry, each image point in the first view corresponds to a line in the second image where its matching point must lie. This geometric restriction is expressed by the *fundamental matrix* $\mathbf{F}$, a 3×3 matrix of rank 2. For each pair of corresponding image points $\mathbf{x} \leftrightarrow \mathbf{x}'$, the *epipolar constraint* holds:

$$\mathbf{x}'^{\top} \mathbf{F} \mathbf{x} = 0. \tag{3.6}$$

In practice, robust estimation methods such as RANSAC are employed to compute the fundamental matrix from noisy point correspondences (see Section 3.2.4).

For calibrated camera systems, the epipolar relationship is represented by the *essential matrix* $\mathbf{E}$. This matrix operates on normalized image coordinates and encapsulates the relative orientation and position of the two camera views:

$$\hat{\mathbf{x}}'^{\top} \mathbf{E} \hat{\mathbf{x}} = 0. \tag{3.7}$$

The fundamental matrix $\mathbf{F}$ can be converted into the essential matrix $\mathbf{E}$ using the intrinsic calibration parameters of both cameras:

$$\mathbf{E} = \mathbf{K}'^{\top} \mathbf{F} \mathbf{K}. \tag{3.8}$$

While $\mathbf{F}$ suffices for projective reconstruction, $\mathbf{E}$ enables metric reconstruction and pose estimation between calibrated views [21].

Modern SfM systems, such as *COLMAP* [56] or *Meshroom*[19], make extensive use of epipolar geometry to validate feature correspondences. The advantage is practical: instead of searching for a match across the entire image, the search is restricted to the corresponding epipolar line. This reduces computational effort and improves robustness in large-scale or resource-constrained scenarios [21, 60]. Thus, epipolar geometry serves as the bridge between feature matching and robust 3D reconstruction.

3.2.4 *RANSAC (Random Sample Consensus)*

Even robust feature matching methods usually produce some incorrect correspondences (outliers). These outliers can strongly distort the estimation of geometric models such as epipolar geometry or camera poses. RANSAC (Random Sample Consensus) [13] is a robust estimation method that can successfully compute models even when there are many outliers.

Basic idea: RANSAC follows a hypothesis-and-test strategy. Instead of estimating a model from all data points, it repeatedly selects small random subsets (the *minimum sample set*) and computes a candidate model. Each hypothesis is then tested against the full dataset.

Algorithm steps:

1. *Minimal sample:* Randomly choose the smallest subset of point correspondences required to estimate the model (e.g., seven correspon-

dences are needed to compute the fundamental matrix **F**, see Equation (3.6) [21]).

2. *Model estimation:* Compute a candidate model from this subset.
3. *Evaluation:* Count how many other points (the *inliers*) are consistent with the model, that is, within a defined error threshold.
4. *Iteration:* Repeat the steps above. The number of iterations is adapted based on the inlier ratio of the best model found so far to ensure a high probability of finding the correct solution [13].
5. *Selection:* The final model is determined by selecting the candidate with the most inliers identified during the evaluation phase.
6. *Refinement (optional):* Re-estimate the final model using all inliers for higher accuracy.

Applications in 3D reconstruction: RANSAC is a key component in modern SfM systems such as *COLMAP* [56]. Typical use cases include:

- Estimation of the fundamental matrix **F** or essential matrix **E** from 2D–2D correspondences (see Equations (3.6)–(3.8)),
- Estimation of camera poses with PnP algorithms from 2D–3D correspondences (see Section 3.2.6),
- Estimation of homographies for planar scenes.

3.2.5 *Triangulation - Estimating 3D Points from Image Pairs*

Triangulation refers to the geometric procedure used to reconstruct the spatial position of a 3D point based on its projections observed in two or more camera views. It builds directly on the epipolar geometry (see Section 3.2.3), which constrains the possible location of corresponding points to epipolar lines. Once consistent correspondences have been identified, triangulation computes their 3D position in space. The method requires known or previously estimated intrinsic and extrinsic camera parameters, represented by the projection matrices P and P' (see Section 3.2.1). As a core step in 3D reconstruction pipelines, triangulation provides the first 3D structure (sparse point cloud) from 2D correspondences.

GEOMETRIC IDEA

Let $\mathbf{x} \leftrightarrow \mathbf{x}'$ be a pair of corresponding image points. The projection equations

$$\mathbf{x} = P\mathbf{X}, \quad \mathbf{x}' = P'\mathbf{X} \tag{3.9}$$

define two rays in space that pass through the respective camera centers and image points. In the absence of noise, these rays intersect at the true

3D point **X**. In practice, due to noise and calibration errors, they do not intersect exactly. Triangulation then estimates the 3D point that minimizes the distance to both rays (Fig. 3.4), based on the projection equations in Equation (3.9) [21].

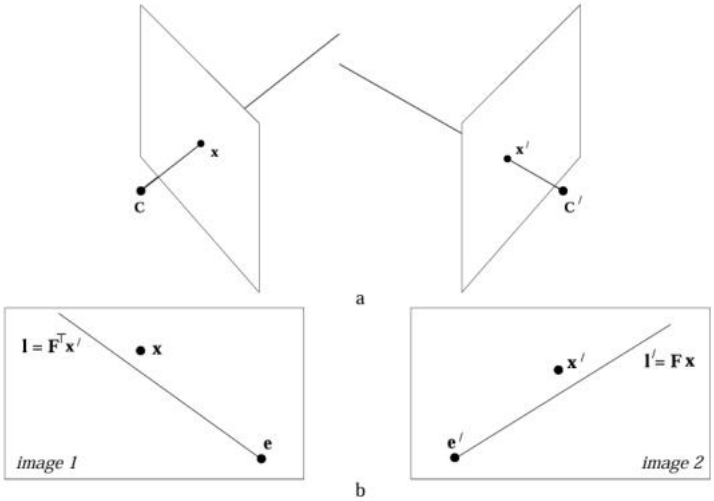


Figure 3.4: Each image point defines a ray in 3D space. In the ideal case, the rays intersect at the true 3D point **X**. With noise, the estimated point is chosen closest to both rays. [21]

REPROJECTION ERROR

The quality of a triangulated point is measured by its reprojection error, i.e., the distance between the original image points and the projections of the estimated 3D point:

$$\|\mathbf{x} - \hat{\mathbf{x}}\|^2 + \|\mathbf{x}' - \hat{\mathbf{x}'}\|^2. \tag{3.10}$$

A small error according to Equation (3.10) indicates a high consistency with the observations (Fig. 3.5). [21]

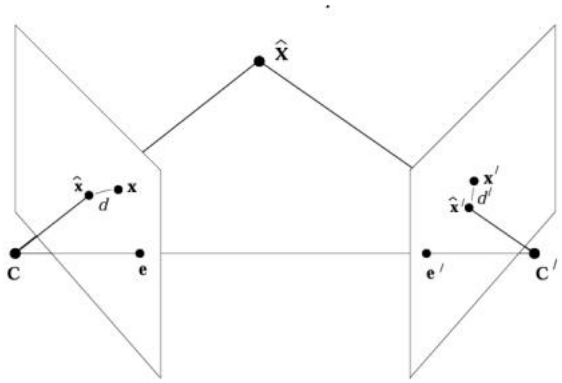


Figure 3.5: The 3D point **X** is optimized such that its projections lie as close as possible to the observed image points **x** and **x**′. [21]

METHODS

A common approach is *linear triangulation*, where the problem is formulated as a homogeneous linear system.

$$A\mathbf{X} = 0 \tag{3.11}$$

and solved via Singular Value Decomposition (SVD). This formulation in Equation (3.11) is efficient, but can be inaccurate in degenerate configurations (e.g. narrow baselines). More accurate results are obtained with *nonlinear triangulation*, which minimizes the reprojection error from Equation (3.10), typically through iterative optimization [21].

ROLE IN SFM

In SfM systems, triangulation provides the initial 3D points that form a sparse point cloud. These points are then used to estimate additional camera poses and are refined together with camera parameters in the subsequent BA step (see Section 3.2.7). Robust pipelines combine triangulation with RANSAC to remove outlier correspondences. The precision of triangulation directly impacts the quality of the final reconstruction.[21].

3.2.6 *Perspective-n-Point (PnP)*

After an initial set of 3D points has been reconstructed using triangulation, the next step is to determine the poses of additional cameras. This is formulated as the *Perspective-n-Point* (PnP) problem. PnP is essential in SfM pipelines, as it allows the incremental expansion of reconstruction by localizing new cameras relative to already known 3D points.

PROBLEM DEFINITION

The PnP problem is defined as the task of estimating the position and orientation of a calibrated camera from a set of known 3D points and their 2D image projections [37]. Formally, given:

- A calibrated camera with known intrinsic matrix $\mathbf{K} \in \mathbb{R}^{3\times3}$,
- A set of $n \geq 4$ reference points with known 3D coordinates $\{\mathbf{X}_i\}_{i=1}^{n}$ in a world reference frame,
- Their corresponding 2D projections $\{\mathbf{x}_i\}_{i=1}^{n}$ in the image plane.

Each 3D–2D correspondence is related by the perspective projection equation

$$\lambda_i \begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix} = \mathbf{K} \begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix} \begin{bmatrix} X_i \\ Y_i \\ Z_i \\ 1 \end{bmatrix}, \tag{3.12}$$

where $\lambda_i \in \mathbb{R}^+$ is a projective depth factor, and $\mathbf{X}_i$ is expressed in homogeneous coordinates. The rotation matrix $\mathbf{R}$ is orthonormal ($\mathbf{R}^\top \mathbf{R} = \mathbf{I}_3$) and orientation-preserving ($\det(\mathbf{R}) = +1$), making it a member of the *special orthogonal group* $\mathrm{SO}(3)$. The vector $\mathbf{t} \in \mathbb{R}^3$ denotes the translation.

The objective of the PnP problem is therefore to estimate the camera pose parameters $[\mathbf{R} \mid \mathbf{t}]$ that best satisfy Equation (3.12) for all n correspondences, following the definition and notation introduced by Lepetit, Moreno-Noguer, and Fua [37].

CLASSICAL METHODS

P3P (Perspective-Three-Point). The camera pose can be estimated from exactly three correspondences, yielding up to four possible solutions. A fourth point is required to disambiguate the correct one [17].

DLT (Direct Linear Transform). For $n \geq 6$, the pose can be estimated linearly via a homogeneous equation system solved with SVD (Singular Value Decomposition)[21]. This method is efficient, but sensitive to noise and unstable when the 3D points are nearly coplanar, as this configuration leads to an ambiguous solution for the translation component.

Iterative methods. Nonlinear optimization methods (e.g., Gauss-Newton or Levenberg-Marquardt) minimize the reprojection error (see Equation (3.10)) [21, 42]. They are accurate but slower and require good initialization.

MODERN METHOD: EPNP

The Efficient Perspective-n-Point (EPnP) algorithm provides a closed-form solution with linear complexity. Each 3D point is represented as a linear combination of four predefined control points:

$$\mathbf{X}_i = \sum_{j=1}^{4} \alpha_{ij} \mathbf{c}_j \quad \text{with} \quad \sum_{j=1}^{4} \alpha_{ij} = 1. \tag{3.13}$$

This reduces the number of unknowns, allowing the pose to be estimated via a linear system followed by solving a small quadratic set of constraints. An optional Gauss–Newton refinement step improves precision with minimal computational overhead. EPnP is robust to noise and performs well even in nearly planar configurations.[37]

ROBUST ESTIMATION USING RANSAC

In practice, 2D - 3D correspondences often contain outliers. To handle this, PnP is commonly combined with RANSAC. The algorithm repeatedly selects small random subsets, estimates a candidate pose, and counts how many correspondences agree with it. The pose with the most inliers is chosen, followed by a refinement step using all inliers.[13, 37]

POSE REFINEMENT IN THE PIPELINE

The pose estimated by PnP serves as an initial guess for BA (see Section 3.2.7). In this global optimization, all camera poses and 3D points are

refined jointly by minimizing the total reprojection error, which substantially enhances the overall precision and coherence of the reconstructed scene.

3.2.7 *Bundle Adjustment (BA)*

BA is the final optimization step in a SfM pipeline. In earlier steps – Triangulation (Section 3.2.5) and PnP (Section Section 3.2.6) – camera poses and 3D points are estimated **locally**. BA takes all cameras and all 3D points and optimizes them jointly. This makes it possible to remove the accumulation of drift and systematic errors, providing the most accurate and consistent reconstruction of the entire scene [62]. Reprojection error is widely used and preferred because it evaluates predictions directly against the measured image points.[40]

MATHEMATICAL IDEA

BA minimizes the *global* reprojection error over all camera and point parameters. A common multi-view formulation is

$$\min_{\{\mathbf{y}_i\},\{\mathbf{z}_j\}} \sum_{i,j} v_{ij}\, \rho\Big(\big\|\mathbf{x}_{ij} - \pi(\mathbf{y}_i, \mathbf{z}_j)\big\|^2 \Big), \tag{3.14}$$

where:

- $\mathbf{x}_{ij}$ is the measured image point of 3D point j in camera i,
- $\pi(\cdot)$ is the camera projection function (see Section 3.2.1 and Equation (3.1)),
- $v_{ij} \in \{0, 1\}$ denotes visibility,
- $\rho(\cdot)$ is a robust loss (Huber norm).

A common convention is to fix one reference camera to define the world frame[40].

HOW IT WORKS

This is a nonlinear least-squares problem. In practice, one linearizes per iteration and solves a Gauss–Newton system

$$(\mathbf{J}^\top \mathbf{J})\, \delta \;=\; -\mathbf{J}^\top \mathbf{r}, \tag{3.15}$$

where $\mathbf{J}$ is the Jacobian w.r.t. all camera and point parameters and $\mathbf{r}$ stacks the reprojection residuals [40]. The Jacobian in BA has a block structure (camera vs. point parameters), enabling the classic Schur complement reduction: eliminate point blocks, solve the reduced camera system, then recover points by back-substitution [62]. For efficiency in large-scale settings, practical systems use windowed BA and motion-only BA, and BA can also be viewed as a graph optimization problem (nodes = parameters, edges = measurements) [40].

LOCAL VS. GLOBAL BUNDLE ADJUSTMENT

Two main variants of BA are used in SfM pipelines [59]:

- **Local BA**: Only a subset of the model is optimized, typically the most recently added camera and a limited set of nearby points and cameras. This is common in incremental SfM after registering a new image. It is faster but ensures only local consistency.
- **Global BA**: All cameras and all 3D points are optimized together. This is usually performed at the end of the reconstruction to achieve the highest possible accuracy, but it is more computationally expensive.

SCALING TO LARGE SCENES

For very large datasets, classical BA quickly becomes too slow and memory intensive. Modern approaches therefore apply more efficient optimization strategies, such as iterative solvers and preconditioning techniques, to handle large-scale problems [1].

4

PHOTOGRAMMETRY

Photogrammetry is one of the most established approaches to 3D reconstruction and has been applied for decades in fields such as surveying, architecture, and cultural heritage. Unlike modern neural methods, it is based on classical geometry and image processing, which makes its results metrically accurate and interpretable, as it relies on physically explainable geometric principles rather than learned priors. In the context of this thesis, photogrammetry serves as a methodological baseline: it provides a transparent, reproducible pipeline whose strengths and weaknesses can be directly compared to newer approaches such as 3D Gaussian Splatting (Chapter 5).

The following sections first outline the typical photogrammetry pipeline (Section 4.1), before highlighting common limitations that strongly influence reconstruction quality in practice (Section 4.3).

4.1 PHOTOGRAMMETRY-PIPELINE

The classical photogrammetry pipeline consists of four stages: SfM for camera poses and a sparse 3D structure, MVS for dense geometry, followed by *meshing* and *texturing*. Meshing converts the dense point cloud into a continuous polygonal surface, and texturing projects the input photographs onto this surface via UV parameterization to obtain a photorealistic appearance [19, 39].

SCOPE AND TOOLING.

Each stage is a substantial research area in its own right. In the following, we provide a compact overview that emphasizes classical, geometry-driven methods and, where appropriate, mentions more recent relevant approaches. We describe the concrete realization with *Meshroom*, which we used in our experiments. Other photogrammetry applications tested in this work (e.g., Kiri Engine, Polycam) do not publicly disclose their internal algorithms; therefore, they are not discussed in detail here [19].

4.1.1 *Structure-from-Motion (SfM)*

Structure-from-Motion SfM describes the simultaneous estimation of camera parameters (intrinsic and extrinsic camera parameters) (see Section 3.2.1) and a 3D scene structure from 2D images. The goal is to create a geometrically consistent reconstruction from unordered photo collections. SfM approaches are commonly divided into three categories: **incremental**, **global**, and **hybrid** [25, 56, 75]. This thesis focuses on incremental SfM, the foundation of widely used tools like *COLMAP* and *Meshroom*, due to its robustness

and scalability. Practical scalability is achieved through efficient BA implementations (see Section 3.2.7).

One of the first influential works was the "Photo Tourism" system by Snavely et al. [58]. It introduced a clear SfM pipeline: First, local features such as **SIFT** are detected in the images (see Section 3.2.2). These features are matched across the images (Section 3.2.2.3), and incorrect correspondences are filtered using **epipolar geometry** (Section 3.2.3). The reconstruction starts with a stable image pair, from which initial 3D points are obtained by **triangulation** (Section 3.2.5). The initial pair is chosen to be geometrically strong, with many inliers after robust estimation and a sufficiently wide baseline/parallax to support stable triangulation. New cameras are added with the PnP algorithm (Section 3.2.6). After each step, a **local BA** (Section 3.2.7) is applied to refine the solution. A global BA is run periodically or at the end to reduce drift and improve overall consistency (Section 3.2.7). Certain configurations reduce stability: near-planar scenes, very small baselines, or repeated patterns can lead to weak constraints and higher uncertainty in triangulation (see Section 3.2.3 and 3.2.5). This iterative process produces a gradually growing sparse 3D reconstruction: a set of reliable 3D points that reflect the skeletal geometry of the scene without capturing fine surface details (see Figure 4.1).

Schönberger et al. [56] proposed systematic improvements, introducing more robust feature matching, stable initialization strategies, and efficient camera registration with PnP and RANSAC. Their work also improved the scalability of BA, making it possible to process very large datasets. These improvements form the basis of *COLMAP* , which today is considered the standard implementation of incremental SfM.

The output of SfM is a **sparse 3D reconstruction** together with estimated camera poses. This output provides the geometric framework for the entire photogrammetry pipeline. It serves as input for the next step, **MVS** (Section 4.1.2), which densifies the reconstruction. Note that SfM recovers structure and motion only up to an unknown global scale. An absolute scale can be fixed by adding a metric constraint (e.g., a known distance or external sensor information).

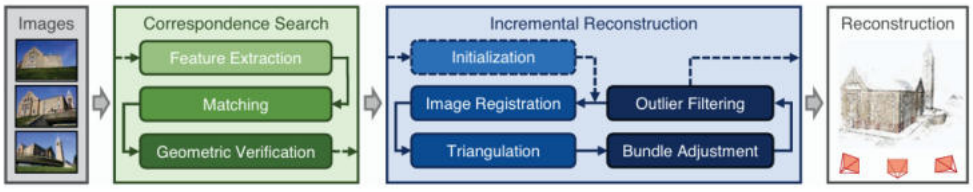


Figure 4.1: Incremental Structure-from-Motion pipeline. [56].

4.1.2 *Multi-View Stereo (MVS)*

Multi-View Stereo MVS takes the camera poses and sparse 3D points estimated by SfM as input and densifies the reconstruction. The goal of MVS is

to compute accurate depth values for a large number of pixels across multiple images, leading to a dense 3D point cloud of the scene. In contrast to SfM, which provides only a sparse reconstruction, MVS exploits redundant information from many overlapping views to achieve high-resolution geometry. This dense point cloud then serves as the foundation for meshing and texturing in the subsequent steps of the photogrammetry pipeline. In offline pipelines, the sparse SfM points also help restrict plausible depth ranges and guide view selection for robust estimation (see Section 4.1.1).

Principle: rooted in classical stereo matching. Stereo matching considers two images taken from slightly different viewpoints. For a pixel in the first image, the corresponding pixel in the second image is searched. The displacement between these pixels is called disparity. Using the camera geometry, the depth can then be computed as:

$$z = \frac{fb}{d}$$

where f denotes the focal length, b represents the baseline, which measures the distance between the two cameras, and d corresponds to the observed disparity between matching pixels.

EXAMPLE (USED HERE): SEMI-GLOBAL MATCHING (SGM).

Semi-Global Matching (SGM) enforces a near-global smoothness prior by aggregating dynamic-programming costs along multiple 1D paths through the image, rather than solving a full 2D optimization. First, a pixel-wise matching cost is computed using *Mutual Information* (MI), which is robust to radiometric differences between images. The costs are then aggregated along multiple directions by accumulating minimal path costs for each pixel and disparity. Small changes in disparity are penalized with a lower term P_1, while larger jumps receive a higher penalty P_2, which encourages smooth surfaces while preserving discontinuities [22]. A winner-takes-all selection over the aggregated costs yields the disparity for each pixel, which can be refined to sub-pixel accuracy. SGM runs with linear complexity $O(W \cdot H \cdot D)$ and achieves a good trade-off between computational cost and reconstruction accuracy. In this thesis, per-view depth maps are computed with *Meshroom*, whose depth-map stage employs SGM with a ZNCC-based multi-view photo-consistency measure and CUDA acceleration [19, 22].

Practical issues and multi-view extension. In practice, stereo matching faces several challenges. Typical failure cases are occlusions, untextured regions, slanted surfaces, and radiometric differences such as changes in illumination or exposure. These issues often lead to errors in disparity estimation. MVS extends this concept from two views to many overlapping images precisely to mitigate these issues: redundancy increases robustness and enables more detailed reconstructions. From the sparse point cloud provided by SfM, MVS builds dense depth maps for each image and fuses them into a consistent 3D representation.

Algorithmic approaches for depth-map generation:

- **Voxel-based methods:** Divide space into voxels and test occupancy. They are robust but memory intensive.
- **Depth-map fusion:** Compute a per-view depth map, then fuse them into a consistent 3D model. This is efficient and widely used. In this thesis we follow this approach with *Meshroom* (SGM-based depth estimation) [19, 22].
- **Patch-based methods (background only):** Estimate small surface patches and refine them iteratively (e.g., PatchMatch Stereo); not part of our pipeline, but widely used in the literature [6, 8].

LEARNING-BASED MVS (BACKGROUND ONLY; NOT EVALUATED).

Recent surveys describe learning-based MVS as building plane-sweep cost volumes and learning their regularization (e.g., with 3D convolutions or coarse-to-fine schemes) before regressing a per-view depth map, followed by offline fusion; online (video) variants often rely on TSDF volumes as an intermediate representation. These models can achieve strong benchmark results but typically require substantial compute and specialized frameworks. We acknowledge this line of work for completeness and keep our focus on the classical, geometry-driven depth-map MVS implemented in *Meshroom* [64]. A representative early end-to-end model is *MVSNet*, which learns to construct and regularize cost volumes directly from multiple views [74].

4.1.3 *Meshing*

After generating a dense point cloud, the next step is to reconstruct a continuous surface ("meshing"). The goal is to transform an unstructured set of 3D points into a polygonal mesh that approximates the underlying geometry. A mesh is typically represented as connected triangles that form a coherent surface. This step converts discrete points into a usable 3D model that can be visualized, edited, or used in downstream applications.

Delaunay-based surface reconstruction. A common approach to surface reconstruction is based on Delaunay triangulation, which partitions space into tetrahedra and allows surfaces to be extracted from their connectivity. Delaunay methods first compute the 3D Delaunay triangulation $D(P)$ from the given set of sample points P, which is the dual of the Voronoi diagram. A surface is then extracted by selecting an appropriate subcomplex of $D(P)$. Two classical routes are: (i) *restriction* of $D(P)$ to a subset that approximates the (unknown) surface. For example, restricted Delaunay/α-complexes, and (ii) *inside/outside labeling* of tetrahedra. The interface between the labeled cells yields the surface. Figure 4.2 illustrates the idea of restricting a Delaunay triangulation to a surface. Under suitable conditions (e.g., closed-ball property), the restricted Delaunay triangulation $D_S(P)$ is topologically consistent with the underlying surface [9].

MESHING IN ALICEVISION/MESHROOM.

In the workflow used here, meshing follows AliceVision's default pipeline. After SfM, the per-view depth maps are back-projected and fused into a dense point cloud using a KD-tree structure. From this point cloud, a 3D Delaunay tetrahedralization is constructed. Per-tetrahedron weights are then computed according to Jančošek et al., and a graph-cut optimization labels tetrahedra as inside or outside to extract the final surface. Finally, local artifacts are filtered out and bilateral smoothing is applied to improve surface quality [19]. This approach implements the *inside/outside labeling* paradigm (via voting and graph-cut) rather than the *restricted-Delaunay* subcomplex extraction illustrated in Fig. 4.2.

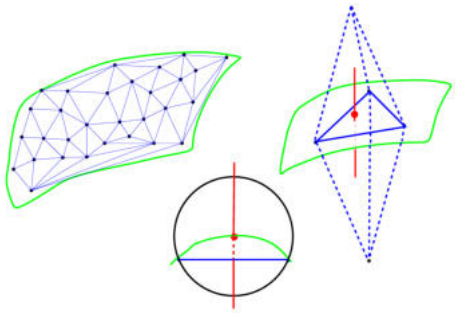


Figure 4.2: Delaunay-based meshing via restriction: a 3D Delaunay tetrahedralization is built on the samples, and a surface is extracted by restricting to the vicinity of the underlying surface. [9])

POISSON RECONSTRUCTION (ANOTHER RELEVANT METHOD).

Poisson Surface Reconstruction (PSR) seeks an implicit indicator function χ whose gradient matches the oriented normal field by solving a Poisson equation $\Delta\chi = \nabla \cdot \vec{V}$. An iso-surface is then extracted from this function. PSR produces watertight surfaces and is robust to noise. However, it may smooth out fine details, and the screened variant mitigates this effect [23].

OTHER RELEVANT VARIANTS.

Beyond Delaunay and Poisson, several other approaches to surface reconstruction have been proposed. Triangulation-based methods include Ball-Pivoting and greedy or Delaunay-based extractions. Point-set surface methods rely on moving least squares, such as APSS or RIMLS. Implicit representations can be obtained with radial basis functions, while primitive or template fitting methods approximate geometry with simple shapes such as planes or cylinders. More recently, learning-based techniques employ neural implicit fields to reconstruct surfaces [23].

4.1.4 *Texturing*

Texturing is the final step of the photogrammetry pipeline. While the previous stages reconstruct the geometry of the object as a mesh, texturing gives

this mesh a realistic appearance. The input photos are projected onto the surface, so that colors, patterns, and fine details become visible [19, 39].

The basic method is called *UV mapping*. The 3D mesh is unfolded into a 2D space. Each vertex of the mesh receives coordinates (u, v). These coordinates link the mesh to a texture image, which can then be projected back onto the surface. In practice, the mesh is divided into parts called *charts* (UV patches). Each chart is mapped separately into the UV space. Figure 4.3 illustrates this process: on the left, the 3D mesh with chart boundaries is shown, in the center the charts are visible as separate UV patches arranged in the 2D texture atlas, and on the right the final textured model.

Figure 4.3: The texturing process: the mesh with chart boundaries (left), the charts as separate UV patches in the 2D texture atlas (center), and the final textured model (right). [39]

Texturing introduces several challenges. Visible seams may appear at the boundaries of charts, especially when input photos differ in lighting or color values. Overlapping coverage, where multiple photos capture the same region, can lead to inconsistencies such as color shifts, blurring, or ghosting if not combined consistently. In addition, distortions arise when the unfolding into the UV plane is not uniform, causing textures to appear stretched or compressed [39].

4.1.4.1 *Least Squares Conformal Maps (LSCM)*

A common method to reduce distortion is the *Least Squares Conformal Maps (LSCM)* algorithm introduced by Lévy et al. [39]. It creates an almost angle-preserving unfolding of the mesh by approximating conformal maps. LSCM is efficient, robust against triangle flips, and requires only minimal constraints. Today, it is considered a classical algorithm for UV parametrization and is implemented in tools such as *Meshroom* [19].

4.1.4.2 *Advanced and semantic texturing approaches*

Beyond classical projection-based methods, recent research explores semantic guidance and learning-based UV mappings to improve texture consistency, alignment, and transfer (see Section 4.1.4.1).

Semantic UV mapping for indoor texture inpainting. Recent work leverages semantic instance segmentation of indoor scenes to guide UV unwrapping and improve texture inpainting after clutter removal [63]. The pipeline segments structural elements (e.g., walls, floor), removes loose objects, reconstructs missing geometry using plane-intersection boundaries, and then aligns UV seams with semantic boundaries. Charts are oriented consistently and unwrapped with low distortion so that adjacent faces in 3D remain adjacent in UV; missing regions are inpainted per element in 2D and reprojected to the mesh, followed by UV repacking for better texture-space usage [63].

Learning aligned UV maps for texture transfer (AUV-Net). AUV-Net learns to embed 3D surfaces into a shared, *aligned* UV space to enable texture transfer, texture synthesis, and textured single-view reconstruction [10]. The model jointly learns (i) a set of texture *basis images* with per-shape coefficients (alignment module), (ii) a shared UV mapper that assigns surface points to UV coordinates, and (iii) a masking network that predicts a *soft segmentation* of the surface into a small number of charts (e.g., "front/back") to reduce distortion. The aligned UV images allow the use of standard 2D generative models (e.g. StyleGAN2) for texture synthesis, supporting tasks such as texture transfer and texture completion, across categories such as cars, chairs, heads, and animals [10].

4.2 TOOLS AND PLATFORMS FOR PHOTOGRAMMETRY

Unlike the still emerging ecosystem around 3DGS (see Chapter 5), photogrammetry has been established for many years and offers a wide spectrum of mature tools. These range from open-source frameworks used in research to professional software suites and, more recently, simplified mobile applications. The diversity of tools reflects the long tradition of photogrammetry as well as its broad adoption across disciplines such as surveying, cultural heritage, and visual effects.

4.2.1 *Open-source software as a baseline*

Frameworks like *COLMAP* and *Meshroom* have become standard tools in academic and experimental settings. They implement the classical pipeline of SfM and MVS, offering transparency, reproducibility, and access to intermediate results. This makes them suitable as a methodological baseline in this thesis. However, they require technical expertise and powerful desktop hardware.

4.2.2 *Commercial software as high-end solutions*

Professional packages such as *RealityCapture*, *Agisoft Metashape*, and *Pix4D* provide optimized pipelines that deliver high accuracy and efficiency. These tools are widely used in industry and cultural heritage documentation, but they are license-based and therefore outside the scope of this work.

4.2.3 *Mobile applications as a recent trend*

In recent years, mobile apps have extended photogrammetry to a broader audience. They hide most of the complexity by outsourcing the pipeline to cloud services. Although this makes the technology highly accessible, it limits control and reproducibility, since intermediate data such as camera poses or sparse reconstructions are not exposed. Moreover, the exact photogrammetry methods implemented in these services are not documented, which prevents a precise methodological comparison. For this reason, such apps are considered in the evaluation mainly from the perspective of accessibility and usability rather than metric accuracy.

In summary, photogrammetry offers a mature and diverse ecosystem of tools. For this thesis, open-source frameworks are used as a transparent baseline, while mobile applications are included to reflect their growing role in making 3D reconstruction accessible to nonexpert users. Detailed analyses of the selected tools are presented in Chapter 7.

4.3 PHOTOGRAMMETRY LIMITATIONS

Photogrammetry performance strongly depends on image data, scene properties, and algorithmic choices. Typical limitations can be grouped into three categories: image- and scene-dependent issues, algorithmic issues, and technical constraints.

Image- and scene-dependent issues

- **Low texture, repetitive patterns, specular/transparent materials.** Feature matching and photoconsistency become unreliable, causing holes, noise or floating points in MVS depth maps [22, 64].
- **Insufficient parallax and degenerate geometry.** Small or uneven baselines, forward-only motion or near-planar scenes make triangulation ill-conditioned. The depth accuracy degrades with distance (small disparities) and weak baselines (see Section 4.1.2) [57].
- **Dynamic content and illumination changes.** Moving or non-rigid objects, motion blur, and rolling-shutter distortions cause mismatches in SfM/MVS. Exposure shifts, white balance differences, and shadows often produce ghosting or color seams in the texture [64].
- **Occlusions and coverage gaps.** Self-occlusions and missing viewpoints reduce overlap and lead to incomplete reconstructions, particularly for thin structures and concavities [57].

Algorithmic issues

- **MVS failure modes.** Even with good poses, multiview depth estimation can fail in textureless, reflective, or slanted/low-baseline regions. Fusion must reject outliers and may leave gaps [22, 64].
- **Meshing trade-offs.** Delaunay-based extraction preserves detail, but may yield nonwatertight surfaces and require cleanup. Poisson or screened Poisson produces watertight results but can over-smooth sharp features [9, 23].
- **Texturing artifacts.** UV unwrapping introduces distortion. Seams become visible under radiometric changes, and multi-view blending can cause blurring or ghosting under parallax. Parameterizations such as LSCM and careful view selection help, but cannot remove all artifacts [19, 39].

Technical constraints

- **Calibration and scale.** Inaccurate intrinsics (e.g., distortion) or extrinsics, and missing control points affect metric accuracy and absolute scale. Passive imaging cannot directly observe the scale without external cues [54].
- **Compute and memory.** Large image sets and high-resolution depth maps are resource intensive. GPU acceleration is commonly used for the depth map stage and texture generation [19].

Mitigation (brief). Good acquisition protocols (sufficient overlap and parallax, exposure locking, sharp images), reliable calibration and scale references, and adapted method / tool choices (e.g. mesh variant, texturing strategy) can substantially reduce these issues, but cannot fully remove the inherent limitations of passive image-based reconstruction.

5

3D GAUSSIAN SPLATTING (3DGS)

5.1 COMPARISON WITH NERF AND PHOTOGRAMMETRY

Before introducing the internal concepts of 3D Gaussian Splatting (3DGS), it is useful to position the method in relation to two established approaches: classical photogrammetry and NeRF.

Photogrammetry, as described in Chapter 4, reconstructs explicit geometry by estimating camera poses, generating point clouds, and subsequently creating meshes and textures [54, 56]. This pipeline yields metrically accurate models and has a long tradition in fields such as surveying and cultural heritage documentation. However, it strongly depends on image quality and overlap, struggles with reflective or textureless surfaces, and requires additional meshing and texturing steps to produce visually convincing results (see Section 4.3).

NeRF, on the other hand, represents a scene as a volumetric function that predicts color and density for each point in space given a viewing direction [45]. This implicit formulation enables the generation of photorealistic novel views that capture both intricate lighting variations and view-dependent appearance effects. Nevertheless, NeRF is computationally demanding: training often requires hours on high-end GPUs, and rendering new images involves dense sampling along camera rays, which makes real-time performance difficult to achieve.

3DGS combines aspects of both approaches. Like NeRF, it directly addresses the task of NVS, but instead of an implicit neural function, it employs an explicit set of Gaussian primitives that can be efficiently optimized [33]. At the same time, 3DGS relies on SfM results, similar to photogrammetry, to initialize camera poses and a sparse reconstruction, which then serve as input for the Gaussian optimization. This hybrid positioning explains why 3DGS has attracted significant attention: it enables real-time rendering while delivering visual quality that can surpass traditional photogrammetry, particularly in challenging scenarios such as fine vegetation or semi-transparent structures (see Section 5.3 for details).

5.2 FOUNDATIONS

To understand the internal concepts of 3D Gaussian Splatting (3DGS), it is necessary to introduce a set of theoretical foundations that link the method to classical and modern reconstruction approaches. While Chapter 3 outlined the general mathematical and algorithmic principles of 3D reconstruction, this section focuses on concepts that are particularly relevant for 3DGS: NVS as the overarching task, radiance fields as a volumetric representation

of scenes, 3D Gaussians as explicit primitives, and real-time rendering as the enabling technology for interactive applications. Together, these notions provide the conceptual bridge between general background knowledge and the specific pipeline of 3DGS, which will be detailed in Section 5.3.

5.2.1 *Novel View Synthesis (NVS)*

Novel View Synthesis NVS refers to the generation of new views of a scene from a limited set of recorded images. In other words, it creates images from perspectives that were not originally captured. To achieve this, the three-dimensional structure of the scene must be reconstructed in such a way that novel perspectives can be rendered computationally [5].

The central challenge of NVS is to represent the scene so that the geometry, texture, and lighting effects appear realistic from arbitrary viewpoints. Instead of relying on classical pipelines that reconstruct explicit geometry, such as meshes or dense point clouds, NVS focuses on creating representations that directly support the synthesis of unseen views.

In the context of this thesis, NVS is particularly relevant because 3DGS achieves novel view generation by modeling the scene through a collection of 3D Gaussian elements. Each Gaussian encodes spatial location, shape parameters, color, and opacity, which enables real-time rendering of novel viewpoints once the model has been optimized.

The following section introduces radiance fields as one prominent way to realize NVS through volumetric representations, before discussing 3DGS in detail as an alternative approach.

5.2.2 *Radiance Fields*

Radiance fields describe a scene implicitly. Instead of using explicit geometric shapes such as meshes or points, they define a continuous function. This function assigns each 3D position and viewing direction a color and a density value [53]. Rendering an image then means integrating color and density along the rays that pass through the scene.

The most prominent implementation of radiance fields is NeRF, introduced by Mildenhall et al. [45]. In NeRF, a Multi-Layer Perceptron (MLP) approximates the radiance field function from a set of calibrated input images. For each ray, the network predicts color and density at multiple sample points. These predictions are then combined through volumetric rendering to form the final image. This allows the generation of realistic new views that include fine lighting variations and view-dependent effects.

While radiance fields represent a breakthrough for NVS, they also face significant limitations: training is slow, rendering is computationally intensive due to the large number of samples per ray, and high-end GPUs are typically required [33, 45, 53]. These drawbacks limit their accessibility for everyday or resource-constrained applications.

In the context of this thesis, radiance fields are introduced as a conceptual foundation of NVS. They show the visual potential of implicit volumetric representations. However, their computational demands motivate newer methods such as 3DGS, which uses an explicit point-based scene representation and enables real-time rendering.

5.2.3 *3D Gaussians*

A simple point cloud represents a scene as a collection of individual dots in 3D space. Although useful, this often leads to a "digital" look with visible gaps and jagged edges when rendered, making it difficult to represent fine details such as fur or leaves [20]. 3DGS overcomes this limitation [33]. Instead of plain points, it represents a scene with *3D Gaussians*. Each Gaussian can be understood as a small, soft, and stretchable blob, defined by:

- A **position** in 3D space (its center point).
- A **covariance** matrix, which controls stretching and rotation (its 3D shape).
- An **opacity** value, which controls transparency.
- A **view-dependent appearance**, modeled with Spherical Harmonics (SH), allowing the color to change with viewing direction.

The key innovation is in the rendering. Instead of drawing hard dots, each 3D Gaussian is projected onto the 2D image plane as a disc (see Fig. 5.1). This projection results in an ellipse in screen space, which is then combined with a Gaussian-shaped alpha mask to form a soft splat (see Fig. 5.2) [33, 78]. These splats overlap and blend together seamlessly through an efficient filtering process, which reduces aliasing and produces continuous surfaces with high visual quality [78].

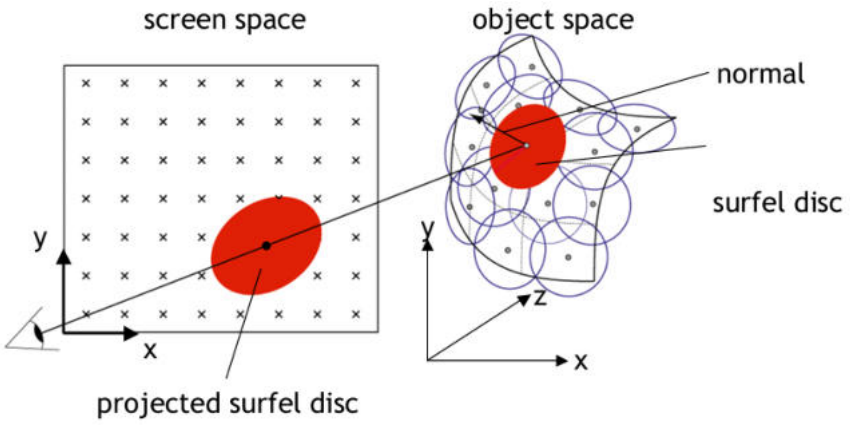


Figure 5.1: Illustration of projected disc rendering: surfels are represented as discs in object space (right), which are projected onto the screen space as ellipses (left). [20].

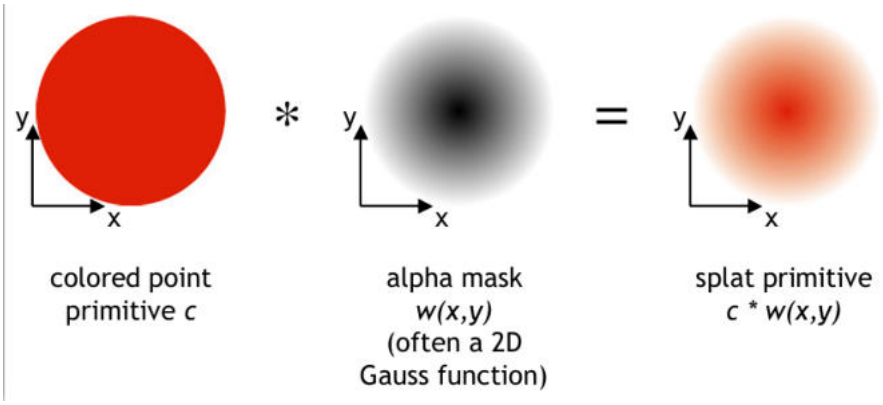


Figure 5.2: Construction of a splat primitive: a colored point primitive c is multiplied with an alpha mask $w(x, y)$, often represented as a 2D Gaussian function, resulting in a smooth splat. [20].

Visually, a scene made with 3D Gaussians appears as a cloud of colorful, soft-edged, and elastic blobs of different sizes and shapes. When viewed from arbitrary angles, these blobs combine to form a smooth and high-quality image of the underlying scene [33].

In summary, 3DGS builds on the classic ideas of point-based graphics [20] and splat rendering [78]. It employs modern optimization techniques to create a representation that is both efficient to render and capable of modeling complex visual detail more effectively than simple points or meshes.

Having introduced the conceptual foundation of 3D Gaussians as the core representation, Section 5.3 will now detail the specific mathematical parameterization and the algorithms that make up the full 3DGS pipeline.

5.2.4 *Real-Time Rendering*

Real-time rendering describes the process of producing and displaying images of a 3D scene fast enough to enable interactive visualization. In general computer graphics, this is often defined as at least 30 frames per second FPS, while immersive Virtual Reality (VR) typically requires 60 FPS or more to maintain presence and prevent motion sickness [35, 70].

In interactive 3D environments, such as those found in VR or Augmented Reality (AR), real-time rendering is essential. It enables natural interaction with virtual content, creates a convincing sense of immersion, and ensures that users can explore or manipulate 3D environments smoothly.

While classical photogrammetry produces static models through an offline process (see Chapter 4), modern methods such as 3DGS are designed from the ground up to meet this real-time requirement. By replacing slow mesh generation or neural raymarching with a rasterization-based pipeline (see Section 5.3.4), 3DGS achieves high frame rates immediately after a fast optimization step, enabling true interactive exploration [33].

5.3 METHOD: 3D GAUSSIAN SPLATTING IN DETAIL

Building on the conceptual foundations introduced in Section 5.2, this section presents in detail the internal pipeline of 3D Gaussian Splatting (3DGS). The approach uses an explicit set of 3D Gaussian elements together with a fast differentiable projection scheme to generate novel views in real time with high visual quality.

The pipeline can be divided into four main components: (1) the **parametrization** of 3D Gaussians, which defines their geometric and photometric attributes; (2) **adaptive density control**, which dynamically refines or prunes Gaussians during optimization; (3) a differentiable **rasterization process**, which projects Gaussians into image space; and (4) the final **rendering** stage, where splats are blended into photorealistic images.

Together, these components form the algorithmic core of 3DGS, allowing both efficiency and visual fidelity. The following subsections describe each of them in turn.

5.3.1 *Parametrization of the 3D Gaussians*

The core idea of 3DGS is to replace the discrete points of a classical point cloud with continuous, anisotropic 3D Gaussian primitives. Each Gaussian $\mathcal{G}_i$ is defined by the following parameters:

- **Position** $\mu_i \in \mathbb{R}^3$: the center of the Gaussian in 3D space.
- **Covariance Matrix** $\Sigma_i \in \mathbb{R}^{3\times3}$: controls anisotropic scaling and orientation, resulting in an ellipsoidal shape. To ensure that Σ_i stays positive semi-definite throughout the optimization process, it is parameterized as

  $$\Sigma_i = \mathbf{R}(\mathbf{q}_i) \cdot \mathrm{diag}(\mathbf{s}_i)^2 \cdot \mathbf{R}(\mathbf{q}_i)^\mathsf{T},$$

  where $\mathbf{q}_i$ is a unit quaternion representing rotation and $\mathbf{s}_i \in \mathbb{R}^3$ is a scale vector.
- **Opacity** $\alpha_i \in [0, 1]$: determines the transparency and blending weight during composition. For rendering, these opacities are combined using classical α-compositing.
- **Color**: modeled via SH, enabling smooth view-dependent changes. By default, SH of degree $L = 3$ are used, which corresponds to 16 basis functions per channel and thus 48 coefficients per Gaussian. For a detailed explanation of the SH formulation, see Section 5.3.1.1.

Together, these parameters define an anisotropic 3D Gaussian "blob". The initial set of Gaussians is typically initialized from a sparse SfM reconstruction (e.g., from *COLMAP*), which provides a strong geometric prior [33].

Visualization: Ellipsoids with variable density and color. **Initialization:** From SfM point cloud or random.

5.3.1.1 *Spherical Harmonics for View-Dependent Appearance*

In contrast to position, covariance, and opacity, the **color parameter** of a Gaussian requires special treatment to capture a view-dependent appearance. In 3DGS, directional appearance is represented using *spherical harmonics* (SH), which form a compact orthonormal basis for functions defined on the sphere. Each Gaussian stores a set of SH coefficients, typically up to degree $L = 3$ (48 coefficients in total), allowing its color to vary smoothly with the viewing direction. This enables complex view-dependent effects such as specular highlights to be modeled efficiently, although the parameter count per Gaussian also introduces a memory bottleneck in large reconstructions.

Figure 5.3 illustrates the first spherical harmonics basis functions. They form the building blocks for representing direction-dependent color components in 3DGS, where higher-order terms capture finer variations of light and reflection across viewing angles.

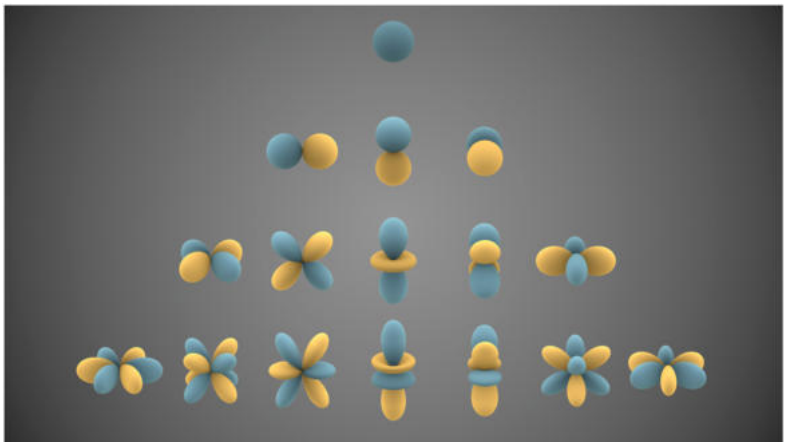


Figure 5.3: Visualization of the first spherical harmonics functions. Blue regions indicate positive values, yellow regions negative values, while the distance from the center corresponds to the magnitude of the function. [69]

Further mathematical details and implementation aspects are provided in Appendix .1.

5.3.2 *Adaptive Density Control*

The initial set of 3D Gaussians is derived from the sparse SfM point cloud (see Section 4.1.1). On its own, this initialization is insufficient to capture all the details of the scene. To achieve a dense and accurate representation, the system must add Gaussians where detail is missing and remove those that no longer contribute [33].

This process, called *adaptive density control*, is periodically executed during training. It relies on the positional gradient (how strongly a Gaussian wants to move) and its size to decide whether to **clone**, **split**, or **prune** a Gaussian (see Fig. 5.4) [33]:

- **Cloning (filling missing detail):** If a Gaussian shows a large gradient and has a small volume, it means that the model needs more coverage in that area. The algorithm duplicates the Gaussian and shifts the copy in the direction of the gradient. This adds more detail in regions that were underrepresented [33].
- **Splitting (refining structure):** If a Gaussian is too large to capture fine details, yet still exhibits a high gradient, it is split into two smaller Gaussians. Their scales are reduced (by a factor of $\varphi = 1.6$) and their positions are adjusted according to the original distribution.
- **Pruning (removing redundancy):** If a Gaussian opacity α falls below a small threshold, it is removed to avoid wasting resources on elements that do not contribute to rendering.

In essence, cloning fills the gaps, splitting refines the coarse regions, and pruning removes redundant elements. This dynamic adjustment enables the system to evolve from a sparse initialization to a dense, high-quality representation that is both accurate and efficient [33].

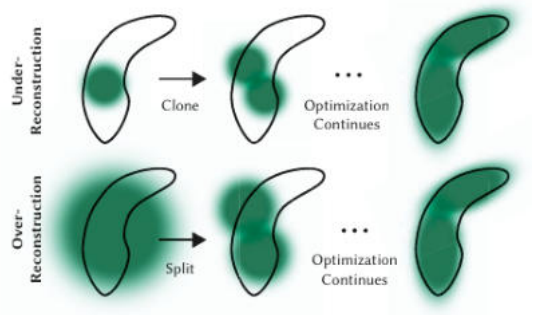


Figure 5.4: Adaptive Gaussian densification scheme from Kerbl et al. [33]. Top row: under-reconstruction (clone). Bottom row: over-reconstruction (split).

5.3.3 *Differentiable Rasterization*

A key innovation of 3DGS is its custom *differentiable rasterizer*. In contrast to NeRF methods that rely on slow ray marching, 3DGS employs a fast, tile-based rasterization technique that is fully differentiable, enabling both real-time rendering and efficient training [33].

The goal is to project millions of 3D Gaussians onto the 2D image plane and blend their colors correctly and efficiently. The process includes three main steps:

- **Frustum & Tile Culling:** Gaussians outside the camera view are discarded. The screen is divided into 16×16 pixel tiles. Each Gaussian is assigned to overlapping tiles, but for efficiency each tile stores at most 128 Gaussians [33].

- **Depth Sorting:** Within each tile, Gaussians are sorted by their depth using a GPU-based Radix sort, ensuring correct blending at interactive rates.
- **Alpha Blending:** Each Gaussian is projected onto the 2D image plane as an ellipse with an associated opacity. This principle is similar to the surfel disc projection shown in Figure 5.1, but extended with per-Gaussian covariance and opacity. Within each tile, the ellipses are blended from back to front using standard alpha compositing:

$$C = \sum_{i=1}^{N} \alpha_i \, c_i \prod_{j=1}^{i-1} (1 - \alpha_j), \tag{5.1}$$

where c_i is the color and α_i the opacity of the i-th Gaussian. This accumulation ensures that multiple overlapping Gaussians contribute smoothly, resulting in continuous and photo-realistic images (see Eq. 5.1).

DIFFERENTIABILITY IN TRAINING

The rasterizer is central to not only rendering, but also optimization. During the backward pass, it computes how each Gaussian parameter (position, scale, rotation, opacity, color) influence the final image and the loss:

- Gradients are propagated throughout the blending process.
- Unlike earlier methods, any number of Gaussians can contribute per pixel, so even deep Gaussians remain trainable.
- Gradients for rotation and scale are computed explicitly for stability and speed, avoiding the overhead of generic automatic differentiation [33].

In summary, the differentiable rasterizer combines the efficiency of GPU rasterization with the learning capability of volumetric rendering, making 3DGS both fast and trainable [33].

5.3.4 *Rendering Pipeline*

The rendering pipeline of 3DGS integrates initialization, adaptive density control, and differentiable rasterization into a single workflow. It begins with a sparse point cloud from SfM, which provides the initial Gaussians and camera poses (see Section 4.1.1). From there, the system alternates between density control and rasterization until the scene is represented at high quality. An overview of this process is shown in Figure 5.5, adapted from Kerbl et al. [33].

Once the Gaussians have been optimized, the rendering of a novel view is extremely efficient and executed in real time. The process performed for each frame can be summarized in three main steps [33]:

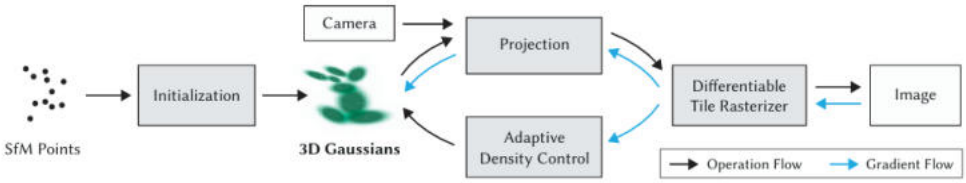


Figure 5.5: Pipeline of 3D Gaussian Splatting: starting from an SfM point cloud, Gaussians are optimized through adaptive density control and differentiable rasterization. The final stage produces real-time renderings suitable for interactive exploration. [33].

1. **Projection:** Each Gaussian, defined by its mean μ, covariance matrix Σ, and opacity α, is projected into the view space of the camera. In the image plane, this results in a 2D ellipse with covariance Σ'. The Gaussian color is evaluated from its SH coefficients.

2. **Tile-Based Sorting:** The screen is divided into tiles (e.g., 16×16 pixels). Each Gaussian is assigned to overlapping tiles, but for efficiency each tile stores at most 128 Gaussians. A GPU-based Radix sort orders Gaussians by tile ID and depth, ensuring a correct rendering order.

3. **Alpha Blending and Composition:** Within each tile, the Gaussians are blended back-to-front using the alpha compositing rule (Eq. 5.1). Once the accumulated opacity is close to 1, deeper Gaussians are skipped, a technique known as *early termination*.

This single-pass rasterization pipeline avoids costly ray marching and achieves real-time performance, often exceeding 100 FPS at 1080p resolution. Its efficiency makes 3DGS suitable for interactive applications and one of the most impactful recent methods in neural rendering [33].

5.4 PRACTICAL IMPLEMENTATION AND TOOLS

After outlining the theoretical foundations and internal workflow of 3D Gaussian Splatting (3DGS) in the preceding sections, this part turns to its practical implementation. The goal is to outline how the method can be applied in real-world scenarios, which tools are available, and what requirements need to be considered.

We discuss both the open-source reference implementation by Kerbl et al. [33], as well as software solutions and platforms that integrate or build upon 3DGS. These include preprocessing systems for initialization, the training and rendering pipeline of the original codebase, and user-oriented tools that make the method accessible beyond expert communities. Together, these aspects form the bridge from the algorithmic concept to its use in practice, which is crucial for the comparative analysis in Chapter 6.

5.4.1 *Pre-processing with COLMAP*

Initialization of 3DGS requires camera poses and a sparse geometric prior. These are obtained via incremental SfM (see Section 4.1.1) using *COLMAP* , which estimates the intrinsics, extrinsics of the camera and a sparse 3D point cloud [56]. This sparse reconstruction is sufficient to initialize the first set of Gaussians [33]. Importantly, dense MVS (see Section 4.1.2) is not required, avoiding both the computational cost and the fragility of dense reconstruction steps [33].

COMMON PITFALLS. Since 3DGS relies on the accuracy of the *COLMAP* SfM output, typical issues in this step directly propagate into the Gaussian initialization:

- **Pose inaccuracies.** Errors in camera registration or drift in estimated poses produce inconsistent point clouds, leading to misaligned Gaussian initialization. This often results in blurred or unstable reconstructions and, in severe cases, flickering artifacts in novel views. [56]
- **Insufficient image coverage.** *COLMAP* 's reconstruction quality strongly depends on sufficient image overlap. If the input photos are too few or poorly distributed, parts of the scene remain uncovered and Gaussians cannot be initialized there. [26]

5.4.2 *Training Pipeline (Original Code)*

The official implementation of 3DGS follows a streamlined pipeline with three main stages. The input is a sparse reconstruction from *COLMAP* (see Section 5.4.1), which provides intrinsic camera data, extrinsic data, and a sparse 3D point cloud. [26, 56]

1. **Data conversion.** The script `convert.py` transforms the *COLMAP* output into the internal format required by the 3DGS codebase. The camera parameters and the sparse points are converted into Gaussian primitives for initialization. [16, 33]
2. **Training.** The core optimization is performed with `train.py`, which refines the position, scale, opacity, and SH coefficients of the Gaussians. Key hyperparameters include the learning rate for Gaussian attributes and density thresholds that control adaptive densification (splitting, cloning, and pruning) [16, 33].
3. **Rendering.** After training, novel views can be generated with `render.py`, which uses the differentiable rasterizer to produce images or video sequences. In addition, the repository provides an OpenGL-based real-time viewer for interactive exploration of the trained model. [16, 33] This uses the tile-based rasterization pipeline described in Section 5.3.3, enabling real-time rendering.

In summary, the reference codebase integrates *COLMAP* -based initialization with a dedicated training and rendering workflow. This makes it possible to obtain real-time 3D scene representations while avoiding the computational overhead of dense MVS reconstruction. [33]

5.4.3 *From Code to Application: User-Oriented Tools and Platforms*

Although the original codebase offers full control over the training process, its setup requires technical expertise (CUDA, PyTorch, *COLMAP* preprocessing) and access to high-end GPUs, which limits accessibility for non-experts. [16, 33]

To bridge this gap, several user-oriented platforms have emerged that automate image input to visualization pipeline. These can be broadly grouped into three categories:

- **Desktop applications** with a local GUI that integrate *COLMAP* and Gaussian splatting but still require a capable GPU.
- **Managed cloud services** that process uploaded images on remote servers, removing local hardware requirements but introducing costs and external dependencies.
- **Cloud infrastructure solutions** that provide access to GPUs via simplified front-ends or direct code execution environments.

Concrete examples of such tools and platforms, along with their technical requirements, workflows, and cost models, are introduced in the following Chapter 6, while their practical evaluation is presented later in the Results and Discussion Chapter 7.

5.5 TECHNICAL LIMITATIONS

Despite its strengths in efficiency and visual quality, 3DGS also faces several technical limitations that are important to consider in practical applications [33, 36]:

- **High memory requirements.** Each scene is represented by millions of Gaussians storing position, scale, opacity, and SH coefficients. For large-scale environments, this number grows rapidly, resulting in high GPU memory consumption and making reconstructions difficult on hardware with limited Video Random Access Memory (VRAM). [33]
- **Reflective and transparent surfaces.** Like other image-based rendering approaches, Gaussian Splatting struggles with mirror-like or transparent materials. Their appearance depends strongly on view-dependent effects and global light transport, which are not fully captured by local Gaussian primitives. Reflections and refractions are, therefore, often rendered inaccurately. [36]

- **Extreme viewing angles.** Quality decreases for viewpoints far outside the range of the input images. Under such extrapolation, artifacts emerge because Gaussians were not optimized for unseen regions, a problem that is particularly visible in sparse data sets or highly anisotropic structures. [33, 36]

In summary, 3DGS achieves real-time performance while maintaining high visual quality. However, its applicability is limited by hardware requirements, material properties, and training data coverage. These factors are crucial when comparing 3DGS with classical methods such as photogrammetry (see Chapter 4.3).

6

METHOD SELECTION AND EVALUATION SETUP

The preceding chapters have established the theoretical foundation for this comparative study by introducing two major paradigms of 3D reconstruction. Both classical photogrammetry and modern neural rendering rely on SfM to estimate camera poses and sparse 3D structures from multiple overlapping images. While photogrammetry continues this process through dense multi-view stereo and surface reconstruction to produce watertight meshes, (3DGS) leverages the recovered camera parameters to optimize a differentiable, point-based scene representation designed for real-time rendering and high visual fidelity.

Building upon this shared foundation, this chapter defines the methodological framework for the practical comparison of these approaches across different tools and usage scenarios. It serves as a bridge between theoretical analysis and experimental evaluation, structured around three key components:

- **Tool Selection:** Mobile applications, desktop software, and source code implementations, along with recording devices (*iPhone 16 Pro* and *DJI Mini 4K* drone)
- **Experimental Setup:** Hardware configurations (local workstation and AWS cloud instance) ensuring reproducibility
- **Evaluation Design:** Qualitative criteria and quantitative metrics applied to the selected datasets

This framework enables the systematic evaluation in Chapter 7, analyzing performance across Everyday Scenarios, Edge-Cases, and Robustness Tests.

6.1 OVERVIEW OF TESTED TOOLS

The evaluation considers three categories of tools: mobile applications, desktop software, and a source code implementation. This selection ensures coverage of both end-user solutions and research-level methods.

- **Mobile applications:** *Kiri Engine* and *Polycam*. These applications run on modern smartphones, with photogrammetry supported from iPhone 12 Pro (LiDAR) or equivalent Android devices. While accessible and easy to use, they require relatively recent hardware for reliable processing.
- **Desktop software:** *Meshroom* (open-source photogrammetry) and *PostShot*, which is currently in its beta version and is based on 3DGS. Both tools allow for local or hybrid processing with extended parameter

control. In practice, both tools require a CUDA-capable NVIDIA GPU, with at least mid-range consumer GPUs recommended for usable runtimes.

- **Source code implementation:** The official 3DGS repository (GitHub) was fully integrated into the evaluation. Models were trained and tested on GPU-accelerated AWS cloud servers to ensure controlled experimental conditions. This setup allowed for detailed benchmarking of the method with respect to runtime and output quality, complementing the results obtained from mobile and desktop tools.

Detailed technical specifications, including device support, input and export formats, and system requirements, are provided in Appendix .3.

6.2 EXPERIMENTAL SETUP

To ensure comparability and reproducibility, this section describes the hardware and software environments in which the tested tools were executed, as well as the data acquisition setup.

6.2.1 *Mobile Applications*

The mobile applications *Kiri Engine* and *Polycam* were executed on an Apple *iPhone 16 Pro*. In contrast to the standard workflow of capturing images directly within the app, all tests in this study used the same custom datasets introduced in Section 6.4.1 to ensure comparability across methods. The images were uploaded to the respective applications, where processing was performed via the vendors' cloud infrastructure. Detailed device specifications are listed in Appendix .3.

6.2.2 *Desktop Software*

The desktop software *Meshroom* and *PostShot* were executed on a local workstation with the following configuration:

Table 6.1: Workstation specifications for desktop software testing.

Component	**Specification**
CPU	Intel Core i7 (4th Generation)
GPU	NVIDIA GeForce RTX 2070 Ti (CUDA Compute Capability 7.5)
RAM	32 GB
Operating System	Windows 10 Pro

Both *Meshroom* and *PostShot* were run with default configurations, using GPU acceleration on the RTX 2070 Ti (with *PostShot* trained for 30,000 iterations).

6.2.3 *Source Code Implementation*

The official 3DGS repository was executed on GPU-accelerated AWS cloud servers. In accordance with the repository requirements [16], the exact CUDA and PyTorch versions were installed. *COLMAP* was compiled in headless mode, as the instance provided no graphical user interface. Training was executed headless and final renderings generated without GUI.

Table 6.2: AWS EC2 specifications for source code experiments.

Component	**Specification**
Instance type	g5.2xlarge
GPU	NVIDIA A10G (24 GB VRAM)
vCPUs	8
Memory (GiB)	32
Storage	EBS volume (SSD), 300 GB
Operating System	Ubuntu 22.04 LTS
CUDA / PyTorch	As specified in the official repository
Additional tools	*COLMAP* (headless), git, cmake, ffmpeg, ImageMagick

All datasets introduced in Section 6.4.1 were trained for 30,000 iterations (to ensure comparability with the training conditions applied in *PostShot*) at full resolution (1280×720, portrait: 720×1280). An exception are the *Citadel* datasets, which had a native 4K resolution (3840×2160). The *Citadel200* dataset was downscaled to half resolution due to GPU memory constraints.

6.2.4 *Drone-based Data Acquisition*

Selected datasets were recorded using a *DJI Mini 4K* drone to obtain aerial perspectives. The aerial imagery complements ground-level smartphone captures, providing additional viewpoints for larger structures.

6.3 EVALUATION CRITERIA

To fully capture reconstruction quality, three perspectives are applied: quantitative image metrics, visual inspection, and tool-related aspects. The metrics are described in Section 6.3.1, while the criteria for visual and tool evaluation are summarized in Tables 6.3 and 6.4.

6.3.1 *Metric Evaluation*

Quantitative evaluation relies on objective measures that assess how closely a reconstructed model matches the original image data. In this study, three complementary metrics are employed: PSNR, SSIM, and LPIPS. Together, they

capture different aspects of reconstruction quality, ranging from pixel-level fidelity to perceptual similarity. Nevertheless, as discussed in the following paragraph on limitations, these metrics cannot be applied consistently across all tools and datasets.

PEAK SIGNAL-TO-NOISE RATIO (PSNR)

PSNR expresses the ratio between the maximum possible pixel value and the mean squared error (MSE) between a reference image I and a reconstruction $\hat{I}$. It is reported in decibels (dB):

$$\text{PSNR} = 10 \cdot \log_{10} \left(\frac{MAX_I^2}{\text{MSE}} \right), \tag{6.1}$$

with

$$\text{MSE} = \frac{1}{N} \sum_{i=1}^{N} \left(I_i - \hat{I}_i \right)^2, \tag{6.2}$$

where MAX_I is the maximum pixel value (e.g., 255 for 8-bit images) and N is the number of pixels. Higher PSNR values indicate closer fidelity to the reference [32].

STRUCTURAL SIMILARITY INDEX (SSIM)

SSIM compares a reference image x and a distorted image y by evaluating luminance, contrast, and structural information. It is defined as:

$$\text{SSIM}(x, y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)}, \tag{6.3}$$

where μ_x and μ_y denote the mean luminance values, σ_x^2 and σ_y^2 represent the variances, and σ_{xy} is the covariance between both images. C_1 and C_2 are small constants added to stabilize the division. An SSIM score close to 1 indicates a high degree of structural similarity [68].

LEARNED PERCEPTUAL IMAGE PATCH SIMILARITY (LPIPS)

LPIPS evaluates perceptual similarity by analyzing differences in deep feature representations between a reference image x and its reconstruction $\hat{x}$, extracted from a pretrained convolutional network (e.g., VGG). For each layer l of the network, feature activations y_l and $\hat{y}_l$ are compared:

$$\text{LPIPS}(x, \hat{x}) = \sum_l \frac{1}{H_l W_l} \sum_{h,w} w_l \cdot \| \hat{y}_{l,h,w} - y_{l,h,w} \|_2^2, \tag{6.4}$$

where H_l, W_l are spatial dimensions of the feature map and w_l are learned scalar weights [76]. Lower values indicate higher perceptual similarity; scores below 0.3 typically correspond to images with no or only minor perceptual differences. In contrast to PSNR and SSIM, LPIPS is designed to approximate

human visual perception more closely, since it evaluates similarity based on learned feature representations rather than raw pixel differences.

LIMITATIONS OF THE EVALUATION APPROACH

The applied metrics (PSNR, SSIM, LPIPS) require exact camera poses to render reference views, a condition only fulfilled for models generated with the 3DGS/*COLMAP* pipeline. For mobile and desktop tools, no exportable poses are available, restricting their evaluation to qualitative assessments.

This results in a dual evaluation strategy: quantitative metrics for the 3DGS source code implementation, and qualitative visual inspection for all other tools. The qualitative assessment follows the criteria defined in Section 6.3.2 and Section 6.3.3, focusing on geometric accuracy, texture quality, and overall visual plausibility based on human perception.

6.3.2 *Visual Evaluation*

Since metric evaluation cannot be applied consistently across all tools, visual inspection becomes the primary method to ensure comparability. It also captures aspects of model quality that numerical measures cannot reflect. Two dimensions are considered: geometric accuracy and texture quality. The criteria are summarized in Table 6.3, with each item rated on a 1–5 scale.

Table 6.3: Criteria for visual evaluation of reconstructed models.

Dimension	**Criteria**
Geometric Accuracy	Detail preservation (fine structures, edges) Completeness (missing areas or holes) Distortions (unnatural deformations) Noise (surface smoothness)
Texture Quality	Color fidelity (accuracy of colors) Resolution (sharpness of textures) Consistency (seams, artifacts)

6.3.3 *Tool Evaluation*

Beyond reconstruction quality, the tools themselves are compared with respect to processing speed, usability, scalability, model quality, and costs. The dimensions are summarized in Table 6.4.

6.4 EXPERIMENTAL DESIGN

This section describes the datasets, evaluation protocol, and experimental procedures used to compare the reconstruction methods across different scenarios and conditions.

Table 6.4: Criteria for tool evaluation.

Dimension	**Criteria**
Processing Speed	Time required to generate a model after data input or capture, including potential delays due to queuing or hardware limitations
Usability and Workflow	Ease of use (intuitiveness of the interface) Workflow automation (manual steps vs. one-click processing) Degree of parameter customization
Scalability	Performance on large datasets (handling of complex or extensive scenes) Dependence on hardware resources (GPU, RAM, cloud capacity)
Model Quality	Strengths and weaknesses of each tool in model reconstruction
Costs	Cost model of the tool, including free versions, subscription models, cloud-related expenses, or one-time licenses

6.4.1 *Datasets and Evaluation Protocol*

The evaluation uses two dataset categories (introduced below) to assess both typical capture conditions and challenging reconstruction scenarios. All datasets were recorded by the author, except for the external *Citadel* dataset, which originates from a professional drone recording by National Television Afghanistan (RTA Mili TV) at 4K resolution [4]. The *Castle* datasets were captured with a *DJI Mini 4K* drone, while all remaining datasets were recorded with an *iPhone 16 Pro*.

EVERYDAY SCENARIOS

This category reflects common capture conditions with varying lighting, perspectives, and backgrounds. It includes: *Bush*, *Castle Frontside*, and *Citadel*. The *Citadel* dataset exemplifies typical, freely accessible imagery used in everyday 3D reconstruction scenarios.

EDGE-CASES

These datasets represent challenging conditions with strong deviations from optimal acquisition, such as motion blur, dense vegetation, or inconsistent viewpoints. The category includes: *Owl*, *Forest 360-Degree Shot*, *Car Ride*, *Walkway*, and *Castle Compound*.

DATA SPLITS AND EVALUATION PROTOCOL

For the 3DGS source code implementation, models followed an 80/20 train-test split according to the official protocol, using the test set for metric eval-

uation (PSNR, SSIM, LPIPS). All other tools (mobile applications and desktop software) were trained on 100% of the data, as they lack dedicated evaluation splits and exportable camera poses required for metric assessment.

Visual comparisons across all tools are based on consistent rendering sources: screenshots from *Kiri Engine*, *Polycam*, *Meshroom*, and *PostShot*, and rendered test views from 3DGS source code evaluation scripts.

6.4.2 *Experimental Procedure*

The evaluation applies the criteria from Section 6.3 through a structured three-phase procedure:

- **Edge-Cases – Feasibility Testing:** All tools are tested on challenging datasets to determine if they can produce recognizable 3D models. The feasibility definition is given in Section 6.4.2.1. The outcome is binary (Yes/No), complemented by qualitative observations.
- **Everyday Scenarios - Quality Assessment:** Mobile applications and desktop software are evaluated using the visual criteria (Table 6.3) with ratings on a 1–5 scale for geometric accuracy and texture quality.
- **Robustness - Scalability Testing:** Methods are tested on the Citadel dataset with varying input sizes (200, 100, 50, 20 images) to evaluate sensitivity to reduced image coverage and scalability with larger inputs. Visual evaluation follows the same criteria as everyday scenarios.

The evaluation follows a tool-specific approach: 3DGS source code models are assessed quantitatively using the metrics defined in Section 6.3.1, while mobile applications and desktop software undergo qualitative visual inspection. To prevent bias, quantitative evaluation was performed after qualitative assessment.

6.4.2.1 *Feasibility Criteria*

A reconstruction is considered feasible if it yields a coherent and recognizable 3D representation of the intended object or scene, prioritizing scene-wide structural coherence over fine detail. Dataset-specific notes and examples are provided in Appendix .4.

7 RESULTS AND DISCUSSION

This chapter presents and interprets the results from the experimental design in Section 6.4. The analysis follows the dataset taxonomy (Section 6.4.1) and evaluation procedures (Section 6.3): we first report photogrammetry-based reconstructions, distinguishing between *feasibility tests* on challenging *Edge-Cases* (see Section 6.4.2.1) and the *visual evaluation* of *Everyday Scenarios* (details in Section 6.3.2).

Subsequently, we present results for 3D Gaussian Splatting (3DGS) using the same structure. Tool-based 3DGS models (*Kiri Engine*, *Polycam*, *PostShot*) are assessed qualitatively, while models trained with the official 3DGS source code are evaluated quantitatively using PSNR, SSIM, and LPIPS (Section 6.3.1).

Finally, we include robustness analyses on the *Citadel* dataset—investigating how reconstruction quality responds to varying numbers of input images and to different tools—and close with a cross-method comparison, a tool evaluation, and the limitations of this study.

7.1 PHOTOGRAMMETRY MODELS RESULTS

This section summarizes results from three photogrammetry pipelines: two mobile applications (*Kiri Engine*, *Polycam*) and a desktop solution (*Meshroom*). We report (i) feasibility on *Edge-Case* datasets—i.e., whether a recognizable 3D reconstruction can be produced (criteria in Appendix .4) —and (ii) a visual evaluation on *Everyday Scenarios* using the rubric defined in Section 6.3.2 (Table 6.3). All figures show representative views of the reconstructed models; tables list the corresponding feasibility (Yes/No) and 1–5 visual scores.

7.1.1 *Feasibility Tests on Edge-Case Datasets (Photogrammetry)*

We qualitatively report the per-dataset feasibility outcomes below.

7.1.1.1 *Owl Dataset (Photogrammetry)*

Table 7.1 summarizes feasibility for the *Owl* dataset. As shown in Figure 7.1, both *Kiri Engine* and *Polycam* reconstructed a recognizable owl, although parts of the back are missing in both models. By contrast, *Meshroom* failed during dense reconstruction/texturing (cf. Sections 4.1.2, 4.1.4) and yielded only a sparse, incomplete structure.

7.1.1.2 *Car Ride Dataset (Photogrammetry)*

Table 7.2 summarizes feasibility for the *Car Ride* dataset. As illustrated in Figure 7.2, both *Kiri Engine* and *Polycam* produced only fragmented, distorted

Table 7.1: Feasibility results for the *Owl* dataset (photogrammetry).

Photogrammetry Tool	**Feasibility**
Kiri Engine	Yes
Polycam	Yes
Meshroom	No

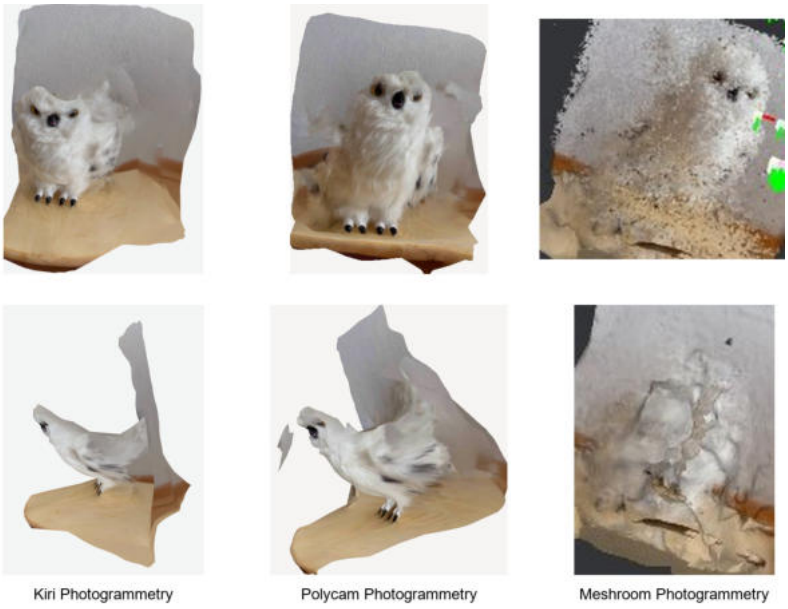


Figure 7.1: Photogrammetry-based reconstruction of the *Owl* dataset using three different tools: Kiri Engine (left), Polycam (center), and Meshroom (right). A toy owl was placed on a rotating turntable and recorded from all sides to capture the dataset.

geometry and did not yield a coherent 3D structure. Given the high motion blur and rapid viewpoint changes during capture (30–50 km/h), neither app produced a usable model. By contrast, *Meshroom* failed during the SfM step and produced no model.

Table 7.2: Feasibility results for the *Car Ride* dataset (photogrammetry).

Photogrammetry Tool	**Feasibility**
Kiri Engine	No
Polycam	No
Meshroom	No

7.1.1.3 *Forest 360-Degree Shot Dataset (Photogrammetry)*

Table 7.3 summarizes feasibility for the *Forest 360-Degree Shot* dataset; hereafter Forest 360°. As shown in Figure 7.3, both *Kiri Engine* and *Polycam* failed to generate coherent 3D models: the reconstructions are fragmented, with

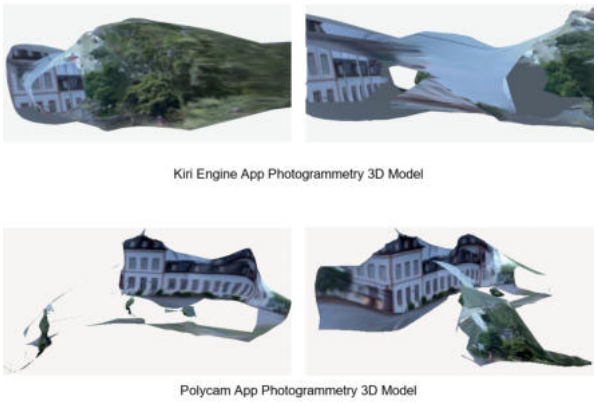


Figure 7.2: Photogrammetry-based reconstructions of the *Car Ride* dataset using the *Kiri Engine* and *Polycam* apps. Recordings were taken from the passenger seat at 30–50 km/h. Screenshots are arranged chronologically from left to right.

distorted geometry and inconsistent surfaces, which is consistent with repetitive tree/foliage patterns that provide too few distinctive features for reliable matching. *Meshroom* likewise produced no model.

Table 7.3: Feasibility results for the *Forest 360-Degree Shot* dataset (photogrammetry).

Photogrammetry Tool	**Feasibility**
Kiri Engine	No
Polycam	No
Meshroom	No

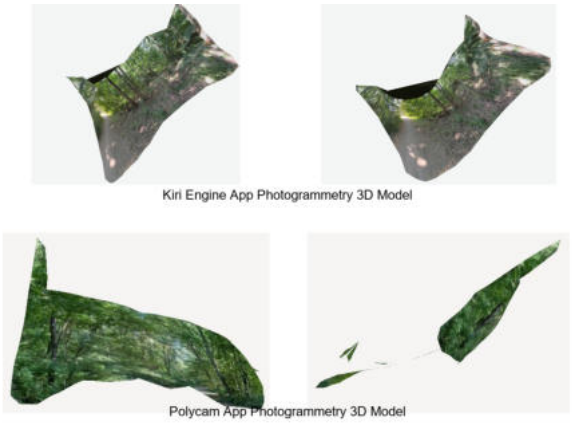


Figure 7.3: Photogrammetry-based reconstructions of the *Forest 360-Degree Shot* dataset. The top row shows Kiri Engine; the bottom row shows Polycam.

7.1.1.4 *Walkway Dataset (Photogrammetry)*

Table 7.4 shows the feasibility for the *Walkway* dataset. As illustrated in Figure 7.4, outputs from *Kiri Engine* and *Polycam* do not meet the feasibility criterion of a coherent and recognizable scene. Although some structures such as paths and trees were visible, the models remained fragmented and inconsistent, with distorted geometry and stretched textures. *Meshroom* produced no model.

Table 7.4: Feasibility results for the *Walkway* dataset (photogrammetry).

Photogrammetry Tool	**Feasibility**
Kiri Engine	No
Polycam	No
Meshroom	No

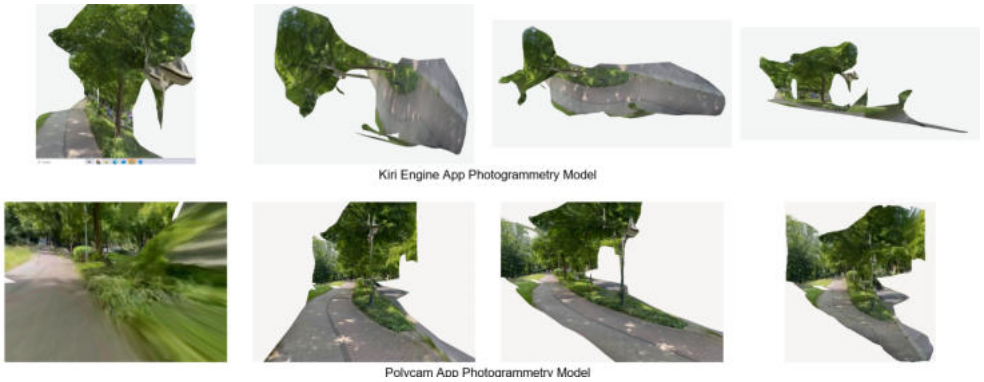


Figure 7.4: Photogrammetry-based reconstructions of the *Walkway* dataset, captured while walking along a path. Top: *Kiri Engine*; bottom: *Polycam*.

7.1.1.5 *Castle Compound Dataset (Photogrammetry)*

Table 7.5 summarizes feasibility for the *Castle Compound* dataset. As illustrated in Figure 7.5, both *Kiri Engine* and *Polycam* produced recognizable models of Schloss Philippsruhe, but the combined dataset led to misalignments, ghost structures, and fragmented roof areas. While facades remain partly recognizable, the geometry is incoherent and deviates from the castle's symmetric U-shaped layout. *Meshroom* produced no model.

Table 7.5: Feasibility results for the *Castle Compound* dataset (photogrammetry).

Photogrammetry Tool	**Feasibility**
Kiri Engine	Yes
Polycam	Yes
Meshroom	No

Figure 7.5: Photogrammetry-based reconstructions of Schloss Philippsruhe (*Castle Compound* dataset). Top: Kiri Engine. Bottom: Polycam.

Key takeaway: Photogrammetry is feasible for static, well-textured objects, but fails with motion, low parallax, or repetitive vegetation; cloud apps fare better than Meshroom, and combined captures are feasible but prone to misalignment/ghosting.

7.1.2 *Photogrammetry Visual Evaluation of Everyday Scenarios*

We now assess the visual quality of photogrammetry results on the *Everyday Scenario* datasets (e.g., *Castle Frontside*, *Bush*). Evaluations follow the rubric in Section 6.3.2 (Table 6.3) with 1–5 scores per criterion. Unlike the feasibility tests, this section compares successfully reconstructed models across tools.

7.1.2.1 *Castle Frontside Dataset (Photogrammetry)*

Table 7.6 and Figure 7.6 show that photogrammetry captures the overall U-shaped layout of Schloss Philippsruhe in the *Castle Frontside* dataset. *Kiri Engine* and *Polycam* yield recognizable facades and roofs, though textures are blurry and fine architectural details are missing. *Meshroom* reconstructs the basic geometry but shows irregular surfaces and stronger artifacts, which is reflected in the lower scores.

7.1.2.2 *Bush Dataset (Photogrammetry)*

Table 7.7 summarizes the visual evaluation for the *Bush* dataset. As shown in Figure 7.7, both *Kiri Engine* and *Polycam* capture the overall bush volume, but fine leaf structures are missing and textures appear blurry. Distortions and

Table 7.6: Visual evaluation of photogrammetry-based reconstructions for the *Castle Frontside* dataset.

Tool / App	Detail pres.	Complete-ness	Distortions	Noise	Color fidelity	Reso-lution	Consis-tency	Total Points
Kiri Engine	3	4	3	3	3	3	3	22
Polycam	3	4	3	3	3	3	3	22
Meshroom	2	3	2	2	3	2	2	16

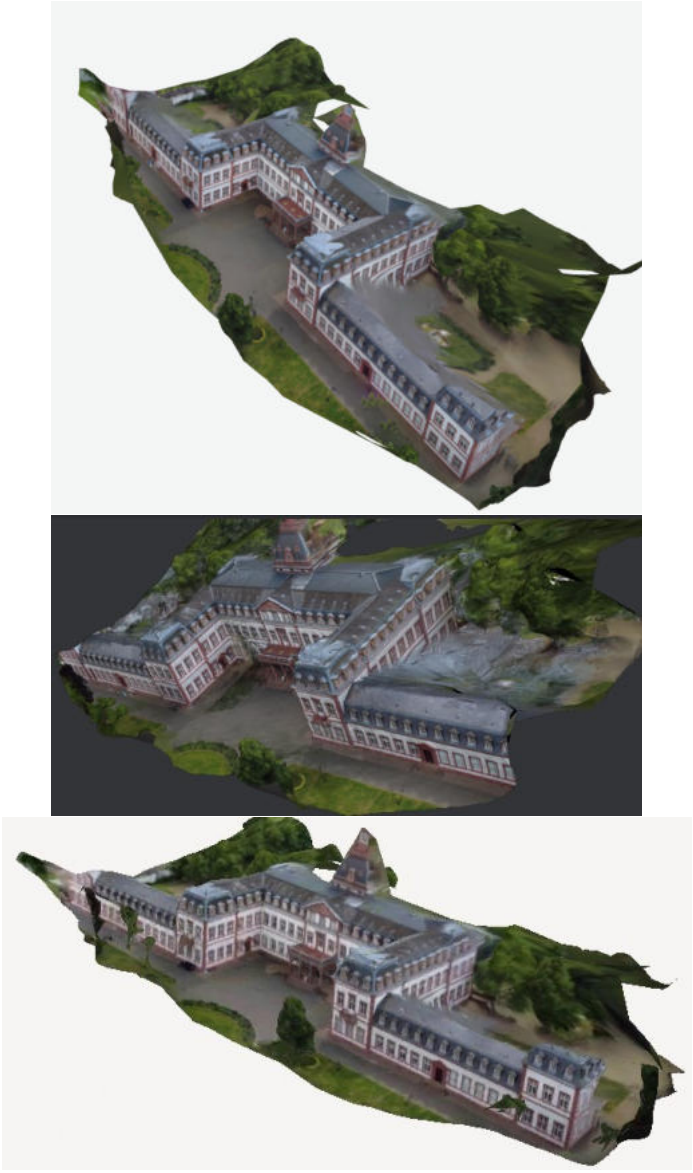


Figure 7.6: Photogrammetry-based reconstructions of Schloss Philippsruhe (front side): top Kiri Engine, middle Meshroom, bottom Polycam.

noise are present in both models; overall, *Kiri Engine* scores slightly higher than *Polycam* in our rubric. *Meshroom* did not yield a coherent model and produced only fragmented structures.

Table 7.7: Visual evaluation of photogrammetry-based reconstructions for the *Bush* dataset.

Tool / App	Detail pres.	Complete-ness	Distortions	Noise	Color fidelity	Reso-lution	Consis-tency	Total Points
Kiri Engine	2	2.5	3	3	3	2	1.5	17
Polycam	2	2	2	2	3	2	2	15
Meshroom	1	1	1	1	2	1	1	8

Figure 7.7: Photogrammetry-based reconstructions of the *Bush* dataset. Top: Kiri Engine; middle: Polycam; bottom: Meshroom. Multiple viewpoints are shown, including side and top views.

7.1.3 *Robustness Test for Photogrammetry Citadel Model*

We evaluate photogrammetry robustness on the *Citadel* dataset by varying the number of input images (20, 50, 100, 200) for *Polycam*, *Kiri Engine*, and *Meshroom*. Using the visual rubric in Section 6.3.2 (Table 6.3), we examine how input size and tool choice affect completeness and perceived visual quality under the same capture conditions.

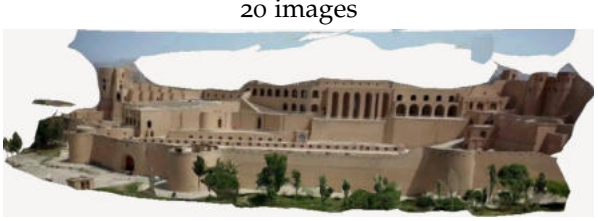
7.1.3.1 *Citadel Dataset (Polycam – Photogrammetry)*

Table 7.8 and Figure 7.8 summarize the *Polycam* photogrammetry results on the *Citadel* dataset. With 20 images, the facade is recognizable but roofs/upper sections remain incomplete; distortions, sky gaps, and irregular surfaces are visible, with soft textures despite reasonable colors. Using 50 or 100 images yields more coherent geometry and clearer facades; **50 images achieves the highest total (30 points)**, with 100 images second (26 points). At 200 images, coverage is most complete (Completeness=5) but distortions and inconsistency increase, reducing visual fidelity. More input does not necessarily improve photogrammetry quality.

Table 7.8: Visual evaluation of *Polycam* photogrammetry reconstructions for the *Citadel* dataset (20–200 images).

Polycam / Images	**Detail pres.**	**Complete-ness**	**Distortions**	**Noise**	**Color fidelity**	**Reso-lution**	**Consis-tency**	**Total Points**
20 images	3	2	3	3	5	3	3	22
50 images	4	4	3	5	5	5	4	30
100 images	3	4	3	3	5	4	4	26
200 images	2	5	4	2	5	2	5	25

20 images



50 images

100 images

200 images

Figure 7.8: Polycam photogrammetry-based reconstructions of the *Citadel* dataset.

7.1.3.2 *Citadel Dataset (Kiri Engine – Photogrammetry)*

Table 7.9 and Figure 7.9 summarize the *Kiri Engine* photogrammetry results on *Citadel*. With 20 images, the model is blurred and incomplete (sky largely

missing). At 50 images, coverage improves, but wall edges remain soft. The 100- and 200-image models are most complete and visually strongest, with sharper textures and more stable geometry; scores tie at 29. Distortions remain similar across all runs, while noise and resolution improve with image count, indicating diminishing returns beyond ∼100 images.

Table 7.9: Visual evaluation of Kiri Engine photogrammetry reconstructions for the *Citadel* dataset (20–200 images).

Kiri Engine / Images	**Detail pres.**	**Completeness**	**Distortions**	**Noise**	**Color fidelity**	**Resolution**	**Consistency**	**Total Points**
20 images	2	2	3	2	5	2	3	19
50 images	3	3	3	3	5	3	3	23
100 images	4	4	3	4	5	5	4	29
200 images	4	4	3	4	5	5	4	29

7.1.3.3 *Citadel Dataset (Meshroom – Photogrammetry)*

Table 7.10 and Figure 7.10 summarize the *Meshroom* photogrammetry results on *Citadel*. With 20 images, Meshroom produces a sufficiently coherent reconstruction with recognizable facades and walls (Total = 16). At 50 images, quality improves slightly (more detail/resolution; Total = 18), although distortions remain. At 100 images, the result degrades marginally relative to 50 (notably stronger artifacts in the sky and less consistent color; Total = 17). At 200 images, the texture step fails, producing fragmented surfaces and poor color reproduction (Total = 10).

Table 7.10: Visual evaluation of Meshroom photogrammetry reconstructions for the *Citadel* dataset (20–200 images).

Meshroom / Images	**Detail pres.**	**Completeness**	**Distortions**	**Noise**	**Color fidelity**	**Resolution**	**Consistency**	**Total Points**
20 images	1	3	2	2	3	2	3	16
50 images	2	3	3	2	3	3	2	18
100 images	3	3	2	2	2	3	2	17
200 images	1	4	1	1	1*	1	1	10

* The texturing step failed, leading to fragmented surfaces and poor color reproduction.

Key takeaway: Across tools on Citadel, the sweet spot is 50–100 images. Polycam peaks at 50, Kiri Engine levels off at 100–200, and Meshroom degrades beyond 50 and fails texturing at 200. Thus, more input increases coverage but not fidelity and can destabilize some pipelines.

20 images

50 images

100 images

200 images

Figure 7.9: Kiri Engine photogrammetry-based reconstructions of the *Citadel* dataset.

7.2 3D GAUSSIAN SPLATTING MODELS RESULTS

Mirroring the photogrammetry analysis, we evaluate 3DGS on the same dataset categories (Everyday Scenarios, Edge-Case, and the *Citadel* robustness series). Tool-based 3DGS pipelines (*Kiri Engine*, *Polycam*, *PostShot*) are assessed *qualitatively* using the visual rubric from Section 6.3.2, whereas the official 3DGS source-code implementation is evaluated *quantitatively* with the metrics defined in Section 6.3.1 (PSNR, SSIM, LPIPS).

7.2.1 *Feasibility Tests on Edge-Case Datasets (3DGS)*

Analogous to the photogrammetry evaluation (Section 7.1.1), we test whether 3DGS can produce coherent, recognizable reconstructions under non-ideal

20 images

50 images

100 images

200 images

Figure 7.10: Meshroom photogrammetry-based reconstructions of the *Citadel* dataset.

conditions. The same *Edge-Case* datasets probe robustness to motion, unstructured geometry, and challenging illumination. Tool-based pipelines (*Kiri Engine*, *Polycam*, *PostShot*) are qualitatively assessed using the feasibility criteria in Appendix .4 (binary: Yes/No), while the official 3DGS source-code models are additionally reported with quantitative metrics (PSNR, SSIM, LPIPS; Table 7.16) to contextualize visual findings.

7.2.1.1 *Owl Dataset (3DGS)*

Table 7.11 summarizes 3DGS feasibility on the *Owl* dataset (Figure 7.11). Among tool-based pipelines, only *Polycam* yields a partially recognizable model whose overall shape persists under rotation, although with distortions and incomplete geometry. *Kiri Engine* and *PostShot* lose structural co-

herence in rotation, breaking into ghosting or fragmented point distributions. The official source-code model achieves SSIM = 0.841 in static re-renders, but the accompanying PSNR = 20.66 and LPIPS = 0.376 indicate limited perceptual fidelity; blurring and deformations become apparent when rotating (cf. Table 7.16).

Table 7.11: Feasibility results for the *Owl* dataset (3DGS).

3DGS Tool	Feasibility
Kiri Engine	No
Polycam	Yes
PostShot	No

Kiri Engine Owl 3DGS Model

Polycam *Owl* 3DGS Model

PostShot *Owl* 3DGS Model

3DGS Source Code *Owl* Model

Figure 7.11: 3DGS reconstructions of the *Owl* (Eule100) dataset created with four pipelines (top to bottom): Kiri Engine, Polycam, PostShot, and the 3DGS source code. The toy owl was recorded on a rotating turntable to obtain 360° coverage.

7.2.1.2 *Car Ride Dataset (3DGS)*

We evaluated two variants: a model trained directly from the original video (Figure 7.13) and a model trained from ∼100 extracted frames (Figure 7.12). As summarized in Table 7.12, only the *Kiri Engine* video-based model met the feasibility criterion (recognizable scene over time). All other pipelines either reconstructed only the last frames or failed entirely (*PostShot* aborted during camera tracking). Across reconstructions, motion blur and ghosting were prominent and limited visual plausibility.

Quantitatively, the source-code models attained SSIM = 0.656, PSNR = 16.52, LPIPS = 0.397 at 100 frames and SSIM = 0.583, PSNR = 16.27, LPIPS = 0.376 at 200 frames (Table 7.16), indicating low pixel-wise fidelity and weak structural consistency for this high-motion sequence.

Table 7.12: Feasibility results for the *Car Ride* dataset variants (3DGS).

3DGS Tool	**Car Ride 100**	**Car Ride Video**
Kiri Engine	No	Yes
Polycam	No	No
PostShot	No	No

7.2.1.3 *Forest 360-Degree Dataset (3DGS)*

Table 7.13 reports feasibility on the *Forest 360-Degree Shot* dataset. Only *Kiri Engine* produced a feasible 360° reconstruction: the scene is clearly recognizable, albeit with blur and local artifacts across vegetation and ground. *Polycam* yielded scattered green blobs amid large empty regions (no recognizable forest), and *PostShot* aborted during camera tracking.

The official source-code model reproduced only a subset of the rotation with noticeable artifacts; its metrics (SSIM=0.522, PSNR=18.08, LPIPS=0.300; Table 7.16) are consistent with limited structural consistency under repetitive, low-texture conditions.

Key takeaway: among 3DGS variants, only Kiri Engine handled the 360° forest; repetitive, low-texture content remains failure-prone elsewhere.

Table 7.13: Feasibility results for the *Forest 360-Degree* dataset (3DGS).

3DGS Tool	**Feasibility**
Kiri Engine	Yes
Polycam	No
PostShot	No

7.2.1.4 *Walkway Dataset (3DGS)*

Table 7.14 and Figure 7.15 show that the three tools produced feasible 3DGS reconstructions for the *Walkway* dataset. *PostShot* yielded the most stable

Kiri Engine *Car Ride* (100 frames) 3DGS model

Polycam *Car Ride* (100 frames) 3DGS model

Source code *Car Ride* (100 frames) 3DGS model

Figure 7.12: 3DGS reconstructions of the *Car Ride (100 frames)* variant. From top to bottom: Kiri Engine, Polycam, and 3DGS source code. Frames were extracted from a short car ride around Schloss Philippsruhe and are shown in capture order.

Figure 7.13: 3DGS reconstruction of the *Car Ride (video)* variant with **Kiri Engine**. The video was recorded from the passenger seat at ∼30–50 km/h. Frames are shown in chronological order (left to right, top to bottom).

result, but limited navigation to small moves at the place (up / down / left / right). In contrast, *Kiri Engine* and *Polycam* supported zoom-based navigation

Kiri Engine *Forest 360-Degree Shot* 3DGS Models

Polycam *Forest 360-Degree Shot* 3DGS Models

3DGS Source Code *Forest 360-Degree Shot* Models

Figure 7.14: 3DGS reconstructions of the *Forest 360-Degree Shot* dataset. Rows: Kiri Engine (top), Polycam (middle), 3DGS source code (bottom). Within each row, frames are ordered left to right.

that creates a pseudo-forward motion. All reconstructions exhibit motion blur from passing vehicles and minor geometric artifacts; the *Kiri Engine* model additionally shows translucent/sparse geometry.

The source code model achieved SSIM = 0.877, PSNR = 27.13, and LPIPS = 0.137 (Table 7.16), indicating a comparatively high structural precision among the edge case datasets, although localized artifacts and incomplete geometry persist along the walkway.

Key takeaway: all tools were feasible on Walkway; PostShot prioritized stability, while Kiri/Polycam offered more navigability, and motion blur remained the main limiter.

Table 7.14: Feasibility results for the *Walkway* dataset (3DGS).

3DGS Tool	**Feasibility**
Kiri Engine	Yes
Polycam	Yes
PostShot	Yes

Kiri Engine *Walkway* 3DGS Models

Polycam *Walkway* 3DGS Models

PostShot *Walkway* 3DGS Models

3DGS Source Code *Walkway* Models

Figure 7.15: 3DGS reconstructions of the *Walkway* dataset. Recordings were taken while walking with mostly forward-facing camera direction. Rows: Kiri Engine (top), Polycam, PostShot, 3DGS source code (bottom).

7.2.1.5 *Castle Compound (3DGS)*

Table 7.15 and Figure 7.16 show that all three 3DGS-based tools produced feasible reconstructions of the *Castle Compound* dataset, capturing the main volumes of the building. However, all models exhibit ghost structures and spatial inconsistencies. Both *Polycam* and *PostShot* show a characteristic duplication: front and rear facades appear side by side instead of in opposing positions. *Kiri Engine* shows a different ghosting pattern with partially merged and distorted sections.

The source-code model achieved SSIM = 0.603, PSNR = 17.76, and LPIPS = 0.410 (Table 7.16), consistent with the observed duplication artifacts and limited perceptual quality.

Key takeaway: feasible across tools, but multi-segment capture causes facade duplication/ghosting and breaks geometric coherence.

Table 7.15: Feasibility results for the *Castle Compound* dataset (3DGS).

3DGS Tool	Feasibility
Kiri Engine	Yes
Polycam	Yes
PostShot	Yes

Table 7.16: Quantitative evaluation of 3DGS models for *Edge-Case* datasets.

Dataset	Training Duration (min)	SSIM	PSNR (dB)	LPIPS
Owl	17	0.841	20.66	0.376
Car Ride	24	0.656	16.52	0.397
Car Ride 200	27	0.583	16.27	0.376
Forest 360-Degree Shot	32	0.522	18.08	0.300
Walkway	36	0.877	27.13	0.137
Castle Compound	23	0.603	17.76	0.410

Overall, the metrics in Table 7.16 mirror the feasibility findings: 3DGS is most stable on *Walkway* (highest SSIM/PSNR, lowest LPIPS) and weakest on *Car Ride* and *Castle Compound*, consistent with motion- and multi-segment artifacts. In line with the photogrammetry feasibility (Section 7.1.1), both paradigms struggle with fast motion and repetitive vegetation, but with method-specific failure modes (ghosting/duplication for 3DGS vs. fragmented/blurred meshes for photogrammetry).

7.2.2 *3DGS Models from Everyday Scenarios Datasets*

In parallel to the photogrammetry analysis (Section 7.1.2), this section evaluates 3DGS reconstructions on the *Everyday Scenarios* datasets. Tool-based 3DGS models (*Kiri Engine*, *Polycam*, *PostShot*) are assessed qualitatively using the visual rubric from Section 6.3.2 (Table 6.3), while models trained with the official 3DGS source code are evaluated quantitatively using PSNR, SSIM, and LPIPS. This mirrors the overall evaluation design and enables direct comparison with photogrammetry.

7.2.2.1 *Castle Frontside (3DGS)*

Table 7.17 and Figure 7.17 compare 3DGS reconstructions of Schloss Philippsruhe's front side created with *Kiri Engine*, *Polycam*, *PostShot* and the official source code. *Kiri Engine* and *Polycam* produce the sharpest results with a clear facade geometry. *PostShot* shows stronger artifacts and color inconsistencies. Across the tools, minor roof distortions and facade irregularities are

Kiri Engine *Castle Compound* 3DGS Model

Polycam *Castle Compound* 3DGS Model

PostShot *Castle Compound* 3DGS Model

3DGS Source Code *Castle Compound* Model

Figure 7.16: 3DGS reconstructions of the *Castle Compound* dataset with four tools: Kiri Engine, Polycam, PostShot, and the 3DGS source code.

visible, e.g., a slight haze in *Kiri Engine*, a dark spot in *Polycam*, and green patches in *PostShot*, but the overall structure of the building is preserved. The paired top views in Figure 7.17 (right column of each row) reveal slight transparency in the *Kiri Engine* model, while *Polycam* maintains a more stable opacity. The source code model achieves SSIM $= 0.914$, PSNR $= 30.46$,

and LPIPS = 0.111 (Table 7.19), indicating high structural precision and perceptual quality with only minor surface artifacts.

Table 7.17: Visual evaluation of Castle Frontside 3DGS reconstructions (Kiri Engine, Polycam, PostShot).

Castle Frontside 3DGS Model	Detail pres.	Completeness	Distortions	Noise	Color fidelity	Resolution	Consistency	Total Points
Kiri Engine	4	3	3	3	4	5	2	24
Polycam	4	4	4	3	5	5	3	28
PostShot	4	3	2	2	5	5	2	23

Kiri Engine *Castle Frontside* 3DGS model and top view (on the right).

Polycam *Castle Frontside* 3DGS model and top view (on the right).

PostShot *Castle Frontside* 3DGS model and top view (on the right).

3DGS Source Code *Castle Frontside* model and top view (on the right).

Figure 7.17: 3DGS reconstructions of Schloss Philippsruhe (front side) with matching top views for each tool.

7.2.2.2 *Bush (3DGS)*

As shown in Table 7.18 and Figure 7.18, *Polycam* delivers the strongest bush reconstruction, with sharp details, good completeness, and natural color fidelity. *Kiri Engine* performs solidly but shows slightly weaker consistency and minor geometric gaps in dense foliage. In contrast, *PostShot* fails to produce a stable model, exhibiting strong artifacts and fragmentation despite acceptable color reproduction.

The official 3DGS source code model reaches SSIM = 0.917, PSNR = 26.34 and LPIPS = 0.163 (Table 7.19). It reproduces the overall bush shape with consistent colors and stable geometry. However, fine leaf structures remain smoothed, and localized artifacts appear in the densest regions. In general, the result of the source code closely matches the quality of *Polycam* and *Kiri Engine*.

Table 7.18: Visual evaluation of 3DGS reconstructions of the *Bush* dataset.

Tool / Model	**Detail pres.**	**Completeness**	**Distortions**	**Noise**	**Color fidelity**	**Resolution**	**Consistency**	**Total Points**
Kiri Engine	4	3	3	3	3	4	3	23
Polycam	5	4	4	4	5	4	4	30
PostShot	1	1	1	1	3	1	1	9

Table 7.19: Quantitative evaluation of 3DGS Source Code models for *Everyday Scenario* datasets.

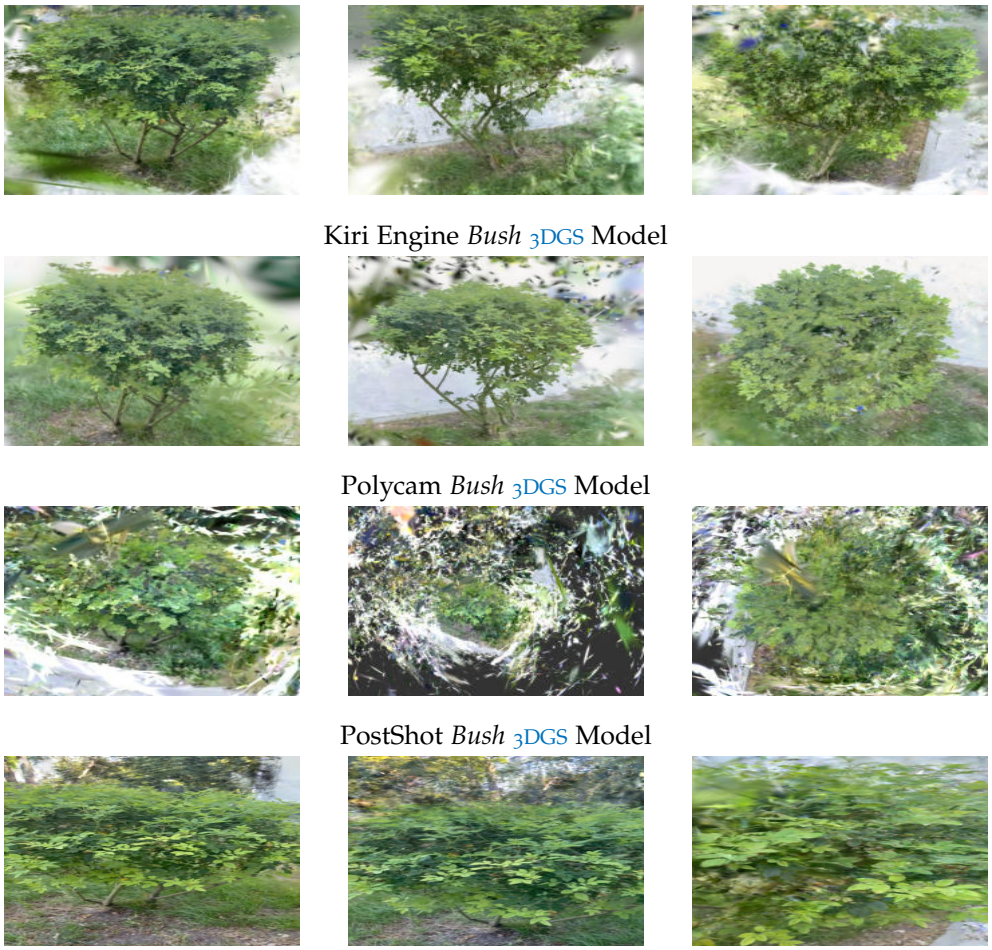
Dataset	**Training Duration (min)**	**SSIM**	**PSNR (dB)**	**LPIPS**
Bush	24	0.917	26.34	0.163
Castle Frontside	52	0.914	30.46	0.111

7.2.3 *Robustness Tests of 3DGS Citadel Models*

This section investigates the robustness of 3DGS on the *Citadel* dataset by varying the number of input images (20, 50, 100, 200). Tool-based reconstructions (*Kiri Engine*, *Polycam*, *PostShot*) are qualitatively assessed using the visual criteria of Section 6.3.2, while models trained with the official source code are quantitatively evaluated with PSNR, SSIM and LPIPS. The analysis focuses on how the count of the images affects the completeness, geometric consistency, and color fidelity.

7.2.3.1 *Citadel Dataset (Polycam – 3DGS)*

Table 7.20 and Figure 7.19 summarize the Polycam 3DGS reconstructions of the *Citadel* dataset. Across all input sizes (20, 50, 100, 200), the models exhibit consistently high visual quality: sharp details, complete geometry, sta-



Kiri Engine *Bush* 3DGS Model

Polycam *Bush* 3DGS Model

PostShot *Bush* 3DGS Model

3DGS Source Code *Bush* Models

Figure 7.18: 3DGS reconstructions of the *Bush* dataset (top to bottom: Kiri Engine, Polycam, PostShot, 3DGS Source Code). All methods use the same 360° capture for direct visual comparison.

ble consistency, and natural color reproduction. No material differences were observed between the four variants, indicating robustness to the number of input images.

Table 7.20: Visual evaluation of Polycam 3DGS reconstructions for the *Citadel* dataset (20–200 images).

Polycam 3DGS / Images	**Detail pres.**	**Complete-ness**	**Distortions**	**Noise**	**Color fidelity**	**Reso-lution**	**Consis-tency**	**Total Points**
20 images	5	5	5	5	5	5	5	35
50 images	5	5	5	5	5	5	5	35
100 images	5	5	5	5	5	5	5	35
200 images	5	5	5	5	5	5	5	35

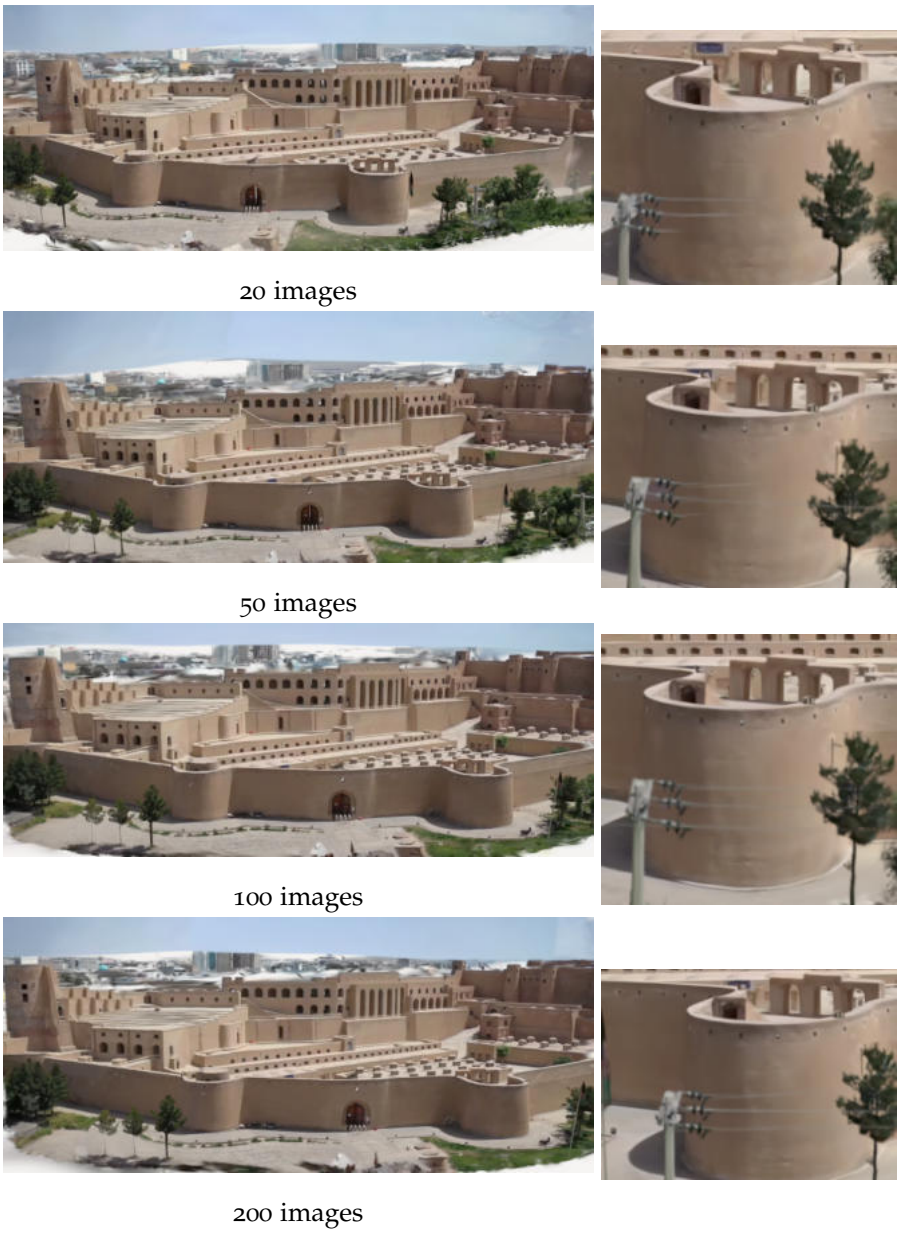


Figure 7.19: 3DGS reconstructions of the *Citadel* dataset using the **Polycam** app. Left: full model (20, 50, 100, 200 images from top to bottom). Right: corresponding detail crop of the electricity pylon in front of the fortress wall.

7.2.3.2 *Citadel Dataset (Kiri Engine – 3DGS)*

Table 7.21 and Figure 7.20 present the *Kiri Engine* 3DGS reconstructions of the *Citadel* dataset. Reconstruction quality scales strongly with input size: with 20 images, the model exhibits low opacity during rotation, left-side distortions, and dark veil-like artifacts. At 50–100 images, these artifacts diminish, geometry stabilizes, and wall edges sharpen; color reproduction becomes more natural. Using 200 images yields the most complete and consistent result overall, with improved opacity and well-preserved details. In contrast

to *Polycam* (Section 7.2.3.1), Kiri Engine benefits markedly from larger input sets.

Table 7.21: Visual evaluation of Kiri Engine 3DGS reconstructions for the *Citadel* dataset (20–200 images).

Kiri Engine 3DGS / Images	**Detail pres.**	**Complete-ness**	**Distortions**	**Noise**	**Color fidelity**	**Reso-lution**	**Consis-tency**	**Total Points**
20 images	2	2	2	2	2	3	2	15
50 images	3	3	3	3	3	4	3	22
100 images	4	4	3	3	4	4	3	25
200 images	5	5	4	4	5	4	4	31

20 images

50 images

100 images

200 images

Figure 7.20: 3DGS reconstructions of the *Citadel* dataset using the **Kiri Engine** app. From top to bottom: 20, 50, 100, and 200 input images.

7.2.3.3 *Citadel Dataset (PostShot – 3DGS)*

Table 7.22 and Figure 7.21 show the *PostShot* 3DGS reconstructions of the *Citadel* dataset. Across all data sets sizes, PostShot delivers robust and coherent models. With only 20 images, the reconstruction is already sharp and stable, showing only minor artifacts on the left facade and in the sky. Increasing to 50 and 100 images further improves completeness and sharpness. At 200 images, PostShot achieves its best result with high completeness, minimal distortions, and strong texture reproduction. Compared to *Kiri Engine* (Section 7.2.3.2), PostShot attains high quality with fewer inputs; relative to *Polycam* (Section 7.2.3.1), it trails slightly at very small input sizes, but remains consistently reliable.

Table 7.22: Visual evaluation of PostShot 3DGS reconstructions for the *Citadel* dataset (20–200 images).

PostShot / Images	**Detail pres.**	**Completeness**	**Distortions**	**Noise**	**Color fidelity**	**Resolution**	**Consistency**	**Total Points**
20 images	4	4	3	3	5	4	4	27
50 images	5	4	4	4	5	4	5	31
100 images	5	5	4	4	5	5	5	33
200 images	5	5	5	5	5	5	5	35

7.2.3.4 *Citadel Dataset (Source Code – 3DGS)*

Table 7.23 and Figure 7.22 summarize the quantitative evaluation of the official 3DGS source-code reconstructions of the *Citadel* dataset. As the input size increases from 20 to 100 images, SSIM and PSNR rise markedly while LPIPS decreases; the best values are reached with 200 images (SSIM $= 0.965$, PSNR $= 34.09$, LPIPS $= 0.091$), indicating that the implementation scales well with additional coverage and yields highly consistent, detailed reconstructions. Across all evaluated datasets, the Citadel series has the highest metric scores,reflecting not only the capacity of the method, but also favorable capture conditions (smooth drone motion, uniform illumination, limited parallax).

Table 7.23: Quantitative evaluation of 3DGS Source Codemodels for the *Citadel* reference series.

Dataset	**Training Duration (min)**	**SSIM**	**PSNR (dB)**	**LPIPS**
Citadel20	111	0.917	26.34	0.163
Citadel50	130	0.956	31.70	0.129
Citadel100	137	0.965	32.87	0.118
Citadel200	54	0.965	34.09	0.091

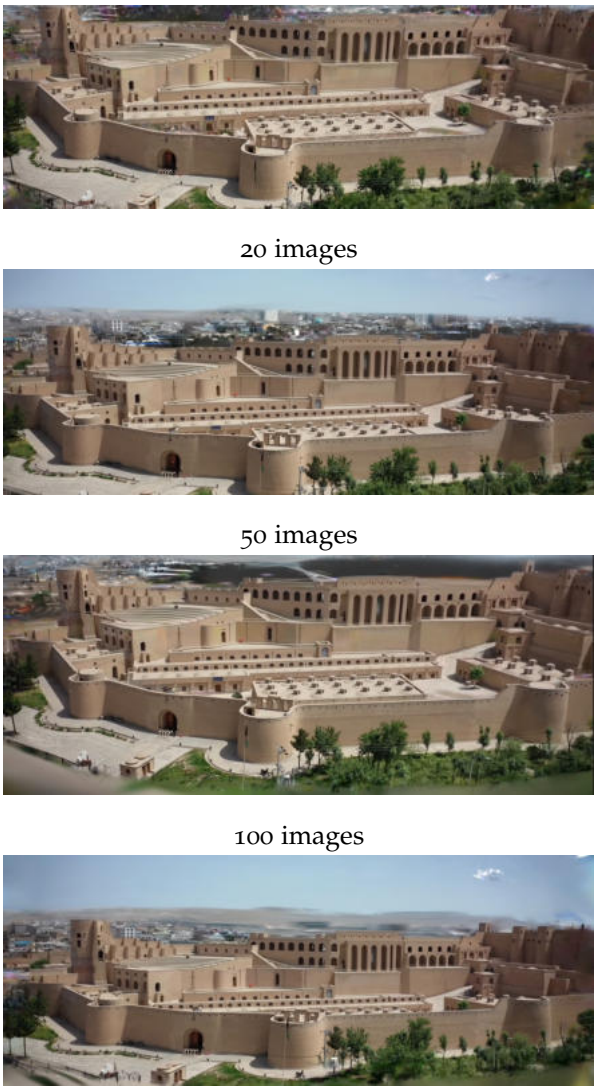


Figure 7.21: 3DGS reconstructions of the *Citadel* dataset using **PostShot**. From top to bottom: 20, 50, 100, and 200 input images.

Key takeaway: Polycam exhibits high robustness for 3DGS even with few input images; PostShot improves steadily and reaches peak performance at 200 images; Kiri Engine requires substantially larger input sets to achieve high quality; and the official source-code variant scales monotonically, achieving the best metrics at 200 images.

7.3 CROSS-METHOD COMPARISON

This section synthesizes and contrasts photogrammetry and 3DGS across the evaluated datasets (Everyday Scenarios, Edge Cases, and the *Citadel* robustness series). We compare feasibility, visual quality, and quantitative metrics (where pose-calibrated renders are available), and analyze sensitivity to the

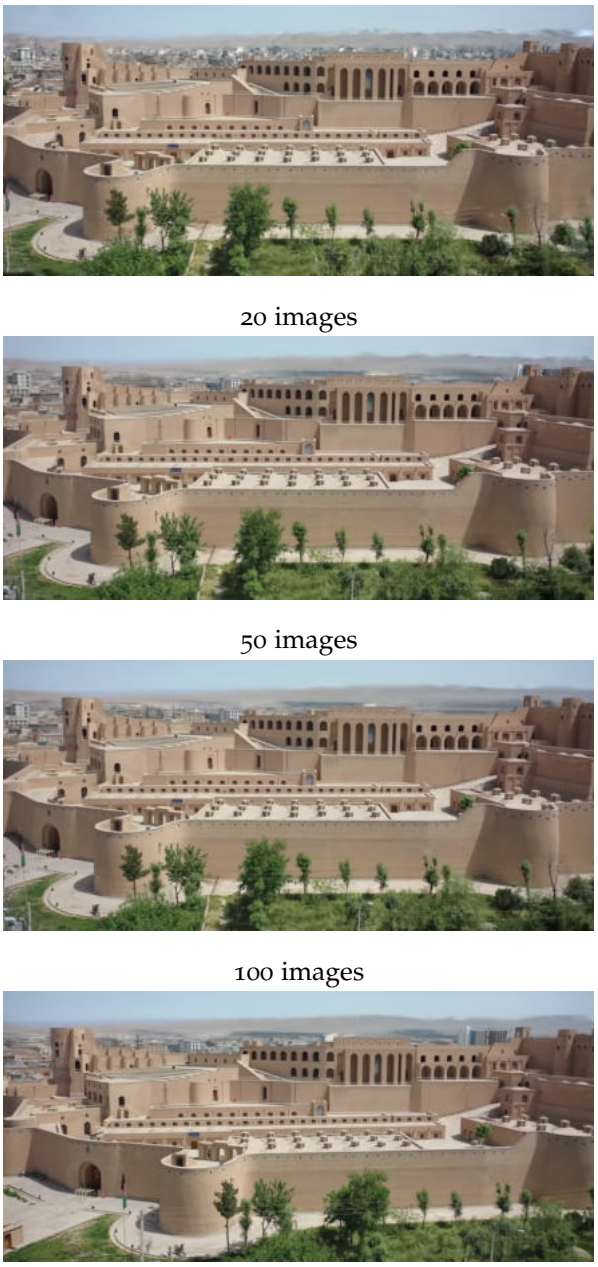


Figure 7.22: 3DGS reconstructions of the *Citadel* dataset using the **Source Code** implementation. From left to right: 20, 50, 100, and 200 input images.

number of input images. The aim is to distill method-specific strengths, failure modes, and application contexts without restating per-dataset details.

PHOTOGRAMMETRY.

Photogrammetry proved effective for producing metrically consistent, watertight meshes in structured, static scenes. This is illustrated by the *Owl* dataset, where a coherent object reconstruction was obtained under a controlled capture setup, and by the *Castle Frontside* dataset, which preserved

the overall architectural geometry (cf. Figure 7.6). Under favorable recording conditions, such as smooth drone flight and stable illumination in the *Citadel* series, mobile tools showed robust behavior for moderate to large inputs. However, performance was not strictly monotonic with the image count. *Polycam* tended to peak around 50–100 images, *Kiri Engine* benefited from larger sets, and *Meshroom* degraded at 200 images due to texturing failure.

Characteristic weaknesses emerged under less ideal conditions (Section 4.3). In scenes with limited parallax, repetitive textures, or motion—such as *Forest 360°* and *Car Ride*—reconstructions fragmented or failed. Lighting/exposure variations further reduced robustness, producing noise and texture seams as observed in *Walkway* and *Bush* (cf. Figures 7.4, 7.7). In *Bush*, fine vegetation (e.g., leaves, twigs) collapsed into dense, clustered surfaces, which is an expected outcome of classical MVS pipelines (Section 4.1.2), as they imposes local smoothness in depth and meshing.

For *Walkway*, the result was essentially an island of objects. The sidewalk was reconstructed coherently, but the adjacent road and the slope were omitted. This reflects an intrinsic bias of object-centric photogrammetry toward closed, self-contained meshes rather than open, spatially continuous environments. Photogrammetry generally performs well in structured, stable settings, but struggles with unstructured geometry, fine vegetation, motion, and inconsistent illumination. This is consistent with the scene- and algorithm-dependent constraints discussed in Section 4.3.

3D GAUSSIAN SPLATTING.

The 3DGS prioritizes perceptual realism and immersive view-dependent scene representations over strict metric accuracy. It performed particularly well in continuous environment-level captures, such as *Forest 360°*, *Walkway*, and *Car Ride*, producing coherent reconstructions with strong depth cues and illumination continuity. Compared to photogrammetry, the 3DGS results for the *Walkway* and *Car Ride* exhibited a visibly higher visual stability and completeness. Photogrammetry was often fragmented or failed in these settings.

The quantitative results from the source code models (Section 6.3.1) confirm solid, though not perfect, numerical fidelity. For *Walkway*, we measured PSNR = 27.13 dB, SSIM = 0.877, and LPIPS = 0.137. For *Car Ride*, the 100-image variant yielded PSNR = 16.52 dB, SSIM = 0.656, LPIPS = 0.397. Densifying to 200 images produced mixed results (PSNR = 16.27 dB, SSIM = 0.583, LPIPS = 0.376), suggesting that additional frames offer no consistent benefit without parameter tuning. These results highlight 3DGS's sensitivity to input coverage and training configuration. Proprietary pipelines (e.g., *Kiri Engine*) likely use tuned hyperparameters, which could explain their superior performance in challenging scenarios such as *Forest 360°* and *Car Ride Video*.

A recurring pattern is that quality peaks near the camera trajectory and degrades toward the periphery. In *Castle Frontside*, for instance, the central

facade is clear, while the outer roof regions blur or distort (cf. Figure 7.17). Similarly, in *Citadel* the nearby fortress is detailed, while distant areas are underrepresented. These observed weaknesses align with the known limitations discussed in (Section 7.5): limited parallax in object-centric setups (*Owl*) causes fragmentation, overlapping segments (*Castle Compound*) induce ghosting, reflective / low-texture regions can produce incomplete opacity, and large scenes (e.g., *Citadel200*) are constrained by GPU memory. Overall, 3DGS excels at producing visually compelling, continuous reconstructions under well-planned capture, but remains sensitive to parallax, peripheral coverage, and hardware. With tuned settings and deliberate acquisition, its quantitative and perceptual performance can approach classical methods, making it especially suitable for real-time, immersive exploration.

IMPACT OF INPUT DATA QUALITY.

Both pipelines are highly sensitive to image quality, motion stability, parallax, and viewpoint coverage. The contrast between the drone sequences of *Citadel* and *Castle Frontside* illustrates this: the Citadel flight followed a smooth, systematic path with high overlap and uniform illumination, producing stable, detailed reconstructions for both photogrammetry and 3DGS. In contrast, the Castle Frontside recording contained abrupt motion and incomplete coverage, producing distortions and fragmented geometry across tools (cf. Sections 7.1.3 and 7.2.3). In short, the capture design and the recording protocol are at least as decisive as the algorithm chosen for the quality of the reconstruction.

SUMMARY.

Photogrammetry yields metrically accurate, watertight meshes and is therefore well suited to documentation, measurement, and downstream uses such as CAD integration or 3D printing. In contrast, 3DGS produces dynamic, view-dependent scene representations that prioritize perceptual realism and interactive exploration in real time. The methods are thus complementary: choose photogrammetry when metric fidelity and topology are paramount. Prefer 3DGS when immersion and visual plausibility are the goal. Across all datasets, capture quality (overlap, parallax, and exposure stability) was the dominant determinant of outcome quality, underscoring that careful data acquisition is a prerequisite for reliable reconstructions.

7.4 TOOL EVALUATION

Beyond reconstruction quality, the evaluated tools were compared with respect to processing speed, usability, scalability, hardware and software requirements, and costs. This comparison includes all photogrammetry- and 3DGS-based tools discussed in this thesis,*Meshroom*, *Kiri Engine*, *Polycam*, *Post-Shot*, and the official implementation of the 3DGS source code trained on AWS.

PROCESSING SPEED.

Mobile apps (*Polycam*, *Kiri Engine*) had the shortest turnaround time because they are cloud-based, typically 2–40 min (photogrammetry is generally faster than 3DGS). *Kiri Engine* had occasional delays due to queues. Desktop tools varied more. *Meshroom* ranged from ∼10 min (*Citadel20*) to ∼2.5 h (*Citadel200*), and *PostShot* from ∼32 min to ∼1 h 51 min. The official 3DGS source code on AWS typically trained in ∼20–120 min per run, depending on dataset size and resolution.

USABILITY AND WORKFLOW.

The mobile apps are highly automated (low barrier to entry) and support simple post-processing (e.g., removing artifacts) directly in-app. *Meshroom* and *PostShot* provide greater parameter control, but require more technical knowledge. The 3DGS source-code pipeline demands the most setup (GPU drivers, CUDA, CLI on AWS) and clearly targets advanced users.

MODEL QUALITY.

Photogrammetry: Polycam and *Kiri Engine* delivered robust results in *Bush* and *Castle Frontside*. *Meshroom* showed lower robustness overall, especially on *Citadel200*, where texturing failed. However, it often produced accurate, stable SfM skeletons. *3DGS: Polycam* was consistently stable, including *Citadel* with 20–200 images. *PostShot* improved noticeably with larger input sets. *Kiri Engine* required more images to reach high quality (see Sections 7.2.3.1–7.2.3.2). The official 3DGS implementation served as the quantitative reference. Direct metric comparisons to app/desktop outputs are not possible due to missing exportable poses.

WHERE MESHROOM TENDS TO FAIL (AND WHY).

Consistent with the pipeline description (Chapter 4), *Meshroom*'s SfM was generally stable (cf. Section 4.1.1). Most degradations arose in MVS, meshing, and texturing (Section 4.1.2, Section 4.1.3, Section 4.1.4). In *Owl* (cf. Figure 7.1), the sparse SfM structure resembled the expected geometry, while the final textured mesh showed local artifacts, patterns aligned with Section 4.3. Mobile services (*Kiri Engine* and *Polycam*) often yielded more robust dense geometry under similar conditions. Although their internal pipelines are proprietary (see Section 6.1), this robustness is a plausible, not verified, interpretation.[1]

SCALABILITY AND REQUIREMENTS.

Cloud apps scale independently of local hardware (*Kiri Engine Pro* up to 500 images, *Polycam Pro* up to 1000 images). *Meshroom* and *PostShot* depend on a local CUDA-enabled GPU and become VRAM-limited on large sets (e.g., *Citadel200* texturing failure). The 3DGS source-code variant requires at least ≥24 GB VRAM for stable training and was the only environment that allowed full quantitative evaluation (PSNR, SSIM, LPIPS).

1 Mobile services do not disclose their internal algorithms; the robustness interpretation is plausible but not confirmed.

COSTS.

Meshroom is open source and free. *PostShot* was free in its beta during testing. Mobile apps use a freemium model (photogrammetry available with input limits, and Pro plans lift limits). At the time of testing (July 2025), *Polycam Pro* cost approximately **USD 30/month** and **USD 220/year**[2], and *Kiri Engine Pro* USD 17.99/month or USD 79.99/year. The 3DGS source code is free, but AWS incurs infrastructure costs (e.g., `g5.2xlarge` ≈ USD 1.21/hour, with typical compute costs ranging 1–2.5 USD per run). Total costs increase once setup, transfers, and idle time are included.

ACCESSIBILITY AND BROADER CONTEXT.

Mobile apps offer the lowest entry barrier—no local setup, cloud-side processing—and add consumer-facing conveniences such as sharing (e.g., web links/QR codes, GLB/GLTF export) and AR features for on-device placement. Desktop and source-code pipelines provide finer control and reproducibility but require expertise and CUDA-class GPUs. Consequently, apps are well-suited to quick, low-overhead reconstructions, whereas desktop/source-code workflows target research and expert use.

7.5 LIMITATIONS

The scope of this study is subject to several limitations. First, quantitative metrics could only be applied to the 3DGS source-code models (see Section 6.3.1) because mobile and desktop tools do not expose calibrated poses. This mixed evaluation design restricts direct, tool-to-tool comparability. Moreover, PSNR, SSIM, and LPIPS quantify image similarity rather than geometric accuracy, and thus may not fully reflect metric fidelity or mesh watertightness.

Second, experiments were constrained by available compute (local: 32 GB RAM, RTX 2070 Ti; cloud: 24 GB VRAM), which affected scalability and required downscaling for large inputs (e.g., *Citadel200*). Input quality also mattered substantially: smoother, better-covered drone footage (*Citadel*) yielded more stable results than less systematic captures (*Castle Frontside*).

Third, internal pipelines of the mobile applications (*Kiri Engine*, *Polycam*) are proprietary and undocumented. Details of dense reconstruction, meshing, or texturing remain unknown, preventing a precise methodological comparison with open-source *Meshroom* or the official 3DGS implementation. Likewise, *PostShot* was evaluated in a beta version; performance may change after release. Cloud services further introduce version drift and potential non-determinism (dynamic models, queueing, server-side updates), which can limit strict reproducibility.

Fourth, feasibility and visual assessments are inherently subjective (see Section 6.4.2). Ratings were performed by a single rater using a 1–5 rubric, no inter-rater reliability or blinded pairwise tests were conducted. In addi-

2 Exact German pricing at the time of testing: **EUR 26.99/month** or **EUR 199.99/year**. Exchange rates and regional taxes may vary.

tion, most tools were run with default or minimally tuned parameters; neither systematic hyperparameter sweeps nor ablation studies (e.g., iterations, learning rates, filtering thresholds) were performed, which may bias results against methods that benefit from tuning.

Finally, this work focuses on reconstruction quality and runtime but does not include a user study, energy measurements, or a systematic cost analysis across providers. Consequently, practical usability at scale, long-term cost efficiency, and cross-cloud portability remain outside the present scope.

8

CONCLUSION

This thesis set out to compare classical photogrammetry and modern differentiable 3D reconstruction methods, particularly 3DGS, under limited hardware and practical conditions. The central goal was to determine how both approaches perform across different capture scenarios and levels of user expertise, and whether non-expert users can successfully produce coherent 3D models using accessible tools.

The results demonstrate that 3D reconstruction is feasible even without professional equipment or expert knowledge. Mobile applications such as *Kiri Engine* and *Polycam* proved especially effective in lowering the entry barrier, enabling the creation of high-quality 3D models directly on a smartphone. Photogrammetry showed particular strength in producing metrically accurate, watertight meshes for structured, static scenes, while 3DGS excelled in immersive scene reconstructions that prioritize visual realism and continuity. Together, these findings confirm that both paradigms complement each other: photogrammetry is best suited for precise documentation, whereas 3DGS is ideal for realistic visualization and experiential learning environments.

A key insight from this study is that image quality and recording design have a greater impact on reconstruction quality than the specific method itself. The consistent results of the *Citadel* datasets underline how stable illumination, sufficient parallax, and complete coverage enable strong reconstructions across all tools. Conversely, unstructured or dynamic datasets, such as *Walkway* and *Car Ride*, revealed the current limitations of both approaches.

Overall, the research objectives were achieved: the study provided a systematic, cross-method comparison, identified hardware-efficient pathways for 3D reconstruction, and demonstrated that even non-expert users can achieve convincing results using consumer-grade devices and cloud infrastructure.

Future work should address the specific challenges observed in this study. One direction is the development of methods capable of handling independent capture segments, as encountered in the *Castle Compound* dataset, where reconstructions from separate viewing angles failed to merge into a coherent model. Improving algorithms for cross-segment alignment and global consistency could enhance the reconstruction of large or partially overlapping structures such as buildings or monuments. Another important aspect concerns motion-related degradation: approaches that explicitly compensate for motion blur or rolling-shutter distortions could substantially increase robustness in handheld and video-based captures. Addressing these issues would significantly improve the reliability of 3D reconstruction methods in real-world, unconstrained scenarios.

Part II

APPENDIX

APPENDIX

.1 MATHEMATICAL DETAILS OF SPHERICAL HARMONICS IN 3DGS

Spherical Harmonics SH provide a complete orthogonal basis for representing functions defined over the unit sphere. They offer a compact representation for functions that vary with direction. Similar to Fourier series, which decompose periodic signals into frequency components, SH decompose functions on the sphere into components of increasing angular frequency. The degree l determines the complexity of the represented patterns:

- $l = 0$: constant, isotropic function.
- $l = 1$: dipole patterns, splitting the sphere into positive and negative hemispheres.
- $l = 2$: quadrupole structures with alternating regions.

.1.1 *Use of SH in Neural Rendering*

Modern neural rendering methods adopted SH to efficiently encode view-dependent effects:

- **Plenoxels (2022)** store SH coefficients directly per voxel, typically up to degree $L = 2$ (9 coefficients per channel). This allows computing color without a neural network [14].
- **InstantNGP (2022)** uses SH as a compact directional encoding. The viewing direction is projected onto the first 16 SH coefficients (up to degree $L = 4$), which are then concatenated with positional encodings and fed into a small MLP [47].

.1.2 *Application in 3D Gaussian Splatting*

3DGS extends this principle by associating SH coefficients with each Gaussian. The rendering pipeline for one Gaussian is as follows:

1. Compute the normalized viewing direction d.
2. Evaluate SH basis functions $Y_l^m(d)$ up to degree $L = 3$.
3. Combine basis values with learned coefficients k_l^m to obtain the color:

$$c(d) = \sum_{l=0}^{L} \sum_{m=-l}^{l} k_l^m \cdot Y_l^m(d).$$

4. Repeat per channel (RGB). The final result is mapped into $[0, 1]$ using a sigmoid activation.[1]

By default, $L = 3$ is used in 3DGS, which yields 16 coefficients per channel, or 48 coefficients per Gaussian. With millions of Gaussians, this becomes the primary storage and performance bottleneck.

.1.3 *Limitations and Advances*

The reliance on SH makes color modeling accurate but memory-intensive. Recent work such as SG-Splatting (2024) introduces *spherical Gaussians* (SGs) as a more compact alternative, reducing parameters to 7 per SG while maintaining visual fidelity [67]. Hybrid models combining low-degree SH with SGs are also explored.

.1.4 *Conclusion*

SH provide the mathematical foundation for view-dependent appearance in 3DGS. They represent a powerful but resource-demanding tool, motivating ongoing research into more efficient alternatives such as SG-Splatting.

.2 DATASET COLLAGES

This appendix-section provides an overview of all datasets used in this thesis. Each collage shows either all frames of the dataset or representative samples of the captured images. The collages serve as documentation of acquisition conditions and dataset characteristics.

.2.1 *Everyday Scenario Datasets*

- Bush (see Figure .1)
- Castle Fronstside (see Figure .2)

.2.2 *Edge Case Datasets*

- Owl (see Figure .3)
- Walkway (see Figure .4)
- Forest 360-Degree Shot (see Figure .5)
- Car Ride (see Figure .6)
- Castle Compound (see Figure .7)

1 In the reference implementation this is explicitly realized as `torch.sigmoid()`.

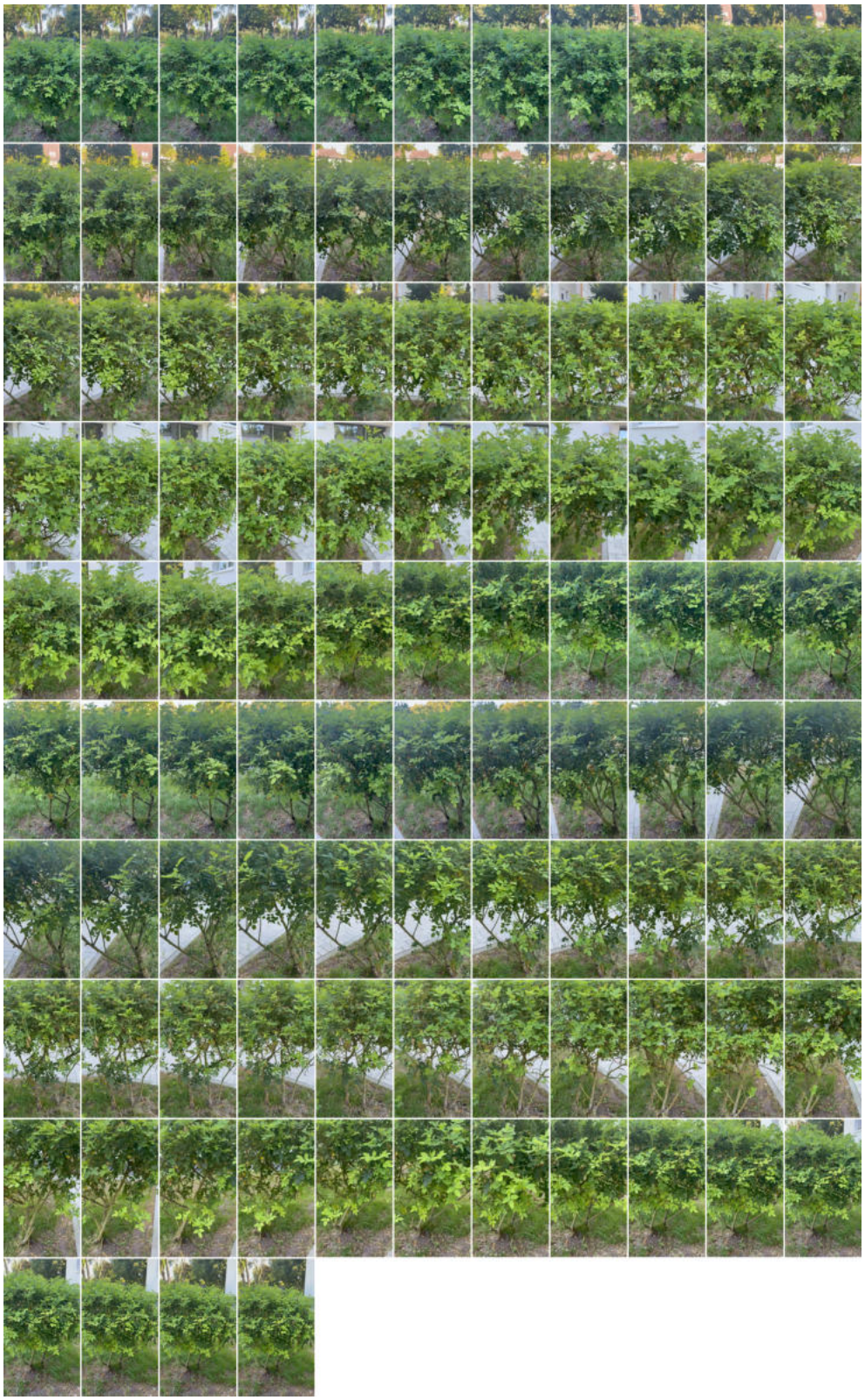


Figure .1: All frames of the **Bush** dataset as a collage (thumbnails).

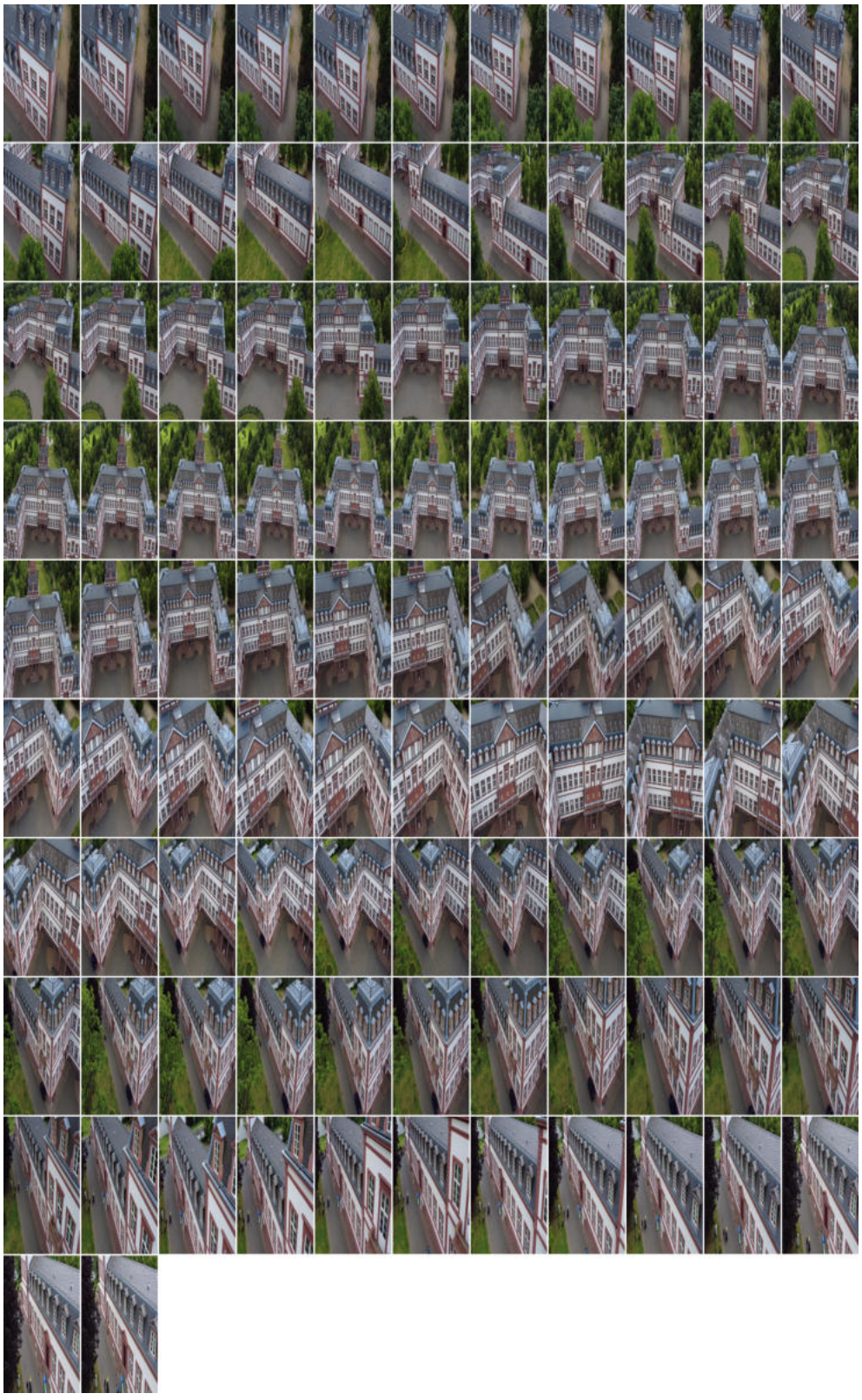


Figure .2: Overview of the captures in the **Castle Compound** dataset.

.2.3 *External Datasets*

- Citadel (see Figure .8)

.3 TOOL SPECIFICATIONS

This appendix-section provides detailed technical specifications of the tools included in the evaluation. While Chapter 6 introduces the tools in brief, the following tables give a comprehensive overview of supported devices, hardware and software requirements, input and output formats, and cost models.

.3.1 *Mobile Applications*

Table .1 summarizes the specifications of the three mobile apps considered in this thesis: Kiri Engine and Polycam,

Table .1: Comparison of mobile 3D reconstruction apps [27–31, 49–52].

Category	**Kiri Engine**	**Polycam (Pro)**
Apple device	from iPhone 12 Pro/Pro Max (LiDAR) or iPad Pro 2020+ (LiDAR)	from iPhone 12 Pro/Pro Max (LiDAR) or iPad Pro 11″ (2nd+), 12.9″ (4th+), 13″ (M4)
Android device	Android smartphones for photogrammetry (cloud-based)	Samsung Galaxy S20+, Google Pixel 6+; $\geq$ 3.5 GB RAM, 64-bit; no PowerVR GE8320
Desktop	Web application	Web application
Photogrammetry	Yes	Yes
3DGS	Yes (Pro plan only)	Yes (Pro plan only)
Input	20–100 photos (Basic) / 20–300 (Pro); Video: 1 min (Basic) / 3 min (Pro); Formats: JPG, PNG, JPEG / MP4, MOV	Photogrammetry: 20–2000 photos; Gaussian Splat: 20–1000 photos; Video: 00:15–16 GB; Formats: PNG, JPG / MP4, MOV, AVI, M4V
Costs (USD)	Free (Basic), Pro: $79.99/year ($6.66/month) or $17.99/month	Free (Basic), Pro: $17/month
Export formats	Photo Scan / Featureless Object Scan: OBJ, FBX, STL, GLB, GLTF, USDZ, PLY, XYZ; 3DGS: PLY (native) or OBJ, FBX, GLB, GLTF, USDZ, PLY, XYZ; LiDAR: OBJ, USDZ	MESH: OBJ, GLTF, FBX, DAE, STL, USDZ; POINT CLOUD: Splat PLY, PLY, LAS, PTS, XYZ, DXF; OTHER: Video, Blueprint, Images

.3.2 *Desktop and Server Solutions*

Table .2 provides a comparison of the evaluated desktop/server tools: Meshroom, PostShot, and the official 3DGS source code implementation.

Table .2: Comparison of selected desktop/server solutions for 3D reconstruction [16, 24, 44].

Category	Meshroom	PostShot	3DGS (GitHub)
Hardware requirements	NVIDIA GPU (compute capability $\geq$ 2.0), recommended: 16–32 GB RAM	Windows 8.1+, NVIDIA RTX 2060 (CC 7.5+)	CUDA-capable GPU (CC $\geq$ 7.0), 24 GB VRAM for paper-level quality
Software requirements	Windows x64, Linux, or macOS (limited); AliceVision framework integrated	Windows 8.1 or newer	Conda (recommended), C++ compiler for PyTorch extensions, CUDA SDK 11 (recommended: 11.8); compiler–CUDA compatibility required
Costs (USD)	Open-source	Beta, free of charge	see Section 7.4 for EC2 costs
Export formats	OBJ, PLY	–	Point cloud

.4 FEASIBILITY CRITERIA

This appendix-section defines Feasibility under which conditions a reconstruction is considered successful for each edge-case dataset. A model is regarded as *feasible* if it produces a coherent and recognizable 3D representation of the intended object or scene. The following dataset-specific criteria were applied consistently across all methods.

- **Owl (Object Capture):** Feasible if the owl's overall body shape and head are clearly recognizable from multiple viewpoints, even if fine textures or small details are missing.
- **Car Ride (Dynamic Scene):** Feasible if continuous geometry representing the roadside environment is reconstructed, allowing the car's path and adjacent objects (buildings, trees) to be recognized as part of a coherent scene rather than isolated fragments.
- **Forest 360-Degree Shot (Natural Environment):** Feasible if a 360° forest scene can be identified, with visible vegetation, trees, and ground structure forming a continuous environment rather than disconnected or empty regions.
- **Walkway (Outdoor Motion Scene):** Feasible if the walkway and surrounding vegetation are clearly recognizable as a pedestrian path scene, allowing basic spatial orientation and partial navigability within the reconstructed model.

- **Castle Compound (Large-Scale Combined Capture):** Feasible if the castle structure is reconstructed in a recognizable architectural form, including identifiable facades and an overall building layout, even if ghosting or duplicated elements occur.

Feasibility was evaluated qualitatively for photogrammetry- and tool-based 3DGS models (*Kiri Engine*, *Polycam*, *Meshroom*, *PostShot*) according to these definitions and rated in binary form (*Yes/No*).

The feasibility criteria are subjective in nature, as they rely on human judgment of recognizability and structural coherence rather than on quantitative measures. Moreover, the specific conditions under which a reconstruction is considered *feasible* were defined subjectively for this study, reflecting practical interpretation rather than an established or standardized framework.

Figure .3: Sample overview of the captures in the **Owl** dataset.

Figure .4: All frames of the **Walkway** dataset as a collage (thumbnails).

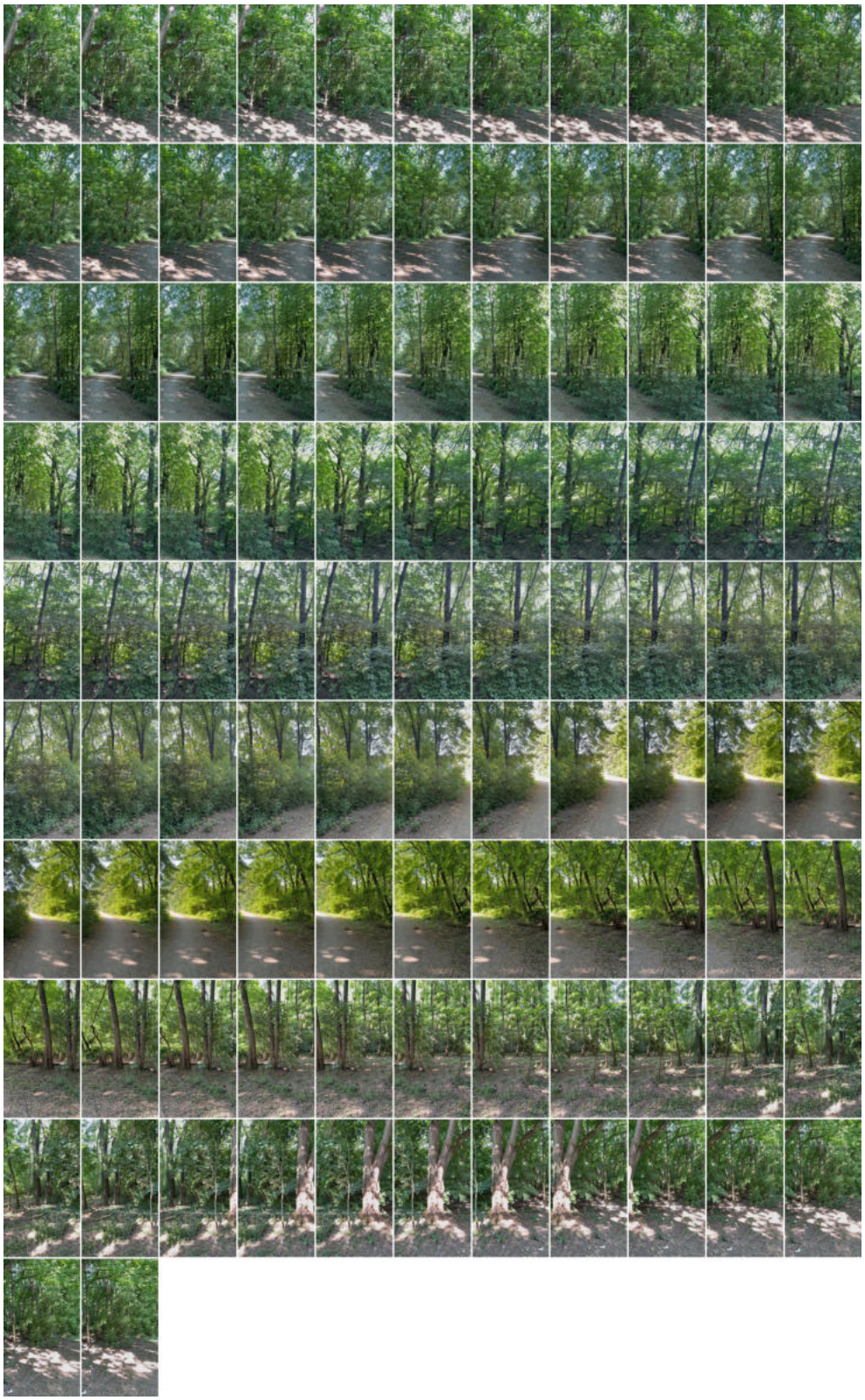


Figure .5: All frames of the **Forest 360-Degree Shot** dataset as a collage (thumbnails).

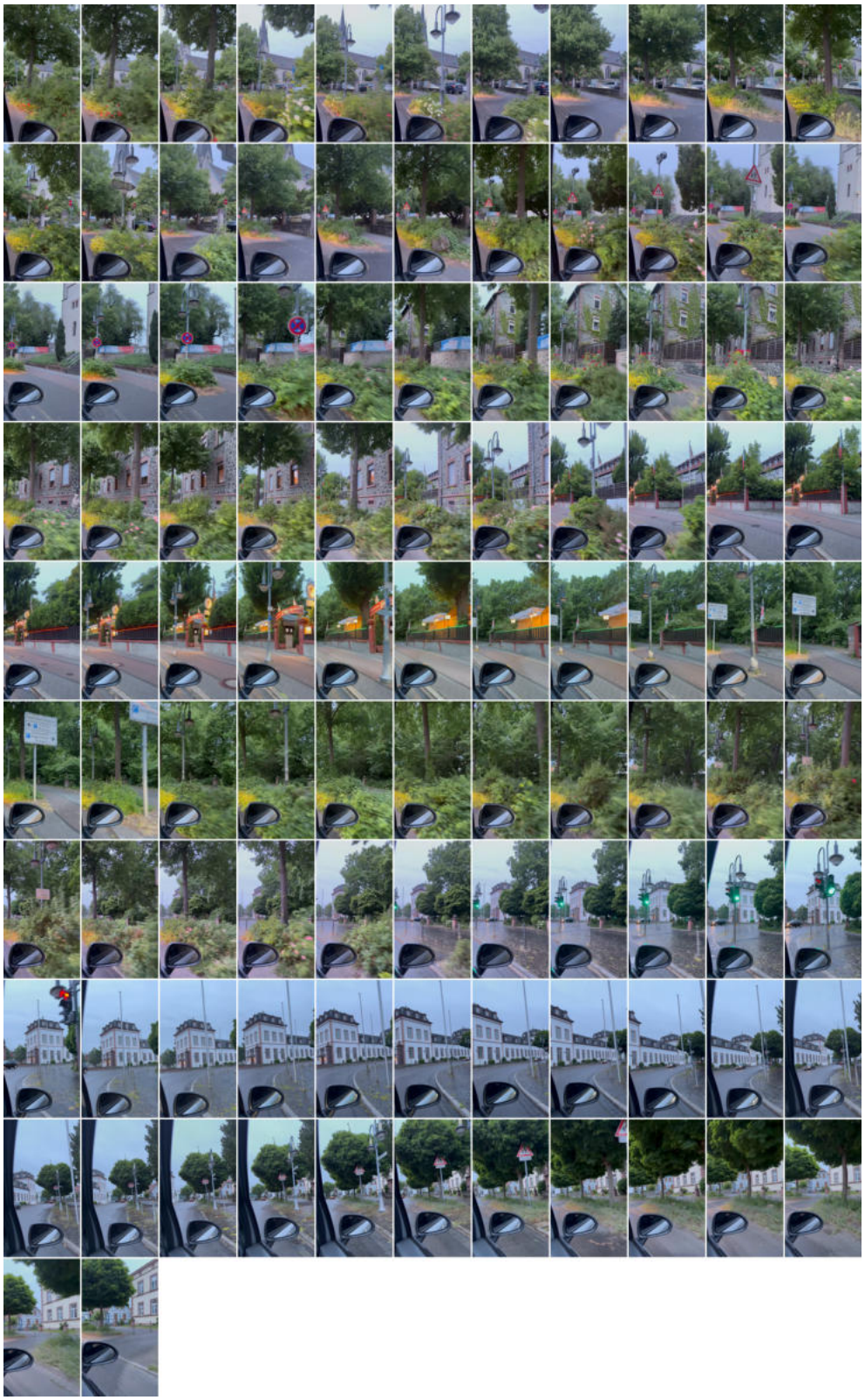


Figure .6: All frames of the **Car Ride** dataset as a collage (thumbnails).

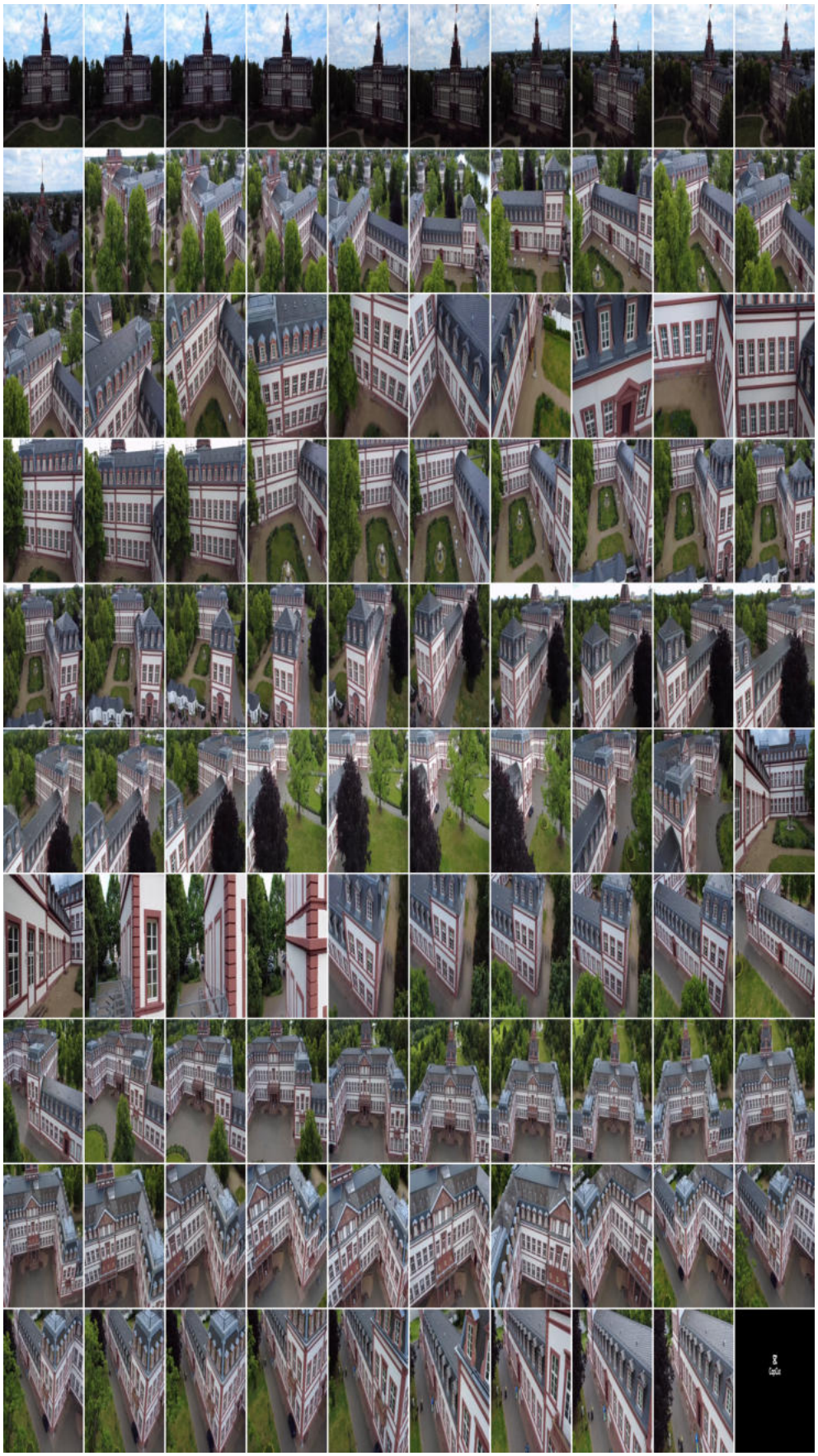


Figure .7: All frames of the **Castle Compound** dataset as a collage (thumbnails).

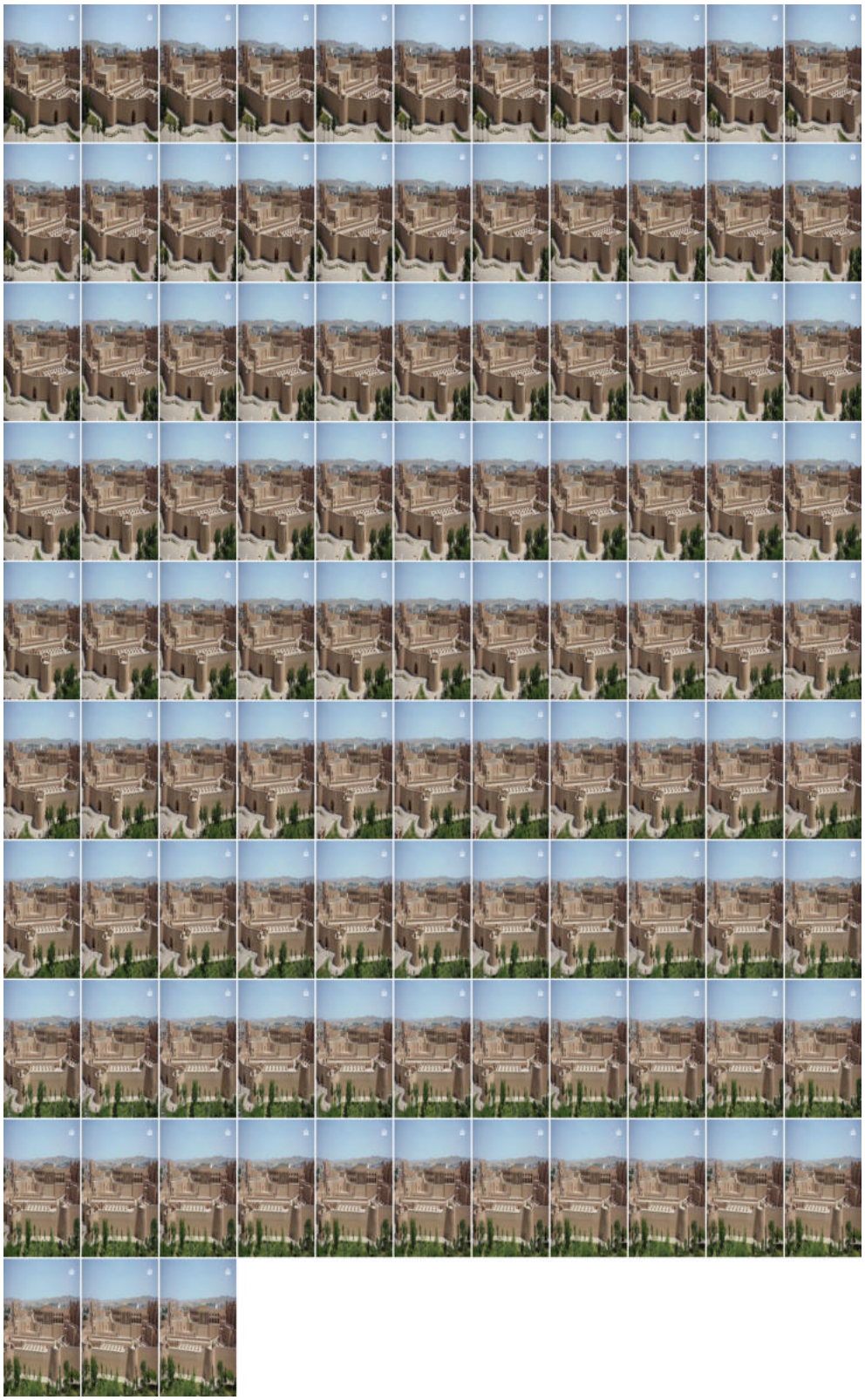


Figure .8: All frames of the **Citadel** dataset as a collage (thumbnails).

BIBLIOGRAPHY

[1] Sameer Agarwal, Noah Snavely, Steven M. Seitz, and Richard Szeliski. "Bundle Adjustment in the Large." In: *Proceedings of the 11th European Conference on Computer Vision (ECCV)*. Springer, 2010, pp. 29–42. DOI: 10.1007/978-3-642-15552-9_3.

[2] Pablo F. Alcantarilla, Jesús Nuevo, and Adrien Bartoli. "Fast Explicit Diffusion for Accelerated Features in Nonlinear Scale Spaces." In: *Proceedings of the British Machine Vision Conference (BMVC)*. BMVA Press, 2013, pp. 1–11. URL: https://bmvc2013.cs.nott.ac.uk/papers/029/paper029.pdf.

[3] AliceVision. *AliceVision – Photogrammetric Computer Vision Framework*. Zuletzt abgerufen am October 15, 2025. 2025. URL: https://alicevision.org/.

[4] Atalkhan. *Herat Citadel*. YouTube video, https://www.youtube.com/watch?v=vOsv8AD4224. Accessed: 2025-08-14. 2024.

[5] Adrian Azzarelli, Nantheera Anantrasirichai, and David R Bull. "Exploring Dynamic Novel View Synthesis Technologies for Cinematography." In: *arXiv preprint arXiv:2412.17532* (2024).

[6] Connelly Barnes, Eli Shechtman, Adam Finrbllstein, and Dan B. Goldman. "PatchMatch: A Randomized Correspondence Algorithm for Structural Image Editing." In: *ACM Transactions on Graphics* (2009).

[7] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. "SURF: Speeded Up Robust Features." In: *Computer Vision and Image Understanding* 110.3 (2008), pp. 346–359. DOI: 10.1016/j.cviu.2007.09.014.

[8] Michael Bleyer, Christoph Rhemann, and Carsten Rother. "PatchMatch Stereo – Stereo Matching with Slanted Support Windows." In: *Proceedings of the British Machine Vision Conference (BMVC)*. 2011.

[9] Frédéric Cazals and Joachim Giesen. *Delaunay Triangulation Based Surface Reconstruction: Ideas and Algorithms*. Research Report RR-5393. INRIA, 2004.

[10] Zhiqin Chen, Kangxue Yin, and Sanja Fidler. "AUV-Net: Learning Aligned UV Maps for Texture Transfer and Synthesis." In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2022, pp. 1455–1464. DOI: 10.1109/CVPR52688.2022.00152. URL: https://openaccess.thecvf.com/content/CVPR2022/papers/Chen_AUV-Net_Learning_Aligned_UV_Maps_for_Texture_Transfer_and_Synthesis_CVPR_2022_paper.pdf.

[11] F. Condorelli et al. "A Comparison Between 3D Reconstruction Using NeRF Neural Networks and MVS Algorithms on Cultural Heritage Images." In: *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*. Vol. XLIII-B2-2021. 2021, pp. 565–570. DOI: 10.5194/isprs-archives-XLIII-B2-2021-565-2021.

[12] Daniel Duckworth, Jonathon Luiten, Quoc-Huy Pham, Philippe Weinzaepfel, Jerome Revaud, Christian Rupprecht, and Bastian Leibe. "SMERF: Streamable Memory Efficient Radiance Fields for Real-Time Large-Scene Exploration." In: *arXiv preprint arXiv:2312.07541* (2023). URL: https://arxiv.org/abs/2312.07541.

[13] Martin A. Fischler and Robert C. Bolles. "Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography." In: *Communications of the ACM* 24.6 (1981), pp. 381–395.

[14] Sara Fridovich-Keil, Alex Yu, Matthew Tancik, Qinhong Chen, Ben Recht, Angjoo Kanazawa, and Ren Ng. "Plenoxels: Radiance Fields without Neural Networks." In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2022, pp. 5501–5510.

[15] *Future Learning Space. Homepage*. Accessed: 2025-01-21. 2025. URL: https://futurelearning.space/.

[16] GRAPHDECO-INRIA. *Gaussian-Splatting — GitHub Repository*. Zuletzt abgerufen am October 15, 2025. 2025. URL: https://github.com/graphdeco-inria/gaussian-splatting.

[17] X-S Gao, X-R Hou, J Tang, and H-F Cheng. "Complete solution classification for the perspective-three-point problem." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 25.8 (2003), pp. 930–943.

[18] L. Gomes et al. "3D reconstruction methods for digital preservation of cultural heritage: A survey." In: *Pattern Recognition Letters* 50 (2014), pp. 3–14. ISSN: 0167-8655. DOI: 10.1016/j.patrec.2014.03.023.

[19] Carsten Griwodz, Simone Gasparini, Lilian Calvet, Pierre Gurdjos, Fabien Castan, Benoit Maujean, Yann Lanthony, and Gregoire De Lillo. "AliceVision Meshroom: An open-source 3D reconstruction pipeline." In: *Proceedings of the 12th ACM Multimedia Systems Conference (MMSys '21)*. New York, NY, USA: Association for Computing Machinery, 2021, pp. 241–247. ISBN: 978-1-4503-8434-6. DOI: 10.1145/3458305.3478443. URL: https://doi.org/10.1145/3458305.3478443.

[20] Markus Gross, Hanspeter Pfister, Marc Alexa, Mark Pauly, Marc Stamminger, and Matthias Zwicker. *Point-Based Computer Graphics*. Eurographics 2002 Tutorial T6. Tutorial Notes. 2002.

[21] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2004.

[22] Heiko Hirschmüller. "Stereo Processing by Semiglobal Matching and Mutual Information." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 30.2 (2008), pp. 328–341. DOI: 10.1109/TPAMI.2007.1166.

[23] Zhangjin Huang, Yuxin Wen, Zihao Wang, Jinjuan Ren, and Kui Jia. *Surface Reconstruction from Point Clouds: A Survey and a Benchmark*. arXiv:2205.02413 [cs.CV], version 1, 5 May 2022. 2022. arXiv: 2205.02413 [cs.CV].

[24] Jawset Visual Computing. *Postshot User Guide Release Notes v0.1*. Zuletzt abgerufen am October 15, 2025. 2023. URL: https://www.jawset.com/docs/d/Postshot+User+Guide/Release+Notes/v0.1.

[25] Nianjuan Jiang, Zhaopeng Cui, and Ping Tan. "Global Structure-from-Motion Revisited." In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. 2013, pp. 1748–1755. DOI: 10.1109/ICCV.2013.220.

[26] Johannes L. Schönberger and Jan-Michael Frahm and the COLMAP community. *COLMAP*. General-purpose Structure-from-Motion and Multi-View Stereo pipeline with both GUI and CLI. 2025. URL: https://colmap.github.io/.

[27] KIRI Engine. *Export Formats – KIRI Engine*. Zuletzt abgerufen am October 15, 2025. 2025. URL: https://www.kiriengine.app/features/export-formats.

[28] KIRI Engine. *LiDAR Scan – KIRI Engine*. Zuletzt abgerufen am October 15, 2025. 2025. URL: https://www.kiriengine.app/features/lidar-scan.

[29] KIRI Engine. *Pricing – KIRI Engine*. Zuletzt abgerufen am October 15, 2025. 2025. URL: https://www.kiriengine.app/pricing.

[30] KIRI Innovations. *3D Gaussian Splatting Feature Overview*. Accessed: 2025-08-12. 2024. URL: https://www.kiriengine.app/features/3d-gaussian-splatting.

[31] KIRI Innovations. *Photo Scan Feature Overview*. Accessed: 2025-08-12. 2024. URL: https://www.kiriengine.app/features/photo-scan.

[32] Onur Keleş, M. Akın Yılmaz, A. Murat Tekalp, Cansu Korkmaz, and Zafer Doğan. "On the Computation of PSNR for a Set of Images or Video." In: *arXiv preprint arXiv:2104.14868* (2021).

[33] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. "3D Gaussian Splatting for Real-Time Radiance Field Rendering." In: *ACM Transactions on Graphics* 42.4 (2023).

[34] Jason Kottke. *Medieval illustrations of what Europeans thought elephants looked like*. Accessed: 2025-01-21. 2018. URL: https://kottke.org/18/01/medieval-illustrations-ofwhat-europeans-thought-elephants-looked-like.

[35] Joseph J. LaViola. "A discussion of cybersickness in virtual environments." In: *ACM SIGCHI Bulletin* 32.1 (2000), pp. 47–56. DOI: 10.1145/333329.333344.

[36] LearnOpenCV. *3D Gaussian Splatting: A New Era of Real-Time Radiance Field Rendering*. Accessed: 2025-09-05. 2023. URL: https://learnopencv.com/3d-gaussian-splatting/#Gaussian-Splatting-Limitations.

[37] Vincent Lepetit, Francesc Moreno-Noguer, and Pascal Fua. "EPnP: An accurate O(n) solution to the PnP problem." In: *IEEE Conference on Computer Vision and Pattern Recognition*. IEEE. 2009, pp. 1–8.

[38] Stefan Leutenegger, Margarita Chli, and Roland Y. Siegwart. "BRISK: Binary Robust Invariant Scalable Keypoints." In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. IEEE, 2011, pp. 2548–2555. DOI: 10.1109/ICCV.2011.6126542. URL: https://doi.org/10.1109/ICCV.2011.6126542.

[39] Bruno Lévy, Sylvain Petitjean, Nicolas Ray, and Jérome Maillot. "Least Squares Conformal Maps for Automatic Texture Atlas Generation." In: *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*. ACM, 2002, pp. 362–371.

[40] Haoang Li. *Computer Vision II: Multiple View Geometry (IN2228) — Chapter 12: Bundle Adjustment*. Lecture slides. Lecture on 6 July 2023. July 2023. URL: https://cvg.cit.tum.de/teaching/ss2023/cv2 (visited on 10/13/2025).

[41] David G. Lowe. "Distinctive Image Features from Scale-Invariant Keypoints." In: *International Journal of Computer Vision* 60.2 (2004), pp. 91–110. DOI: 10.1023/B:VISI.0000029664.99615.94.

[42] C.-P. Lu, G. D. Hager, and E. Mjolsness. "Fast and globally convergent pose estimation from video images." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22.6 (2000), pp. 610–622.

[43] Thomas Luhmann, Stuart Robson, Stephen Kyle, and Ian Harley. *Close-Range Photogrammetry and 3D Imaging*. 2nd ed. Berlin, Germany: De Gruyter, 2014.

[44] Meshroom Documentation. *Requirements — Meshroom*. Zuletzt abgerufen am October 15, 2025. 2025. URL: https://meshroom-manual.readthedocs.io/en/bibtex1/install/requirements/requirements.html.

[45] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. "NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis." In: *Communications of the ACM* 65.1 (2021), pp. 99–106.

[46] Philipp Mittendorfer and Gordon Cheng. "3D surface reconstruction for robotic body parts with artificial skins." In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2012, pp. 4505–4510. DOI: 10.1109/IROS.2012.6385559.

[47] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. "Instant Neural Graphics Primitives with a Multiresolution Hash Encoding." In: *ACM Transactions on Graphics (TOG), Proceedings of SIGGRAPH*. Vol. 41. 4. 2022, pp. 1–15.

[48] OpenCV Contributors. *Camera Calibration and 3D Reconstruction — OpenCV documentation*. Accessed: 2025-07-10. 2024. URL: https://docs.opencv.org/4.x/d9/d0c/group__calib3d.html.

[49] Polycam. *Gaussian Splatting Tool Overview*. Accessed: 2025-08-12. 2024. URL: https://poly.cam/tools/gaussian-splatting.

[50] Polycam. *What Are the Different Photogrammetry Processing Types?* Accessed: 2025-08-12. 2024. URL: https://learn.poly.cam/hc/en-us/articles/30299027821716-What-Are-the-Different-Photogrammetry-Processing-Types.

[51] Polycam. *Pricing – Polycam*. Zuletzt abgerufen am October 15, 2025. 2025. URL: https://poly.cam/pricing.

[52] Polycam. *Which Devices Are Supported by Polycam*. Zuletzt abgerufen am October 15, 2025. 2025. URL: https://learn.poly.cam/hc/en-us/articles/34419168797972-Which-Devices-Are-Supported-by-Polycam.

[53] Zhihao Que, Yutong Guo, Yuefan Wu, Jingyi Yu, and Lan Xu. "A Survey on Neural Radiance Fields." In: *arXiv preprint arXiv:2004.03805* (2020). URL: https://arxiv.org/abs/2004.03805.

[54] Fabio Remondino and Sabry El-Hakim. "Image-based 3D modelling: a review." In: *The Photogrammetric Record* 21.115 (2006), pp. 269–291. DOI: 10.1111/j.1477-9730.2006.00383.x.

[55] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. "ORB: An efficient alternative to SIFT or SURF." In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. IEEE, 2011, pp. 2564–2571. DOI: 10.1109/ICCV.2011.6126544.

[56] Johannes L. Schonberger and Jan-Michael Frahm. "Structure-from-Motion Revisited." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 4104–4113. DOI: 10.1109/CVPR.2016.445. URL: https://openaccess.thecvf.com/content_cvpr_2016/papers/Schonberger_Structure-From-Motion_Revisited_CVPR_2016_paper.pdf.

[57] Thomas Schöps, Johannes L. Schönberger, Silvano Galliani, Torsten Sattler, Konrad Schindler, Marc Pollefeys, and Andreas Geiger. "A Multi-View Stereo Benchmark With High-Resolution Images and Multi-Camera Videos." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 3260–3269. DOI: 10.1109/CVPR.2017.272. URL: https://openaccess.thecvf.com/content_cvpr_2017/html/Schops_A_Multi-View_Stereo_CVPR_2017_paper.html.

[58] Noah Snavely, Steven M. Seitz, and Richard Szeliski. "Photo Tourism: Exploring Photo Collections in 3D." In: *SIGGRAPH Conference Proceedings*. Association for Computing Machinery, 2006, pp. 835–846. DOI: 10.1145/1141911.1141964.

[59] Noah Snavely, Steven M. Seitz, and Richard Szeliski. "Modeling the World from Internet Photo Collections." In: *International Journal of Computer Vision* 80.2 (2008), pp. 189–210. DOI: 10.1007/s11263-007-0107-3.

[60] Richard Szeliski. *Computer Vision: Algorithms and Applications*. Springer, 2010. ISBN: 978-1-84882-934-3. DOI: 10.1007/978-1-84882-935-0. URL: https://szeliski.org/Book/.

[61] Shaharyar Khan Tareen and Zahra Saleem. "A comparative analysis of SIFT, SURF, KAZE, AKAZE, ORB, and BRISK." In: Mar. 2018. DOI: 10.1109/ICOMET.2018.8346440.

[62] Bill Triggs, Philip F. McLauchlan, Richard I. Hartley, and Andrew W. Fitzgibbon. "Bundle Adjustment — A Modern Synthesis." In: *Vision Algorithms: Theory and Practice*. Springer, 2000, pp. 298–372.

[63] Jelle Vermandere, Maarten Bassier, and Maarten Vergauwen. *Semantic UV Mapping to Improve Texture Inpainting for Indoor Scenes*. Submitted to EG UK Computer Graphics & Visual Computing (CGVC) 2024. 2024. arXiv: 2407.09248 [cs.CV].

[64] Fangjinhua Wang, Qingtian Zhu, Di Chang, Quankai Gao, Junlin Han, Tong Zhang, Richard Hartley, and Marc Pollefeys. *Learning-based Multi-View Stereo: A Survey*. PDF. Preprint; categories of learning-based MVS (depth map-, voxel-, NeRF-, 3D Gaussian Splatting-based, and large feed-forward methods). n.d.

[65] X. Wang et al. "A 3D Reconstruction Method for Augmented Reality Sandbox Based on Depth Sensor." In: *2021 IEEE 2nd International Conference on Information Technology, Big Data and Artificial Intelligence (ICIBA)*. Vol. 2. 2021, pp. 844–849. DOI: 10.1109/ICIBA52610.2021.9687867.

[66] X. Wang et al. "Neural Radiance Fields in Medical Imaging: Challenges and Next Steps." In: *arXiv preprint* (2024). arXiv: 2402.17797 [eess.IV].

[67] Yiwen Wang, Siyuan Chen, and Ran Yi. "SG-Splatting: Accelerating 3D Gaussian Splatting with Spherical Gaussians." In: *arXiv preprint arXiv:2501.00342* (2024).

[68] Zhou Wang, Alan C. Bovik, Hamid R. Sheikh, and Eero P. Simoncelli. "Image quality assessment: From error visibility to structural similarity." In: *IEEE Transactions on Image Processing* 13.4 (2004), pp. 600–612. DOI: 10.1109/TIP.2003.819861.

[69] Wikipedia contributors. *Spherical harmonics — Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/wiki/Spherical_harmonics. Accessed: 2025-08-27. 2023.

[70] Wikipedia contributors. *Real-time computer graphics*. https://en.wikipedia.org/wiki/Real-time_computer_graphics. Accessed: October 15, 2025. 2025.

[71] Wenhui Xiao, Remi Chierchia, Rodrigo Santa Cruz, Xuesong Li, David Ahmedt-Aristizabal, Olivier Salvado, Clinton Fookes, and Leo Lebrat. "Neural Radiance Fields for the Real World: A Survey." In: *arXiv preprint arXiv:2501.13104* (2025). CC BY 4.0 License. URL: https://arxiv.org/abs/2501.13104.

[72] Bahia Yahya-Zoubir, Latifa Hamami, Llies Saadaoui, and Rafik Ouared. "Automatic 3D Mesh-Based Centerline Extraction from a Tubular Geometry Form." In: *Information Technology and Control* 45.2 (2016), p. 12162. DOI: 10.5755/j01.itc.45.2.12162. URL: http://dx.doi.org/10.5755/j01.itc.45.2.12162.

[73] Yao Yao, Zexiang Luo, Shiwei Li, Tian Fang, and Long Quan. "Recurrent MVSNet for High-Resolution Multi-View Stereo Depth Inference." In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2019, pp. 5525–5534. DOI: 10.1109/CVPR.2019.00567. URL: https://openaccess.thecvf.com/content_CVPR_2019/html/Yao_Recurrent_MVSNet_for_High-Resolution_Multi-View_Stereo_Depth_Inference_CVPR_2019_paper.html.

[74] Yao Yao, Zixin Luo, Shiwei Li, Tian Fang, and Long Quan. "MVSNet: Depth Inference for Unstructured Multi-View Stereo." In: *Proceedings of the European Conference on Computer Vision (ECCV)*. Springer, 2018, pp. 767–783. DOI: 10.1007/978-3-030-01234-2_47.

[75] Guofeng Zhang, Jiaya Jia, Lu Yuan, and Hujun Bao. "An Efficient and Robust Hybrid SfM Method for Large-Scale Scenes." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018, pp. 6558–6567. DOI: 10.1109/CVPR.2018.00685.

[76] Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. "The Unreasonable Effectiveness of Deep Features as a Perceptual Metric." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018, pp. 586–595. DOI: 10.1109/CVPR.2018.00068. URL: https://openaccess.thecvf.com/content_cvpr_2018/html/Zhang_The_Unreasonable_Effectiveness_CVPR_2018_paper.html.

[77] Zhengyou Zhang. "Microsoft Kinect Sensor and Its Effect." In: *IEEE MultiMedia* 19.2 (2012), pp. 4–10. DOI: 10.1109/MMUL.2012.24.

[78] Matthias Zwicker, Hanspeter Pfister, Jeroen van Baar, and Markus Gross. "Surface Splatting." In: *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '01. New York, NY, USA: ACM, 2001, pp. 371–378. DOI: 10.1145/383259.383300.