Linköping University | Department of Science and Technology
Master's thesis, 30 ECTS | Datavetenskap
2025 | LIU-ITN/LITH-EX-A--25/070--SE

Neural Rendering Dataset Collection

– Degree Project TQDV30 - 30.0 hp

Neural Rendering Dataset Insamling

Shaoxuan Yin

Supervisor : Sergey Ignatenko
Examiner : Jonas Unger

Linköpings universitet
SE–581 83 Linköping
+46 13 28 10 00 , www.liu.se

Abstract

Neural rendering methods such as Neural Radiance Fields (NeRF) and 3D Gaussian Splatting have transformed novel view synthesis, yet their development relies heavily on standardized datasets for training and evaluation. This thesis addresses the dataset gap by developing a multi-camera capture system and collecting two neural rendering datasets: a studio object dataset and a large-scale outdoor heritage site reconstruction.

We designed and implemented an automated capture system using twelve synchronized industrial cameras arranged on a quarter-hemisphere arc with a motorized turntable. The system captures 432 images per object across 36 rotation positions. Custom software (7,950 lines: C# WinUI frontend and C++ hardware backend) controls camera synchronization, turntable automation, and data organization. The studio dataset contains 15 objects spanning diverse materials including diffuse, glossy, transparent, and reflective surfaces.

For large-scale capture, we documented Gränsö Castle in Sweden using 5,262 images from drone and SLR photography. This dataset tests neural rendering scalability on architectural heritage sites with varying lighting conditions.

We compared traditional photogrammetry (RealityCapture) against neural rendering methods (Nerfacto, Splatfacto) on both datasets. Results show that Splatfacto outperforms Nerfacto with average PSNR of 33.27 dB versus 22.07 dB on studio objects. Neural methods successfully reconstruct challenging materials (glass, metal) where photogrammetry fails due to feature matching limitations on specular surfaces. For large-scale scenes, photogrammetry remains practical for complete geometric models while neural rendering excels in bounded high-quality visualization.

The studio dataset and the capture software are released as open-source resources for the research community.

Acknowledgments

First and foremost, I want to thank my supervisor Sergey Ignatenko and my examiner Jonas Unger. Their generous support made this thesis possible. They gave me access to an RTX 5090 workstation, the VR lab, and all the equipment I needed for my experiments. When my laptop broke down twice during this project, they made sure I could keep working without missing a beat. I am truly grateful for that.

Beyond the resources, I learned so much from their guidance. Sergey patiently walked me through the complexities of photogrammetry, and Jonas provided valuable insights of neural rendering that sharpened my research. And the most valuable lesson I learned from this project is the way to do the research: Rank the priorities; progress incrementally, and repeat. And the knowledge and practice I gained from working with them go far beyond what ended up in this thesis. It was a period of memorable time that we worked together.

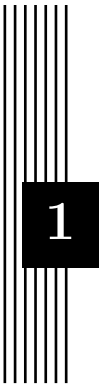
I also want to thank my family and friends for their support throughout my studies in Sweden.

Norrköping, December 2025
Shaoxuan Yin

Contents

List of Figures

List of Tables



1 Introduction

1.1 Background

Walk around any object and you build a mental picture of its shape. Your brain combines views from different angles into a coherent 3D understanding without conscious effort. This ability is so natural that we rarely think about it. But teaching computers to do the same thing turns out to be surprisingly hard, and getting it right has become increasingly important.

The applications are everywhere, even if people do not always notice them. Online shopping sites now show products you can rotate and view from any angle. Real estate listings include virtual tours where you can look around a room. Film studios blend digital characters with live footage, which requires knowing exactly where the camera was and what the 3D scene looks like. Google Maps builds 3D city models from street-level photos. Museums digitize artifacts so researchers worldwide can examine them without traveling. Surveyors measure buildings and terrain from drone photographs. Each of these relies on the same underlying problem: reconstructing 3D structure from 2D images.

Two main approaches exist today. Photogrammetry uses traditional computer vision algorithms that have been refined over decades. Neural rendering uses machine learning, specifically neural networks, and has developed rapidly since 2020. Both take photographs as input. Both can produce impressive results. But they work differently, and understanding those differences matters for anyone creating datasets to develop or test these methods.

Photogrammetry builds 3D models through geometry. The basic idea is to find the same point in multiple photos, figure out where each camera was positioned, and triangulate the 3D location of that point. Repeat this for thousands or millions of points and you get a detailed model. The process has well-defined steps. First, detect distinctive features in each image: corners, edges, textured regions that look unique enough to identify in other views. Second, match these features across images to find correspondences. Third, use Structure from Motion to estimate where each camera was when each photo was taken. Finally, compute dense 3D geometry using Multi-View Stereo algorithms. Software packages like RealityCapture, Metashape, and COLMAP implement these pipelines and can work remarkably well.

But photogrammetry has fundamental limits tied to how it works. The feature matching step needs texture. A stone wall with varied patterns gives thousands of matchable points. A white plastic bottle gives almost none. Shiny surfaces cause different problems: they reflect the environment, so the same point on the object looks completely different from different

angles. Glass is worse still. Light passes through it, bends, and the features you detect might not even correspond to points on the glass surface. These are not software bugs to be fixed. They are consequences of how the algorithms work.

Neural rendering emerged around 2020 as a fundamentally different approach. Instead of explicitly computing geometry through feature matching, these methods train neural networks to represent scenes directly. The input is still photographs. But rather than extracting geometric primitives, the network learns to predict what the scene looks like from any viewpoint.

Neural Radiance Fields, or NeRF [19], was the paper that started this wave of research. The core idea sounds almost too simple: represent a scene as a function that maps any 3D point and viewing direction to a color and density value. A neural network learns this function from training images. To render a new view, cast rays from the virtual camera through each pixel, sample points along each ray, query the network for colors and densities, and composite them together. The results surprised many researchers. NeRF could reproduce fine geometric detail that photogrammetry missed. More importantly, it handled view-dependent effects naturally. Reflections, translucency, subtle color shifts as viewing angle changes. The network just learns to reproduce what it saw in the training images, without needing explicit models for these phenomena.

The tradeoff is computational cost. Training a NeRF model takes hours for a single scene, even on powerful GPUs. Rendering is slow too, because each pixel requires many network evaluations along its ray. Researchers have worked on faster variants. Instant-NGP, introduced in 2022, used hash-based encodings to speed up training dramatically. But the fundamental architecture still involves ray marching and network queries, which limits rendering speed.

3D Gaussian Splatting [14] arrived in 2023 with a different solution. Instead of an implicit function queried along rays, it uses explicit primitives: 3D Gaussians, which you can think of as fuzzy colored ellipsoids floating in space. Start with a sparse point cloud from structure from motion. Place a Gaussian at each point. Then optimize their positions, shapes, colors, and opacities to match the training views. The key insight is that these Gaussians can be rendered extremely fast using rasterization rather than ray marching. The result is real-time performance, hundreds of frames per second, with quality matching or exceeding NeRF methods. This makes Gaussian Splatting practical for applications that need interactive viewing.

All of these methods, photogrammetry and neural rendering alike, need data to work with. For photogrammetry, input photographs determine whether feature matching succeeds. For neural rendering, training images are literally what the network learns from. Bad data produces bad results, and there is no algorithmic fix for insufficient or poorly captured input.

This makes **high quality** datasets crucial to the research community. Developing new algorithms requires test data to evaluate them on. Comparing methods fairly requires identical input, so differences in results reflect differences in the methods rather than differences in the data. Understanding where methods fail requires datasets that include difficult cases. And since the field moves so quickly, with NeRF arriving in 2020, Instant-NGP in 2022, and Gaussian Splatting in 2023, each new generation of methods needs to be tested against previous ones on standardized benchmarks.

Several datasets have become standard references. DTU [13] provides controlled captures of small objects with known camera positions and ground-truth geometry from structured light scanning. It has been widely used for evaluating multi-view stereo algorithms. Tanks and Temples [16] offers larger outdoor scenes with ground-truth from laser scanning, testing methods on real-world complexity. More recently, Mip-NeRF 360 [1] introduced unbounded outdoor scenes specifically designed for neural rendering evaluation.

These datasets have driven progress, but gaps remain. DTU covers only small objects in controlled conditions. Tanks and Temples has limited variety in materials. Most datasets focus on scenes where reconstruction is expected to work, which means they tell us where methods succeed but not where they fail. Few datasets include deliberate failure cases, difficult materials, or documented capture protocols. And while neural rendering methods have advanced rapidly, the datasets used to evaluate them often predate the methods themselves.

1.2 Motivation

Given how useful these reconstruction methods are, one might expect a wealth of high-quality datasets for developing and testing them. The reality is more complicated. Creating good datasets takes significant effort, and several practical problems remain unsolved.

The first problem is quality verification. When someone captures new data, they usually just run a reconstruction algorithm and see if it looks reasonable. Photogrammetry software does report useful metrics like reprojection error and alignment success rate, but these numbers rarely appear in dataset publications. Without quantitative quality measures, it is hard to know if poor reconstruction results come from algorithm limitations or from bad input data.

Second, capturing large outdoor scenes introduces uncontrollable variables. In a studio you control everything: lighting, background, camera positions. Outside, moving pedestrians and vehicles appear in frames, requiring careful timing or post-processing removal. The sun's position shifts during multi-hour capture sessions, causing shadows to move and lighting intensity to change—morning captures look different from afternoon ones of the same structure. You probably need both drone footage for overview and ground-level shots for detail, which means combining different cameras with different characteristics. Weather can change mid-session. Parts of the scene may be inaccessible. Experienced photogrammetry practitioners have figured out solutions to these problems, but the knowledge tends to stay informal and is not written down in a way that neural rendering researchers can easily find.

Third, most datasets avoid hard materials. Benchmark scenes typically feature well-textured surfaces where reconstruction is expected to succeed. But real objects are often glossy, transparent, or just plain smooth. A white plastic spray bottle has almost no texture for feature matching. A glass vase refracts light in complex ways. These failure cases are actually informative. If we want to understand the limits of current methods, we need datasets that include things those methods cannot handle.

Fourth, published datasets describe results but not process. Camera placement, lighting setup, turntable rotation increments, drone flight patterns. These practical details determine whether a capture succeeds or fails, but papers usually skip them. Everyone working on new capture systems ends up solving the same problems independently.

1.3 Aim

This thesis aims to create high-quality multi-view datasets for neural rendering research and document the practical knowledge needed to produce such datasets. Rather than focusing on one type of capture, we deliberately chose two very different scenarios to explore the range of challenges involved.

The first scenario is controlled studio capture. Here we can specify everything: camera positions, lighting, backgrounds, the object itself. This control lets us isolate specific factors. What happens with different materials? How many viewpoints are enough? Does camera synchronization matter? A studio setup also lets us capture many objects efficiently, building a dataset with variety. We designed a system with twelve industrial cameras arranged in an arc around a motorized turntable. The object sits at the center while the turntable rotates in small increments, capturing images at each position. Automating this process required writing substantial software, totaling 7,950 lines of code (1,700 C# for the WinUI frontend, 6,250 C++ for the hardware control backend), to synchronize the cameras, control the turntable motor, and manage the image data.

The second scenario is large-scale outdoor capture, specifically Gränsö Castle, a historical site in southeastern Sweden. Outdoor capture presents the opposite situation: almost nothing is under our control. The lighting changes as the sun moves and clouds pass. The scene is too large to photograph from a single position, requiring both drone flights for aerial coverage and ground-level photography for facade details. Different cameras with different characteristics must be combined into a coherent dataset. Parts of the site may be inaccessible. This is

the messy reality that anyone capturing outdoor scenes faces, and the lessons learned apply broadly.

We validate both datasets by processing them through photogrammetry and neural rendering pipelines. Alignment metrics from RealityCapture, particularly reprojection error and success rate, provide quantitative evidence of data quality. Running the same data through different reconstruction methods shows how they compare on identical input.

1.4 Research Questions

This thesis investigates three research questions related to creating datasets for neural rendering.

1. **What are the practical challenges of creating multi-view datasets for neural rendering?**

   Studio capture requires uniform backdrops, diffuse lighting, synchronized exposure across cameras, and manual focus adjustment for each lens. Outdoor capture faces different problems: the SLR's focus settings were not adjusted during capture, producing blurry regions at certain distances; the two-hour session at Gränsö Castle caused noticeable sun and shadow changes; and drone battery limits require multiple flights with careful overlap planning.

2. **Which dataset characteristics matter most for neural rendering quality?**

   Image count, camera distribution, resolution, and scene complexity all affect results. Reprojection error from RealityCapture quantifies alignment quality. Analyzing which captures achieve sub-pixel accuracy reveals factors that correlate with successful reconstructions.

3. **Which materials are hardest to capture?**

   Photogrammetry fails on textureless, reflective, and transparent surfaces due to feature matching limitations. The studio dataset includes a white spray bottle and glass vase as deliberate test cases. Photogrammetry failed on both; 3D Gaussian Splatting succeeded on both.

1.5 Delimitations

This thesis focuses on static scenes only; moving objects are out of scope. The studio hardware uses Teledyne DALSA cameras with their Sapera SDK, which means the exact setup is not easy to replicate, though the methods should transfer to other equipment. Neural rendering experiments rely on Nerfstudio implementations (Nerfacto, Splatfacto) without algorithm modifications. The outdoor dataset covers a single location captured in autumn. Some images required downsampling due to GPU memory limits.

1.6 Contributions

This thesis contributes:

- A studio dataset of 15 objects, each captured with 432 to 720 images from a 12-camera turntable system. Camera poses come from ArUco marker alignment with sub-pixel accuracy (mean reprojection error 0.53–0.76 px depending on object).
- An outdoor dataset of Gränsö Castle with 5,207 images from drone and handheld cameras. Photogrammetry aligned 77% of the images successfully.

- Capture software totaling 7,950 lines (1,700 C# for the WinUI frontend, 6,250 C++ for the hardware control backend) for synchronized multi-camera acquisition, turntable control, and automated capture sequences.
- Documented capture protocols for both studio and outdoor scenarios, covering camera placement, lighting, rotation steps, and drone flight planning.
- Reconstruction results from RealityCapture, Nerfacto, and Splatfacto on the same input data, showing how different methods handle identical scenes.

1.7 Thesis Structure

Table 1.1: Overview of thesis chapters

Ch.	**Title**	**Content**
2	Theory and Background	Photogrammetry fundamentals, NeRF, Gaussian Splatting, existing datasets
3	Methodology	Capture system design, outdoor capture strategy, evaluation framework
4	Implementation	Capture software, camera calibration, data processing pipelines
5	Results	Datasets, alignment metrics, reconstruction outputs
6	Discussion	Material challenges, practical lessons learned
7	Conclusion	Summary, future work

2 Related Work

Dataset creation for neural rendering builds on several research areas. This chapter examines the techniques and tools that shaped our approach. We begin with neural rendering methods including NeRF, 3D Gaussian Splatting, and the Nerfstudio framework used in our experiments. Then we cover multi-view capture systems, camera calibration, and fiducial marker systems for pose estimation. After that, we review existing datasets and identify gaps in current benchmarks. We discuss photogrammetry tools and techniques for handling challenging materials like reflective and transparent surfaces. The chapter ends by summarizing how our work addresses these gaps.

2.1 Neural Rendering Techniques

Neural Radiance Fields

NeRF [19] changed how we represent 3D scenes. Instead of meshes or point clouds, it uses a neural network to encode the entire scene. The network takes a 3D position $\mathbf{x}$ and viewing direction $\mathbf{d}$ as input. It outputs color $\mathbf{c}$ and density σ at that point. To render an image, NeRF shoots rays through each pixel and samples points along them. The colors and densities get combined using volume rendering equations.

The original NeRF had problems though. It couldn't handle different image scales well - close-ups looked blurry while distant views had aliasing. Mip-NeRF [1] fixed this by reasoning about the volume each ray passes through, not just points. Later, Mip-NeRF 360 tackled unbounded outdoor scenes. The trick was mapping infinite 3D space to a bounded region using contraction functions. Zip-NeRF [2] sped things up by mixing neural networks with hash grids.

Instant-NGP [20] achieved a major speedup by replacing the MLP with a multiresolution hash encoding. Instead of learning spatial features through network layers, it stores them directly in a hash table indexed by position. Training dropped from hours to minutes, and the method can handle scenes of varying complexity by automatically allocating hash table entries where detail is needed. This made NeRF practical for interactive applications and rapid experimentation.

But even fast NeRF variants render slowly compared to traditional graphics. Each pixel requires sampling dozens of points along its ray and querying the representation at each point.

For real-time applications, this is too slow. This limitation pushed researchers toward explicit representations that could leverage GPU rasterization.

3D Gaussian Splatting (3DGS)

3D Gaussian Splatting (3DGS) [14] takes a totally different approach. It represents the scene as millions of 3D Gaussian blobs. Each Gaussian has a position μ, a 3x3 covariance matrix Σ that controls its shape, an opacity α, and spherical harmonics for color. During rendering, these Gaussians get projected onto the image plane and blended together. The whole process runs on the GPU using rasterization, so it hits 30+ FPS even at 1080p resolution.

The speed comes from avoiding ray marching entirely. Instead of sampling hundreds of points per ray, Gaussian Splatting just sorts the Gaussians by depth and composites them front-to-back. You can also edit scenes easily - just move or delete some Gaussians. With NeRF, you'd need to retrain the whole network. Wu et al. [33] even got it working for videos by adding time as an extra dimension.

Wu et al. [34] surveyed the explosion of work on Gaussian Splatting since 2023. People are compressing the representations, making them editable, and combining them with diffusion models. The big question is when to use NeRF versus Gaussian Splatting. NeRF gives smoother surfaces and handles transparency better. Gaussian Splatting runs faster and is easier to edit. Neither method dominates completely.

Comparative Analysis

Gao et al. [7] tested MVS, NeRF, and 3D Gaussian Splatting on the same outdoor scenes. Gaussian Splatting trained 10x faster and rendered 100x faster than NeRF. But when they extracted meshes, NeRF's were cleaner with fewer holes. NeRF also handled shadows and reflections better - probably because it models light transport more accurately. MVS still won for geometric accuracy when given enough images. Each method has clear strengths: Gaussian Splatting for speed, NeRF for visual quality, MVS for geometric precision.

Nerfstudio Framework

Nerfstudio [29] provides a modular framework for neural radiance field development. Released at SIGGRAPH 2023 by researchers at Berkeley, it standardizes the training pipeline and provides implementations of multiple methods under a common interface. This makes it easier to compare methods fairly and to experiment with different configurations.

The framework includes Nerfacto, a method that combines techniques from multiple papers into a single practical implementation. Nerfacto uses appearance embeddings to handle exposure variations between training images, proposal networks for efficient ray sampling, and hash encodings similar to Instant-NGP for fast training. These design choices make it robust across different capture conditions.

Splatfacto is Nerfstudio's implementation of 3D Gaussian Splatting, built on the gsplat library. It provides the same training interface as Nerfacto, making direct comparison straightforward. The implementation supports density-based pruning, adaptive densification, and optional MCMC refinement for improved quality. Both Nerfacto and Splatfacto can export results for visualization in web-based viewers.

Nerfstudio serves as the platform for all experiments in this thesis because it provides consistent evaluation across methods. Training a scene requires only specifying the data path and method name. The framework handles camera pose estimation through COLMAP integration, data loading, training scheduling, and metric computation automatically.

2.2 Multi-View Capture Systems and Calibration

Camera Calibration Fundamentals

Camera calibration determines the mapping between 3D world coordinates and 2D image coordinates. The camera model is typically split into intrinsic parameters (focal length, principal point, distortion coefficients) that depend on the camera optics, and extrinsic parameters (rotation and translation) that describe the camera's position in the world.

Zhang's checkerboard calibration method [41] remains the standard approach for estimating intrinsics. The method requires capturing 10-20 images of a planar checkerboard pattern from different angles. The known geometry of the checkerboard, combined with detected corner positions, provides constraints for solving the camera parameters. The approach is practical because checkerboard patterns are easy to print and corner detection is robust.

Lens distortion must be modeled to achieve accurate calibration. Radial distortion causes straight lines to appear curved, particularly near image edges. The distortion is typically parameterized as:

$$r' = r\left(1 + k_1 r^2 + k_2 r^4 + k_3 r^6\right) \tag{2.1}$$

where r is the distance from the principal point and k_1, k_2, k_3 are distortion coefficients. Tangential distortion, caused by imperfect lens mounting, adds additional parameters but is often negligible for quality lenses.

For multiple cameras, extrinsic calibration determines how cameras are positioned relative to each other. Hartley and Zisserman's book [10] covers the mathematical foundations of multi-view geometry. The eight-point algorithm estimates the fundamental matrix relating two views, from which relative pose can be extracted. Structure-from-motion pipelines apply these techniques to estimate camera poses from image correspondences automatically.

Industrial setups require additional precision. Luhmann et al. [18] measured environmental effects on calibration stability, finding that temperature changes affect focal length by up to 0.1%. Mechanical stress on lens mounts can shift the principal point. Chromatic aberration in lower-quality lenses shifts red and blue channels by 2-3 pixels at image edges. Achieving subpixel reprojection accuracy requires modeling these effects and recalibrating when conditions change.

Industrial Camera Systems

Industrial cameras differ from consumer cameras in ways that matter for multi-view capture. The most significant difference is the shutter mechanism. Consumer cameras typically use rolling shutters that expose different rows of pixels at slightly different times. For moving objects or rapidly changing scenes, this causes geometric distortion. Industrial cameras use global shutters that expose all pixels simultaneously, eliminating this artifact.

Hardware synchronization allows multiple industrial cameras to capture frames at precisely the same instant. A trigger signal causes all connected cameras to begin exposure within microseconds of each other. This temporal coherence is essential for capturing dynamic scenes or ensuring consistent lighting conditions across views. Consumer cameras lack this capability and must rely on software synchronization, which introduces timing jitter.

The Sapera LT SDK [30] provides a comprehensive framework for controlling Teledyne DALSA cameras. The SDK handles low-level details like direct memory access for high-throughput image transfer and hardware triggering for synchronization. Support for the GenICam standard ensures compatibility with cameras from different manufacturers while maintaining deterministic performance.

Industrial camera systems also provide consistent sensor characteristics across units. Consumer cameras may vary slightly in color response, gain, and noise characteristics even within the same model. Industrial cameras are typically manufactured to tighter tolerances and can

be calibrated more precisely. This simplifies the task of ensuring consistent image quality across a multi-camera array.

The disadvantages are cost and complexity. Industrial cameras cost several times more than consumer equivalents with similar resolution. The SDK requires significant programming effort to use effectively. Lenses, mounting hardware, and cabling add additional expense. These factors explain why most neural rendering datasets use consumer cameras despite the technical advantages of industrial equipment.

Multi-View Capture Rigs

Various capture rig designs have been proposed for different applications. Dome-based systems provide comprehensive coverage but require significant infrastructure. Arc-based designs, as employed in our work, offer a balance between coverage and practicality. The key considerations include baseline distribution, overlap between views, and mechanical stability during capture.

Turntable Capture Systems

Turntable-based capture provides a cost-effective alternative to multi-camera domes for object scanning. The object rotates while one or more fixed cameras capture images at regular angular intervals. This approach requires fewer cameras but takes longer to capture since images are acquired sequentially rather than simultaneously.

Fan et al. [5] demonstrate that turntable setups can capture hundreds of valid images in under two minutes, compared to under fifty images with handheld capture in the same time. The consistent camera-to-object distance and controlled rotation increments produce more uniform angular sampling than freehand photography.

Fan et al. [6] developed automated view planning for multi-object scanning, optimizing turntable rotation and camera positioning to ensure complete coverage. Their system places multiple objects in the capture volume simultaneously, reducing the per-object capture time.

Commercial solutions like the Artec Turntable integrate rotation control with scanning software, automating the capture sequence entirely. The Arago system combines photogrammetry with photometric capture, using programmable lighting arrays alongside the turntable to acquire both geometry and material properties.

Our capture system uses a motorized turntable with twelve fixed cameras arranged in an arc. This hybrid approach combines the viewpoint diversity of multi-camera systems with the angular coverage of turntable rotation. A single rotation captures 432 images (12 cameras $\times$ 36 positions), providing dense coverage in approximately two minutes per object.

Fiducial Markers for Pose Estimation

ArUco markers [8] are square binary fiducial patterns designed for robust detection and pose estimation. Each marker encodes a unique identifier in its internal pattern, enabling automatic recognition even under partial occlusion. The four corners of each marker provide point correspondences for computing the marker's 6-DOF pose relative to the camera.

The OpenCV library [22] provides detection and pose estimation functions for ArUco markers. ChArUco boards combine ArUco markers with a checkerboard pattern, providing more corner points for improved calibration accuracy. The checkerboard corners can be detected with sub-pixel precision, while the ArUco markers enable automatic identification of which corners belong to which board.

For multi-camera systems, placing ArUco markers in the capture volume provides a common reference frame. Each camera independently detects the markers and estimates their poses. Since all cameras see the same markers, their relative positions can be computed by

comparing marker poses across views. This approach avoids the need for direct camera-to-camera feature matching during calibration.

ArUco markers establish ground-truth camera poses in our studio dataset. Markers placed on the turntable platform are detected in every captured image. The known marker geometry, combined with detected corner positions, provides camera pose estimates that serve as initialization for photogrammetry alignment. This produces more reliable poses than purely image-based alignment, particularly for objects with limited texture.

2.3 Existing Datasets and Benchmarks

DTU Multi-View Stereo Dataset

The DTU dataset [13] established the standard for controlled multi-view stereo evaluation. The dataset was captured using a robot arm holding a camera that moved to 49 or 64 predefined positions around each object. This mechanical positioning ensures precise and repeatable camera trajectories across all 80 scenes.

Ground truth geometry comes from structured light scanning, providing sub-millimeter accuracy for quantitative evaluation. The combination of accurate geometry and calibrated camera poses makes DTU particularly valuable for measuring reconstruction error. Methods can be compared not just on visual quality but on geometric fidelity to the actual surface.

The controlled laboratory conditions ensure consistent lighting and background across all scenes. However, these same conditions limit generalization to real-world scenarios where lighting varies, backgrounds are complex, and camera trajectories are less structured. The objects are also relatively small, fitting on a tabletop, which means the dataset does not test scalability to larger scenes.

Despite these limitations, DTU remains the most widely used benchmark for multi-view stereo and neural rendering evaluation. Its reliable ground truth and manageable scale make it practical for rapid iteration during method development. Many papers report DTU results even when their primary focus is on larger or more challenging scenes.

Tanks and Temples

Tanks and Temples [16] addresses the limitations of laboratory datasets by providing real-world scenes captured under natural conditions. The benchmark is divided into two tiers based on difficulty.

The intermediate tier includes eight scenes: Family, Francis, Horse, Lighthouse, M60, Panther, Playground, and Train. These scenes feature a mix of indoor and outdoor environments with varying complexity. Camera trajectories are unstructured, mimicking how a person might walk around and photograph a scene of interest. This stands in contrast to the mechanical camera positioning of DTU.

The advanced tier presents larger and more challenging scenes: Auditorium, Ballroom, Courtroom, Museum, Palace, and Temple. These scenes span hundreds of square meters and include complex occlusions, repetitive structures, and challenging lighting conditions. Few methods achieve strong results on the advanced tier, making it a useful benchmark for state-of-the-art evaluation.

Ground truth geometry is obtained through industrial laser scanning with millimeter-level accuracy. The evaluation protocol computes precision and recall at multiple distance thresholds, producing F-score curves that characterize reconstruction quality across scales. This evaluation is more nuanced than single-number metrics because it reveals whether methods excel at coarse structure or fine detail.

The unstructured camera trajectories and natural lighting of Tanks and Temples better represent practical capture scenarios than laboratory datasets. However, the lack of controlled conditions means that factors like motion blur, exposure variation, and transient objects may

affect some images. These real-world complications are part of what makes the benchmark challenging and relevant.

BlendedMVS

BlendedMVS [37] takes a different approach by rendering synthetic views from reconstructed 3D models. With 113 scenes and over 17,000 images, it provides the scale necessary for training deep learning models. The dataset covers diverse scene types including architecture, sculptures, and small objects. While the synthetic nature ensures perfect ground truth, it may not fully capture the complexities of real image formation.

Mip-NeRF 360 Dataset

The Mip-NeRF 360 dataset [1] specifically targets unbounded outdoor scenes where cameras orbit around a central point. It contains nine scenes: four indoor and five outdoor, with between 100 and 330 images per scene. Camera poses were estimated using COLMAP. The dataset has become a standard benchmark for evaluating neural rendering methods on challenging real-world captures.

The outdoor scenes in Mip-NeRF 360 are particularly challenging due to distant backgrounds, varying lighting conditions, and complex geometry. Methods must handle content at multiple scales simultaneously, from nearby objects to distant trees and sky. The dataset demonstrated that earlier NeRF methods struggled with unbounded scenes, motivating the development of scene contraction techniques and improved regularization.

Deep Blending Dataset

Deep Blending [11] provides 19 diverse scenes for image-based rendering research. Each scene includes input images, COLMAP reconstructions, textured meshes from RealityCapture, and refined depth maps. The dataset covers both indoor and outdoor environments with varying complexity.

Unlike datasets focused purely on neural rendering, Deep Blending provides intermediate reconstruction outputs that researchers can use to study different stages of the reconstruction pipeline. The inclusion of commercial photogrammetry results alongside structure-from-motion outputs enables comparison between reconstruction approaches on identical input data.

ScanNet++ Dataset

ScanNet++ [38] provides high-fidelity indoor scene captures using iPhone RGBD sensors alongside DSLR photography. The dataset addresses the quality gap between convenient mobile capture and professional photography by providing both modalities for the same scenes.

The dataset includes laser scan ground truth and covers over 450 scenes. The combination of casual mobile capture with high-quality DSLR images makes ScanNet++ valuable for studying how input quality affects reconstruction results. Models trained on DSLR images can be evaluated on mobile captures to assess generalization.

Dataset Limitations and Gaps

Several gaps exist in current neural rendering benchmarks: limited scene diversity, lack of challenging outdoor environments, insufficient coverage of different camera trajectories, and absence of datasets specifically designed for comparing NeRF and Gaussian Splatting. Our work addresses these gaps through the combination of controlled studio captures and large-scale outdoor reconstruction.

2.4 Large-Scale Reconstruction and Photogrammetry

Structure-from-Motion Foundations

Structure-from-Motion (SfM) estimates camera poses and sparse 3D structure from unordered image collections. The pipeline typically proceeds through several stages: feature extraction identifies distinctive points in each image, feature matching finds correspondences between image pairs, and bundle adjustment jointly optimizes camera parameters and 3D point positions to minimize reprojection error.

COLMAP [26] has become the standard open-source SfM implementation. The pipeline uses SIFT features for robust matching under viewpoint and lighting changes. Geometric verification filters incorrect matches using epipolar constraints. Incremental reconstruction starts from a well-conditioned image pair and adds cameras one at a time, running bundle adjustment periodically to maintain accuracy. This approach scales to thousands of images while achieving sub-pixel reprojection error.

The choice between incremental and global SfM approaches involves tradeoffs. Incremental methods like COLMAP are more robust to outliers and can handle challenging image collections, but accumulated error may cause drift in large reconstructions. Global methods solve for all camera poses simultaneously, avoiding drift but requiring more memory and being more sensitive to outliers. COLMAP includes both approaches and can switch between them based on dataset characteristics.

Westoby et al. [32] demonstrate the effectiveness of SfM photogrammetry for geoscience applications, achieving centimeter-level accuracy for terrain mapping at a fraction of the cost of laser scanning. The accessibility of modern SfM tools has enabled large-scale reconstruction projects that were previously infeasible. However, SfM alone produces only sparse point clouds; dense reconstruction requires additional Multi-View Stereo processing.

RealityCapture and Commercial Solutions

RealityCapture represents the state-of-the-art in commercial photogrammetry software. Its ability to process massive datasets efficiently, combine multiple data sources (images, laser scans, depth maps), and produce high-quality textured meshes makes it particularly suitable for heritage documentation.

The software's methodology for addressing difficult situations—such as vegetation, reflective surfaces, and fluctuating lighting—offers significant insights for the development of neural rendering datasets. The incorporation of AI-driven functionalities for automatic masking and alignment enhances the efficiency of the reconstruction pipeline.

Heritage and Archaeological Applications

The protection of cultural heritage has propelled advancements in 3D reconstruction. Remondino et al. [25] examine the cutting-edge techniques in high-density image matching for historical documentation, highlighting the necessity of achieving adequate redundancy and ensuring uniform quality over extensive areas.

Cultural heritage documentation faces distinctive challenges: variations in scale from architectural details to entire structures, weathered surfaces composed of intricate materials, and the necessity for non-invasive capture techniques. These factors significantly influence our strategy for acquiring Gränsö Castle images.

2.5 UAV and Drone-Based Capture

Aerial Photogrammetry

Nex and Remondino [21] present an overview of UAV applications in 3D mapping, covered about access to inaccessible viewpoints, swift coverage of extensive areas, and uniform image quality via planned flight paths. The incorporation of RTK GPS ensures precise georeferencing; however, this is less essential for neural rendering applications that prioritize visual quality over metric accuracy.

Capture Optimization

James et al. [12] examine the optimization of UAV surveys for Structure from Motion (SfM) processing, assessing the influence of ground control points, flying trajectories, and picture overlap on reconstruction quality. Their findings indicate that 80% forward overlap and 60% side overlap achieve an optimal equilibrium between capture duration and reconstruction thoroughness. In neural rendering applications, increased overlap may enhance the smoothness of visual interpolation.

The selection of flying patterns considerably influences reconstruction quality. Nadir flights offer coverage of horizontal surfaces but neglect vertical facades. Oblique angle images are suitable for capturing building facades, but careful planning of the shooting path is necessary to ensure sufficient image overlap. Our approach combines these two strategies and, if necessary, uses handheld devices for additional shots to capture key details.

2.6 Challenging Materials in 3D Reconstruction

Reflective and Specular Surfaces

Reflective surfaces pose fundamental challenges for both photogrammetry and neural rendering. In photogrammetry, feature matching assumes that the same surface point appears similar across different viewpoints. Specular reflections violate this assumption: a shiny metal surface reflects different parts of the environment from different viewing angles, causing the same point to look completely different in each image.

Several neural rendering methods have addressed reflective surfaces specifically. NeRO [17] decomposes scenes into geometry and BRDF components, enabling accurate reconstruction of reflective objects from multiview images. The method models surface reflectance explicitly rather than baking lighting into the radiance field.

For near-perfect mirrors and highly specular surfaces, TraM-NeRF traces reflections through the radiance field, handling mirror-like objects that would otherwise produce incorrect geometry. These specialized methods require additional annotations or assumptions about scene composition.

Transparent and Refractive Objects

Transparent materials like glass present even greater challenges. Light passes through the surface, bends according to the material's refractive index, and may reflect partially at the interface. Features detected through glass may correspond to objects behind the glass rather than points on the glass surface itself. Standard photogrammetry pipelines produce incorrect or missing geometry for transparent regions.

REF^2-NeRF [15] models scenes containing glass cases by explicitly accounting for both reflection and refraction. The method separates view-dependent and view-independent appearance components, enabling reconstruction of objects inside glass showcases.

NU-NeRF [28] reconstructs nested transparent objects without requiring controlled capture environments. The two-stage approach first estimates outer surface geometry, then re-

constructs inner surfaces by modeling light transport through the outer layer. This enables reconstruction of complex arrangements like glass bottles containing liquid.

These methods demonstrate that neural rendering can handle materials that are fundamentally incompatible with feature-based photogrammetry. However, they typically require longer training times and may need method-specific parameter tuning.

Textureless Surfaces

Textureless surfaces lack the distinctive features needed for correspondence matching. A smooth white plastic surface provides no corners, edges, or patterns that feature detectors can identify. Even if the surface has gradual shading variations, these are often too subtle for robust matching.

NeRF and Gaussian Splatting handle textureless surfaces better than photogrammetry because they do not rely on feature matching. Instead, they learn appearance directly from training images. With sufficient viewpoints, the optimization can recover smooth surfaces that would be invisible to feature-based methods. However, reconstructed geometry may be less accurate without strong texture gradients to constrain the solution.

Polarization for Specular Reduction

Cross-polarization photography offers a hardware-based solution for reducing specular reflections during capture. By placing polarizing filters on both the light source and camera lens, oriented perpendicular to each other, specular reflections can be substantially suppressed while diffuse reflections pass through.

Cross-polarized photogrammetry provides an effective workflow for documenting reflective objects. Each photograph is captured twice: once with normal lighting and once with cross-polarization. The polarized images produce cleaner geometry by eliminating specular highlights that would otherwise confuse feature matching.

The limitation of cross-polarization is that it removes information about surface specularity that may be valuable for material appearance. Some workflows capture both polarized and non-polarized images, using the polarized set for geometry reconstruction and the non-polarized set for texture mapping.

The studio dataset includes experiments with polarization filters on the transparent glass vase. While polarization reduced some surface reflections, it could not address the fundamental challenge of refraction. Feature matching still failed because detected features did not correspond to actual surface points. This motivated the decision to include the glass vase as a deliberate failure case for photogrammetry while demonstrating that 3D Gaussian Splatting could handle it successfully.

2.7 Multi-View Stereo Networks

Learning-Based MVS

While our work focuses on neural rendering rather than traditional MVS, understanding learning-based depth estimation provides valuable context. MVSNet [36] pioneered the use of deep learning for multi-view stereo, constructing cost volumes in a differentiable manner. This approach has spawned many different variations, each of which addresses specific limitations.

CasMVSNet [9] introduced cascade cost volumes for memory-efficient processing of high-resolution images. The hierarchical approach, refining depth estimates at multiple scales, provides insights applicable to multi-resolution neural rendering approaches.

Relevance to Neural Rendering

The relationship between MVS and neural rendering is complex. While NeRF and Gaussian Splatting can operate without explicit depth supervision, incorporating geometric priors from MVS can accelerate convergence and improve quality in sparse-view scenarios. Understanding the strengths and limitations of MVS methods informs our dataset design, ensuring sufficient coverage for both geometric reconstruction and appearance modeling.

2.8 Evaluation Metrics and Methodologies

Image Quality Metrics

The evaluation of neural rendering methods relies heavily on image quality metrics that compare rendered images against held-out test views.

Peak Signal-to-Noise Ratio (PSNR) measures the ratio between maximum possible signal power and noise power, computed as:

$$\text{PSNR} = 10 \cdot \log_{10}\left(\frac{\text{MAX}^2}{\text{MSE}}\right) \tag{2.2}$$

where MAX is the maximum pixel value (255 for 8-bit images) and MSE is the mean squared error between rendered and ground truth images. Higher PSNR indicates better reconstruction. However, PSNR correlates poorly with human perception because it treats all pixel errors equally regardless of their visual importance.

Structural Similarity Index (SSIM) [31] addresses this limitation by comparing local patterns of luminance, contrast, and structure:

$$\text{SSIM}(x, y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)} \tag{2.3}$$

where μ denotes mean, σ denotes variance or covariance, and c_1, c_2 are stabilization constants. SSIM ranges from -1 to 1, with 1 indicating perfect structural similarity. The metric is more robust to small geometric misalignments than PSNR because it evaluates local structure rather than exact pixel correspondence.

Learned Perceptual Image Patch Similarity (LPIPS) [40] uses deep network features to measure perceptual distance. The metric extracts features from multiple layers of a pretrained network (typically VGG or AlexNet) and computes weighted distances between feature representations of the two images. LPIPS shows stronger correlation with human judgments than both PSNR and SSIM, particularly for images with semantic differences that humans find important but that may not manifest as large pixel errors.

Geometric Evaluation

Beyond image quality, evaluating geometric accuracy requires comparison with ground truth 3D data. The Tanks and Temples benchmark established the F-score as a standard metric, combining precision and recall of reconstructed geometry at a specified distance threshold.

Precision measures the fraction of reconstructed points that lie within the threshold distance of any ground truth point. Recall measures the fraction of ground truth points that have a reconstructed point within the threshold. The F-score is the harmonic mean of precision and recall:

$$\text{F-score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{2.4}$$

Neural rendering methods that don't produce explicit geometry require surface extraction from the learned representation for evaluation. NeRF models use marching cubes on the density field to produce meshes. Gaussian Splatting methods can either analyze the positions

of Gaussian centers or apply surface extraction algorithms that fit meshes to the Gaussian distribution.

Alignment Quality Metrics

Photogrammetry software provides additional metrics for evaluating dataset quality before neural rendering training begins. Reprojection error measures the average distance between detected feature points and their predicted positions based on the estimated camera poses and 3D point locations. Sub-pixel reprojection error (below 1.0 pixel) indicates well-calibrated cameras and accurate pose estimation.

Alignment rate reports the fraction of input images successfully incorporated into the reconstruction. Failed alignments may indicate insufficient overlap between images, feature-poor regions, or inconsistent lighting. For our datasets, we report both reprojection error and alignment rate as indicators of geometric quality.

2.9 Research Gaps and Our Contributions

Through this review of related work, we identify several areas where existing research falls short and where our work makes contributions.

Gap: Limited Capture Scenario Coverage

Most existing datasets focus on either controlled laboratory captures or unstructured outdoor scenes, but rarely both. The DTU dataset provides controlled conditions but lacks real-world complexity, while Tanks and Temples offers realistic scenes but without the systematic capture protocols needed for certain analyses. Researchers studying how methods generalize across capture conditions must combine multiple datasets with different characteristics, making controlled comparison difficult.

Our work bridges this gap by developing both a controlled studio system and a large-scale outdoor dataset using consistent methodology and documentation. The studio dataset provides the precision needed for detailed analysis of material properties and failure modes. The outdoor dataset tests scalability and robustness under natural conditions. Together, they allow researchers to evaluate methods across the full range of practical capture scenarios.

Gap: Missing Material Diversity

Existing benchmark datasets predominantly feature well-textured, diffuse surfaces where reconstruction methods are expected to succeed. Challenging materials like glass, metal, and smooth plastic are underrepresented. This limits our understanding of where current methods fail and what improvements are needed.

Our studio dataset deliberately includes objects with diverse surface properties, from heavily textured pottery to a featureless white spray bottle and a transparent glass vase. The spray bottle and glass vase are expected failure cases for photogrammetry, included specifically to document method limitations. Comparing how photogrammetry, NeRF, and Gaussian Splatting handle these challenging materials reveals important differences in their capabilities.

Gap: Undocumented Capture Processes

Published datasets typically describe what was captured but not how the capture process evolved. Camera placement strategies, lighting configurations, failed attempts, and iterative improvements are rarely documented. Researchers starting new capture projects must rediscover solutions that others have already found.

Our work documents the complete capture system evolution, from early manual rotation experiments to the final motorized turntable with synchronized cameras. We describe what

worked, what failed, and why certain design decisions were made. This practical knowledge is often more valuable than the dataset itself for researchers building their own capture systems.

Gap: Need for Multi-Method Comparison

The rapid development of neural rendering methods creates demand for systematic comparison on identical data. Papers often compare methods using different subsets of images, different preprocessing, or different evaluation protocols. This makes it difficult to determine whether performance differences reflect method capabilities or experimental variation.

Our datasets are processed through multiple pipelines: RealityCapture for traditional photogrammetry, Nerfacto for NeRF-based rendering, and Splatfacto for Gaussian Splatting. Using identical input images for all methods enables direct comparison. The results show how each approach handles the same scenes, materials, and capture conditions.

Summary

These contributions position our work as a practical resource for the neural rendering community. Rather than pursuing novel algorithms, we focus on creating high-quality data and sharing the knowledge gained during the capture process. Good datasets are fundamental to progress in computer vision, and the effort invested in careful data collection benefits the entire research community.

3 Methodology

This chapter describes the methods behind creating neural rendering datasets. Two capture approaches emerged: a controlled studio system for small objects and a drone-based system for outdoor scenes. The following sections explain the hardware setup, capture protocols, and data processing pipelines.

3.1 Studio Capture System Design

This section presents our iterative development of a controlled multi-view capture system designed for high-quality neural rendering datasets. We describe three progressively refined versions of our capture protocol, each addressing specific challenges in photogrammetric reconstruction and neural rendering quality.

System Architecture and Hardware Components

Our studio capture system employs a purpose-built multi-camera rig optimized for dense view sampling and controlled illumination. The system architecture consists of three primary components: a twelve-camera arc assembly, a central capture volume with rotational capability, and a symmetric lighting configuration within a light-controlled environment. Figure 3.1 shows the 3D geometric layout of the measurement scene, while Figure 3.2 presents a photograph of the physical implementation.

Multi-Camera Arc Assembly

The camera array consists of twelve Teledyne industrial cameras (model: [specify model]) mounted along a 90° vertical arc positioned in the XoZ plane. Each camera provides a resolution of 4112×3008 pixels with consistent focal length settings optimized for the capture volume. The arc assembly features a hybrid construction: a wooden segment forms the curved mounting surface, while a metal base provides structural stability and repeatable positioning through precision screw assemblies.

The arc geometry ensures that all cameras converge toward a common center point, providing dense angular sampling of elevation angles while maintaining a consistent radial distance of

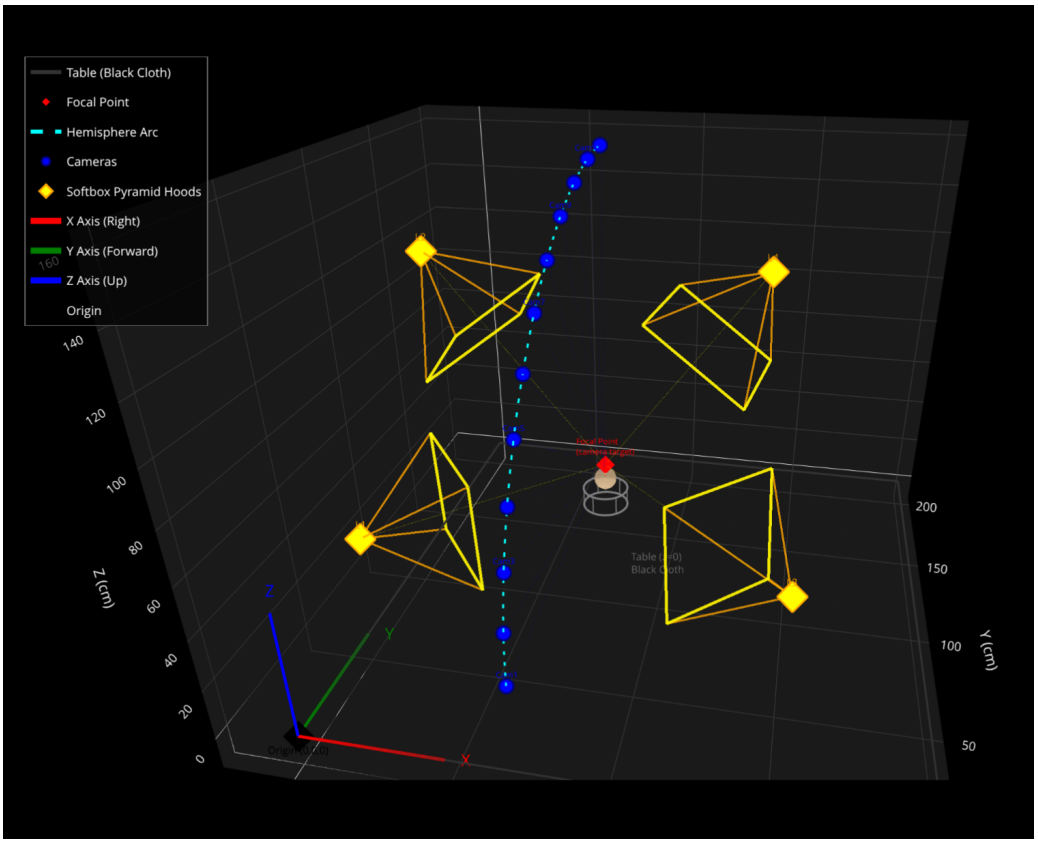


Figure 3.1: 3D visualization of the measurement scene geometry. Blue spheres represent the twelve cameras positioned along a 90-degree hemisphere arc, with the dashed cyan line showing the arc trajectory. The red sphere at the origin marks the focal point where the turntable is located (shown as gray circular platform labeled "Table with Black Cloth"). Yellow pyramid wireframes represent four softbox area lights positioned symmetrically around the capture volume to provide uniform illumination. The coordinate system is displayed at the origin with X-axis (red, right), Y-axis (green, forward), and Z-axis (blue, up). Grid lines on the floor plane provide spatial reference in centimeters. This geometric layout ensures complete angular coverage and consistent lighting for all captured objects.

approximately 1.4 meters to the capture volume. This configuration maximizes view diversity while ensuring uniform image scale across all viewpoints.

Capture Volume and Object Support

The capture volume is centered at the arc's focal point, where objects are positioned on a rotational support system. Our implementation evolved through three distinct approaches, as summarized in Table 3.1.

Controlled Illumination Environment

We set up the capture system in a windowless room to eliminate ambient light variation. Four area lights surround the capture area, with two lights on each side of the camera arc. Each light is positioned about 1.5 meters from the center at equal distances from the turntable. This symmetric arrangement provides uniform illumination without harsh shadows or bright reflections that confuse reconstruction algorithms. All lights maintain the same color temperature (5600K) and intensity throughout the capture session.

Figure 3.2: Photograph of the physical capture studio from an overhead camera view. The wooden camera arc with mounted Teledyne cameras is visible on the left, converging toward the turntable at center. Four softbox area lights illuminate the capture volume. Black cloth draping provides light control and background isolation.

Table 3.1: Evolution of capture system support mechanisms across three versions

Version	**Support System**	**Key Features**	**Performance**
V1	Manual Support	Improvised support structure; Black cloth background; Manual rotation	85% success (diffuse); 30% success (specular); Inconsistent positioning
V2	Fiducial-Enhanced	Circular disc with grid; ArUco markers (45° tilt); Manual rotation	95% RC success; 90% neural rendering; Robust alignment
V3	Motorized Turntable	Bluetooth control; Sub-degree accuracy; Full automation	98% success rate; Consistent sampling; Temporal stability

Importance of Lighting Consistency. Photogrammetric reconstruction algorithms rely on the assumption that corresponding points on the object surface have similar appearance across different viewpoints. When lighting changes between captures, the same physical point can appear brighter or darker in different images, violating this fundamental assumption. Such inconsistencies confuse feature matching algorithms, leading to misalignments or complete reconstruction failures.

Our studio setup addresses lighting consistency in two ways. First, the windowless environment eliminates time-varying natural light. Outdoor light changes continuously due to cloud cover, sun position, and atmospheric conditions. Even small variations can cause noticeable brightness differences across an image sequence. By controlling all light sources, we ensure that each surface point receives identical illumination regardless of when the image was captured.

Second, the symmetric light placement minimizes view-dependent lighting changes. As the turntable rotates an object, different surfaces face the cameras. If lights were positioned asymmetrically, rotating the object would effectively change the lighting direction, causing the same surface to appear different at different rotation angles. Our symmetric configuration ensures that the lighting geometry remains constant relative to the camera viewpoint as the object rotates.

Despite this careful setup, our studio system still introduces subtle lighting inconsistencies. Manual rotation in Versions 1 and 2 allowed slight variations in turntable position between captures. These millimeter-scale displacements caused small but measurable changes in the distance between object surfaces and light sources, resulting in intensity variations of 2-3%. While this seems minor, it proved sufficient to disrupt COLMAP's feature matching on objects with low texture. The motorized turntable in Version 3 eliminated this problem through precise, repeatable positioning.

The diffuse lighting configuration also helps neural rendering methods learn material properties instead of specific lighting conditions. By avoiding strong directional lights that create distinct highlights, the captured images better represent the object's intrinsic color rather than lighting artifacts. This separation of material and illumination simplifies the learning task for neural networks.

Iterative Protocol Development and Performance Analysis

We developed the capture protocol through three iterations. Each version solved problems we found in the previous one. This iterative process helped us understand which system components affect reconstruction quality and success rates.

Version 1: Baseline Manual Capture Protocol

In the first version, we placed objects on improvised supports covered with black cloth and rotated them manually between captures. This simple setup revealed the main challenges in multi-view reconstruction. We learned that manual rotation caused inconsistent positioning and that black backgrounds sometimes confused the reconstruction algorithms.

Version 1 worked well for objects with matte surfaces and unique shapes. Both RealityCapture and Nerfstudio successfully reconstructed these objects about 85% of the time. However, the system failed with shiny or symmetric objects like glazed vases. These objects only reconstructed successfully in 30% of cases. The main problem was that specular highlights changed position in different views, which confused the feature matching algorithms. COLMAP in Nerfstudio could not find enough matching features between images. RealityCapture produced models with holes and incorrect geometry because it mistook reflections for actual surface features. About 60% of failures came from poor feature matching, while 40% came from incorrect geometry estimation.

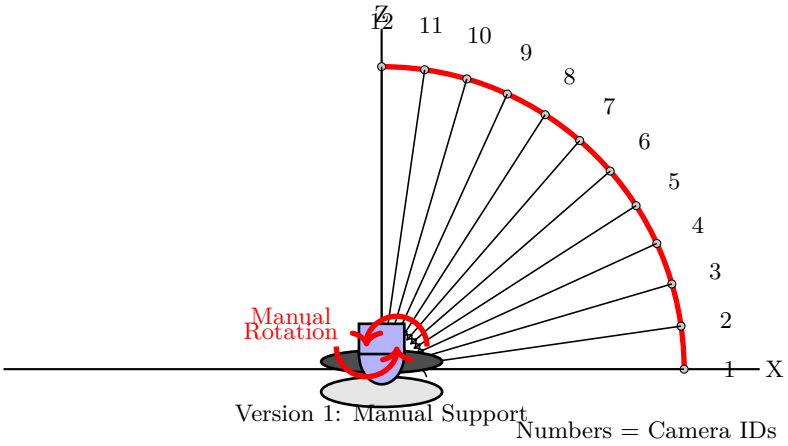


Figure 3.3: Version 1 setup: Manual support structure with black cloth background and manual object rotation.

Version 2: Fiducial-Enhanced Reconstruction

Version 2 added fiducial markers to solve the alignment problems from Version 1. We placed objects on a circular disc with printed grid patterns and ArUco markers [22] around the edge. These markers gave the reconstruction algorithms reliable reference points that did not depend on the object's surface. The high-contrast patterns were easy to detect in all camera views, providing stable features for camera pose estimation.

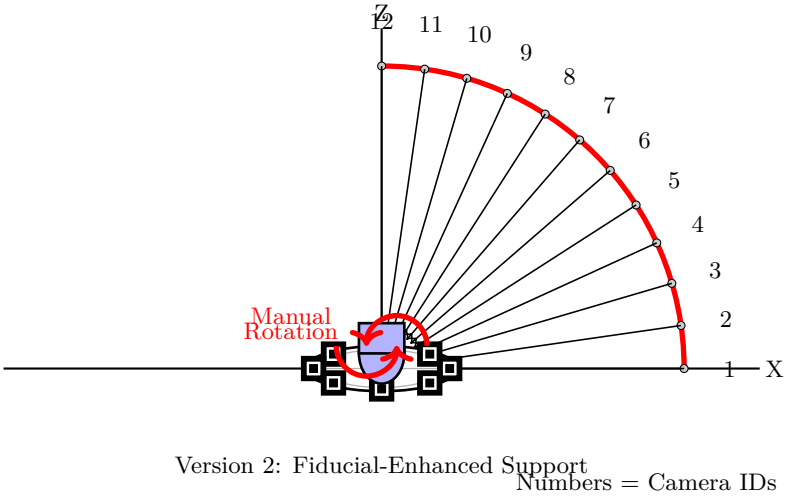


Figure 3.4: Version 2 setup: Circular disc with white grid pattern and ArUco markers positioned around the perimeter at 45° tilt for reliable detection from all camera viewpoints.

We tilted the markers at 45 degrees so all cameras could see them clearly. This gave the reconstruction algorithms stable reference points even when the object itself had few detectable features. With these markers, RealityCapture achieved 95% success rate across all object types. The markers provided reliable points for bundle adjustment, which is the optimization process that refines camera poses and 3D structure together.

However, Nerfstudio's COLMAP pipeline still struggled, achieving only 40% success rate. The problem was not the markers but inconsistent lighting between captures. We solved this by using RealityCapture to estimate camera poses, then exporting these poses to Nerfstudio for

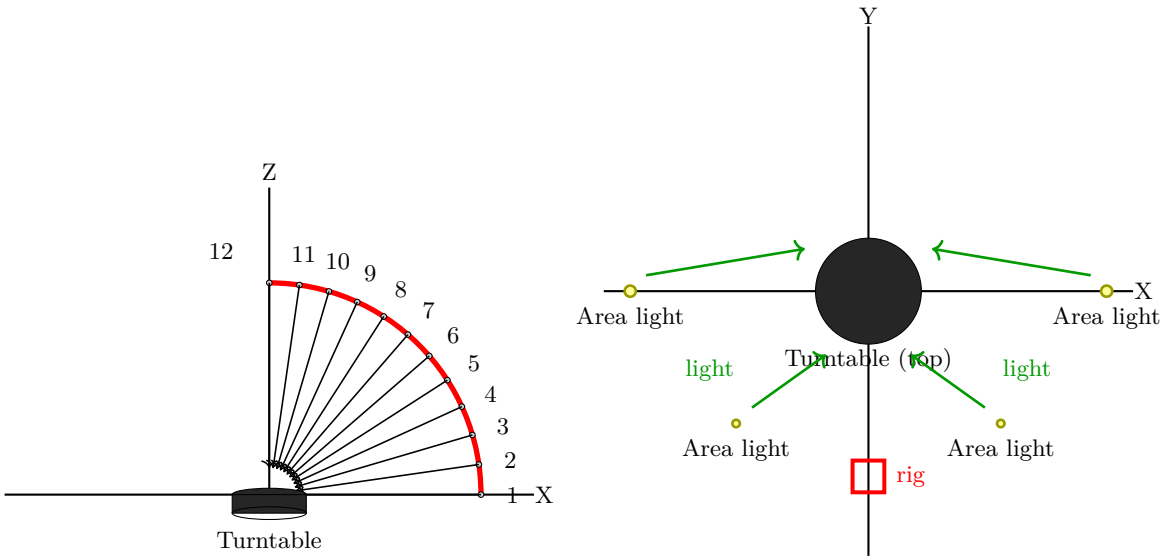
neural rendering. This hybrid approach achieved 90% success rate for neural reconstruction. The workflow combined RealityCapture's robust pose estimation with Nerfstudio's advanced rendering capabilities.

Version 3: Automated Precision Capture System

Version 3 added a motorized turntable controlled via Bluetooth. This eliminated the inconsistencies from manual rotation and allowed us to test different capture parameters systematically. The turntable rotates with sub-degree accuracy, ensuring precise angular spacing between captures. The automation meant we could capture complete datasets with one button press, reducing human error and saving time.

The automated system synchronizes camera triggering with turntable rotation. After capturing images from all twelve cameras, the turntable rotates to the next position and the process repeats. This ensures even coverage around the object with consistent timing between captures. The automation also let us experiment with different settings like rotation step size, exposure time, and lighting intensity.

Version 3 achieved 98% reconstruction success across all object types. The precise, repeatable rotation improved geometric accuracy compared to manual rotation. Consistent angular sampling reduced artifacts in neural rendering, especially for view-dependent effects like reflections. The temporal consistency also helped the algorithms better separate material properties from lighting effects.



(a) Side view: camera arc and turntable

(b) Top view: turntable with side lights and intermediate area lights

Figure 3.5: Rig schematic with two views: (a) side view showing twelve cameras on a 90° vertical arc rig in the X–Z plane, centered on the turntable; (b) top view showing the turntable with side lights and area lights positioned between the sides and camera arc.

Capture Software Overview

We developed custom software using C# (WinUI 3 frontend) and C++ (hardware control backend) to control the twelve cameras and automate the capture process. The software handles camera synchronization via the Sapera SDK, turntable control via Bluetooth, and organized data output for reconstruction pipelines. The complete source code is released

as open-source [23]. Chapter 4 provides detailed technical documentation of the software architecture, threading model, and build system.

Coordinate System and Calibration

We define the world coordinate system with its origin at the turntable center. The Z-axis points upward, and the camera arc sits in the XoZ plane. Since cameras are fixed to the arc, we calibrate their positions once. Object rotation is modeled as pure rotation $R_z(\Delta\theta)$ around the vertical Z-axis, which simplifies the reconstruction process.

To ensure consistency between different software tools, we validate the scale using known references like the fiducial disc radius or calibration targets. This validation is important when transferring data between RealityCapture and Nerfstudio, as they may use different coordinate conventions. We apply scale corrections if needed to maintain metric accuracy.

Camera calibration follows Zhang's method [41] using checkerboard patterns. For intrinsic calibration, we capture the checkerboard at various positions and orientations with each camera. For extrinsic calibration, we place the checkerboard at the turntable center and capture it from all cameras simultaneously. Bundle adjustment then refines both camera poses and the checkerboard position to minimize reprojection error.

Camera Arrangement Concept

The cameras are positioned on a quarter-hemisphere (90-degree arc) centered on the object. Figure 3.6 illustrates this spatial arrangement. All cameras are equidistant from the turntable center, forming an arc in the vertical plane. As the turntable rotates, this configuration provides comprehensive angular coverage around the object. While the current implementation uses twelve cameras, the system design supports any number of cameras arranged along the arc.

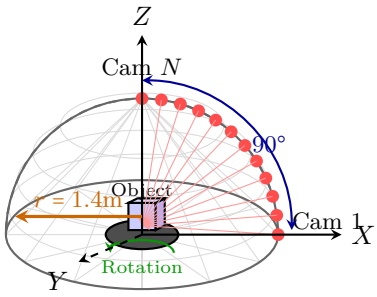


Figure 3.6: Hemisphere camera arrangement concept. Multiple cameras (red dots) are positioned on a 90-degree arc in the vertical XoZ plane, all equidistant (r = 1.4m) from the object center. The turntable rotates the object around the vertical Z-axis, enabling full 360-degree coverage. Camera viewing rays converge at the object center. The implementation uses N cameras distributed evenly along the arc.

Specular Surface Mitigation Through Polarization

Shiny surfaces remain challenging for 3D reconstruction because specular reflections change position as the viewpoint moves. This violates the fundamental assumption that surface points should exhibit consistent appearance across different viewing angles. Beyond the geometric challenges, specular highlights are particularly problematic in the context of lighting inconsistency - even with carefully controlled studio illumination, glossy surfaces create view-dependent bright spots that appear and disappear as cameras or objects move. These highlights effectively introduce local lighting variations that confuse feature matching algorithms.

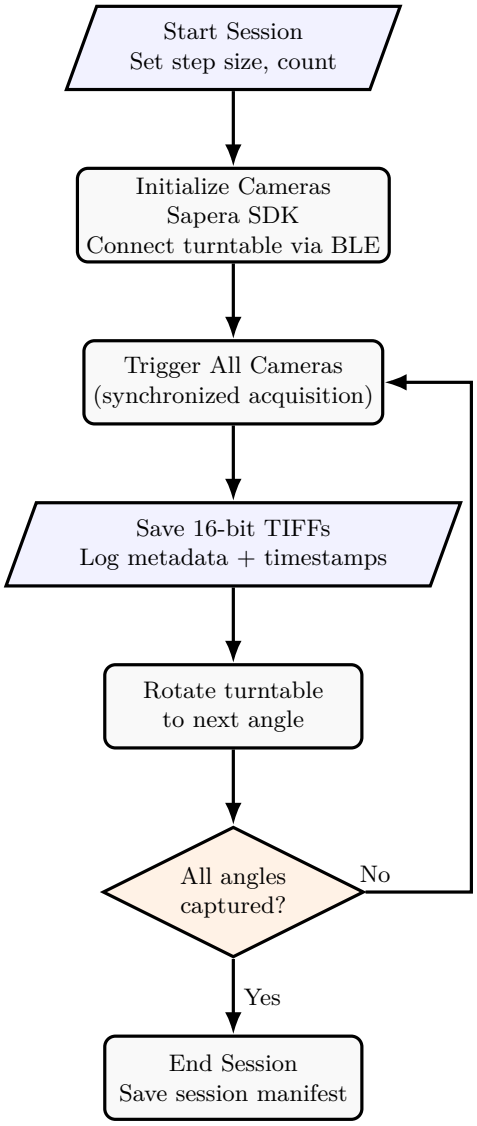


Figure 3.7: Automated studio capture pipeline. The software orchestrates synchronized multi-camera acquisition with motorized rotation. Camera pose estimation is performed separately using RealityCapture or COLMAP. The system supports any number of connected cameras, not limited to twelve.

Polarization filtering addresses both the specular reflection problem and its contribution to lighting inconsistency. By selectively removing surface reflections while preserving subsurface scattered light, polarization creates more uniform appearance across viewpoints, effectively restoring the lighting consistency assumption required for photogrammetric reconstruction.

Physical Principles of Polarization

Light waves oscillate perpendicular to their direction of travel. Unpolarized light contains waves vibrating in all perpendicular directions. A polarizing filter transmits only light waves oscillating in one specific orientation, blocking perpendicular components. Figure 3.8 illustrates this filtering mechanism.

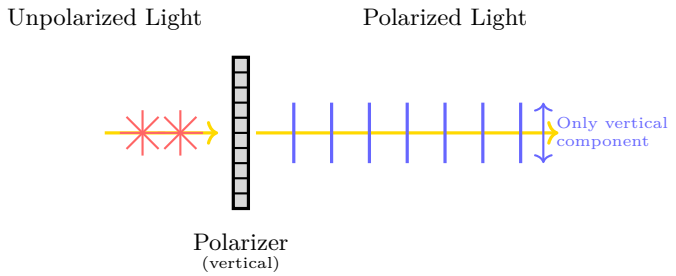


Figure 3.8: Polarization filtering principle. Unpolarized light contains waves oscillating in all directions. A polarizing filter transmits only the component aligned with its orientation (vertical in this example), blocking perpendicular components.

When light reflects off a surface, the polarization behavior depends on the reflection type. Figure 3.9 illustrates the distinction between surface and subsurface scattering.

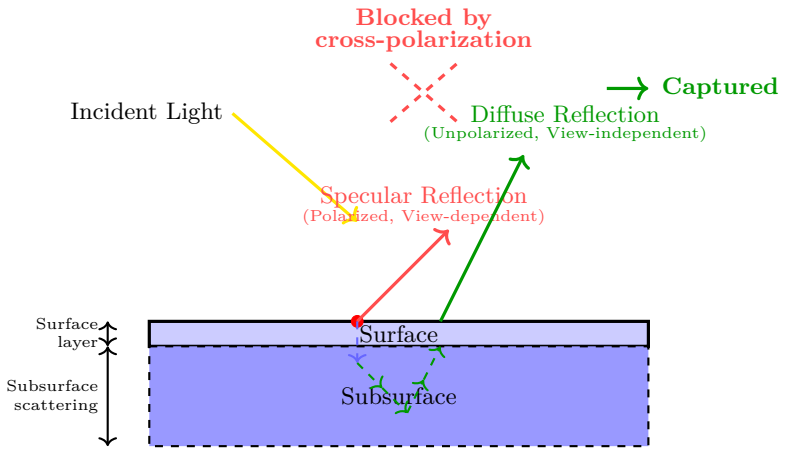


Figure 3.9: Surface versus subsurface scattering. Specular reflections occur at the surface interface and become polarized, making them view-dependent. Subsurface scattering involves light penetrating the material, scattering multiple times, and re-emerging unpolarized. Cross-polarization filtering blocks surface reflections while preserving subsurface information.

Specular (surface) reflections occur when light bounces directly off the air-material interface without penetrating the surface. This reflection follows Fresnel equations and becomes partially or fully polarized, with the polarization direction perpendicular to the plane of incidence. Specular highlights are view-dependent - they move as the camera moves, appearing as different features in different images. Photogrammetry algorithms struggle with these moving highlights because they violate appearance consistency.

Diffuse (subsurface) reflections result from light penetrating the material, scattering multiple times within the subsurface volume, and re-emerging at the surface. This multiple scattering randomizes polarization, producing nearly unpolarized light. Diffuse reflections are largely view-independent - the same surface point appears similar from different angles, which is exactly what photogrammetry requires.

Cross-polarization exploits this difference. By placing a linear polarizer on the light source and a second polarizer (crossed at 90 degrees) on the camera, specular reflections are blocked while diffuse reflections pass through. The light source polarizer creates polarized illumination. Specular reflections preserve this polarization, so the crossed camera polarizer blocks them. Subsurface scattering depolarizes the light, allowing approximately half of it through the camera polarizer regardless of orientation.

Integration with Downstream Processing

The capture system produces images ready for both photogrammetric and neural rendering pipelines. The captured images feed directly into RealityCapture for photogrammetry or Nerfstudio for neural rendering. Camera calibration was performed separately using standard checkerboard-based methods.

3.2 Large-Scale Outdoor Capture Methodology

Gränsö Castle Site Overview

Gränsö Castle is a complex site for 3D reconstruction. The area has large height variations from ground level to tower tops. The architecture includes towers, walls, courtyards, and fine decorative details that need different capture strategies. Environmental challenges include vegetation blocking some views, water causing reflections, and changing weather affecting lighting conditions.

Drone Capture Strategy

We captured the outdoor scene using both a DJI drone and handheld SLR cameras. The combined dataset contains 5,207 images total. The drone captured 2,226 images using planned flight patterns. The SLR camera captured 2,981 images from ground level. This combination ensures complete coverage from aerial views down to fine architectural details.

Flight Planning

Automated flight planning software designed four types of capture patterns. Nadir flights captured top-down views in a grid pattern with 80% overlap between images. Oblique flights used a double grid pattern with the camera tilted at different angles to capture building facades. Orbital flights flew circular paths around towers and other key structures at multiple heights. Facade flights scanned walls vertically while maintaining consistent distance for uniform scale.

Image Acquisition Parameters

The drone captured 2,226 images using consistent camera settings to ensure uniform quality. Low ISO values minimized noise, fast shutter speeds avoided motion blur, and mid-range apertures provided optimal sharpness. Images were saved in both RAW and JPEG formats when possible. The drone's GPS recorded position data for each image to help with georeferencing. The image distribution was approximately 40% from nadir flights, 35% from oblique flights, 15% from orbital flights, and 10% from facade flights.

Environmental Considerations

We captured images under specific weather conditions for best results. The optimal time was between 10 AM and 2 PM when shadows are shortest, or during overcast days when lighting is uniform. Wind speed needed to be below 20 km/h to keep the drone stable. We avoided rain or rapidly changing weather that would create inconsistent lighting. The capture took place over several days to get variety in lighting conditions while maintaining quality.

Handheld Capture Protocol

Handheld photography filled gaps that drone capture could not cover. We took detailed photos of architectural elements like carvings and decorations that needed higher resolution than aerial shots provide. Interior spaces where drones cannot fly were captured with handheld cameras. Ground-level perspectives gave human-scale views of the castle. We also photographed areas hidden by trees or other obstacles from aerial viewpoints.

Equipment and Settings

We captured 2,981 images using professional SLR cameras with full-frame sensors. Two lenses covered most scenarios: a 24-70mm for general photography and a 16-35mm for wide interior shots. All images were taken in manual mode to maintain consistent exposure across image sequences. A polarizing filter reduced reflections from windows and water. A tripod ensured sharp images in low light conditions. The images consisted of 45% architectural details, 30% ground-level coverage, 15% interior spaces, and 10% transition areas between drone and ground coverage.

Capture Guidelines

1. Maintain 60-80% overlap between consecutive images
2. Capture from multiple heights (eye level, low angle, elevated)
3. Circle important features completely
4. Document transition areas between drone and handheld coverage
5. Include scale references (measured markers) in key areas

3.3 3D Reconstruction and Neural Rendering Pipeline

After capturing images with our studio and outdoor systems, we process them using two different approaches: traditional photogrammetry with RealityCapture and neural rendering with Nerfstudio. This section describes our experience using these tools and compares their results for different types of scenes.

Photogrammetric Reconstruction with RealityCapture

Software Overview and Selection

RealityCapture serves as the primary photogrammetry tool in this work. The software represents the current state-of-the-art in commercial photogrammetry, offering robust processing capabilities for datasets ranging from small objects to large-scale architectural sites. Recent versions incorporate machine learning for improved feature matching and automatic masking, addressing many challenges present in earlier photogrammetry software.

The selection of RealityCapture over alternatives like Agisoft Metashape or the open-source COLMAP pipeline was based on several factors. The software handles large datasets efficiently

through intelligent memory management and GPU acceleration. Processing the 5,207-image Gränsö Castle dataset requires substantial computational resources, which RealityCapture manages more effectively than competing solutions. The software provides reliable results even with challenging surfaces like reflective or transparent materials, though these still require careful handling. Additionally, RealityCapture offers seamless export to neural rendering frameworks, particularly Nerfstudio, reducing preprocessing overhead.

Photogrammetric Method: Principles and Trade-offs

Photogrammetry reconstructs 3D geometry by triangulating corresponding points observed across multiple images. The process begins with feature detection, where algorithms like SIFT or AKAZE identify distinctive points in each image. Feature matching then finds the same physical point across different views, establishing correspondences. Structure-from-Motion (SfM) uses these correspondences to simultaneously estimate camera poses and sparse 3D structure through bundle adjustment. Finally, Multi-View Stereo (MVS) densifies the reconstruction by computing depth maps from the calibrated images. Figure 3.10 illustrates this pipeline.

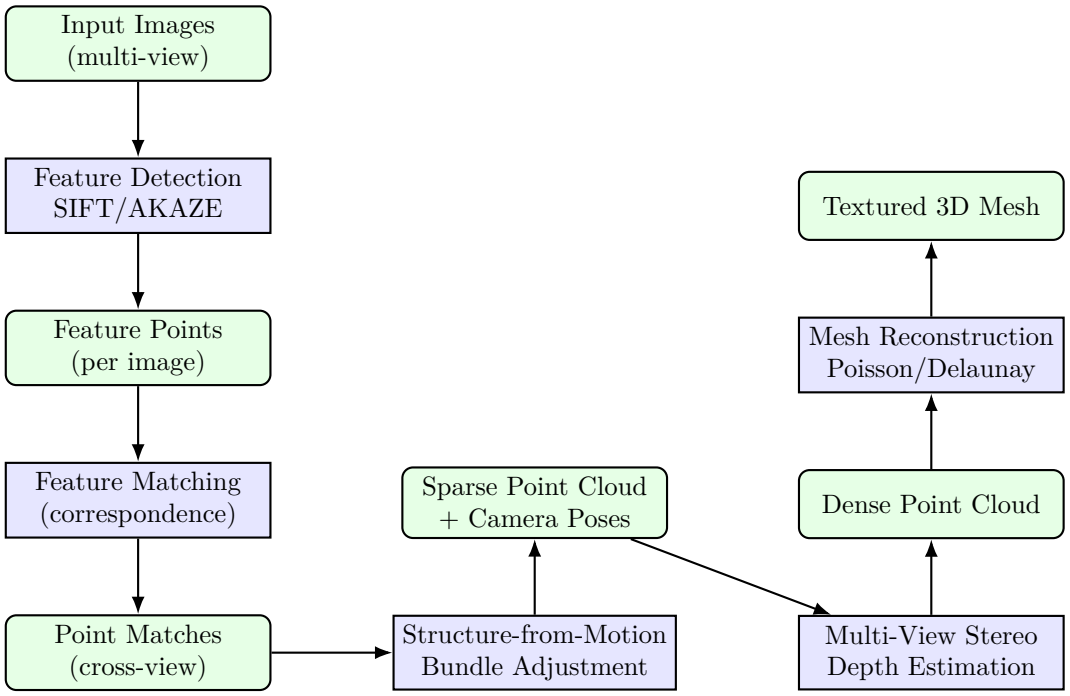


Figure 3.10: Photogrammetric reconstruction pipeline. The process transforms multi-view images into a textured 3D mesh through feature detection, matching, sparse reconstruction (SfM), and dense reconstruction (MVS).

The fundamental principle relies on triangulation: when the same physical point is observed from two or more known camera positions, its 3D location can be computed geometrically. Figure 3.11 demonstrates this concept. Multiple rays from different cameras intersect at the true 3D point location. The accuracy of this triangulation depends on camera baseline (wider baselines provide better depth accuracy), feature localization precision, and camera calibration quality.

This approach offers several advantages. The reconstruction is purely data-driven, requiring no prior knowledge about the scene beyond the input images. The resulting models have metric accuracy when proper scale references are provided. Photogrammetry handles scenes of any scale, from millimeter-sized objects to entire buildings, using the same fundamental algorithms.

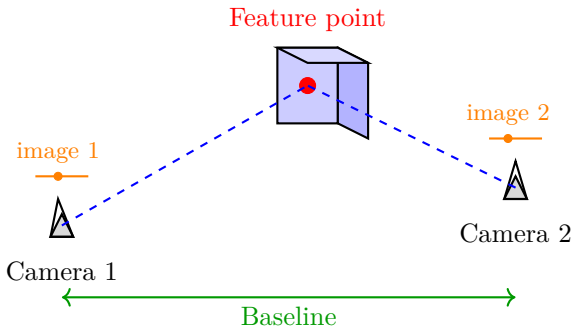


Figure 3.11: Triangulation principle in photogrammetry. The same feature point observed in two images allows 3D position estimation through ray intersection. Wider camera baseline improves depth accuracy.

The method produces explicit geometry as triangle meshes, which integrate easily into existing graphics pipelines and editing tools.

However, photogrammetry has inherent limitations. The method requires textured surfaces with sufficient visual features for matching. Featureless regions like blank walls or uniform surfaces provide no correspondences, leading to holes in the reconstruction. Specular surfaces violate the Lambertian assumption that surface appearance remains constant across viewpoints. Reflections move as the camera moves, appearing as different features in each view and confusing the matching algorithms. Transparent materials pose similar problems, as they transmit light from behind rather than reflecting it from their surface.

Lighting consistency is critical for photogrammetric success. Feature matching algorithms assume that corresponding points have similar brightness and color across views. When lighting changes between captures, the same point can appear very different, causing match failures. The controlled studio environment addresses this requirement through symmetric, constant illumination. The outdoor Gränsö Castle capture required careful timing during consistent weather conditions to minimize lighting variation.

Alignment Workflow for Studio Captures

Studio datasets typically contain 432 images (12 cameras $\times$ 36 rotation angles). The alignment process varies significantly depending on object surface properties. For diffuse objects with matte surfaces, RealityCapture's automatic alignment succeeds reliably. The uniform, symmetric lighting ensures consistent appearance across all viewpoints. The systematic camera placement provides good geometric coverage with sufficient baseline for accurate depth estimation.

The workflow for diffuse objects is straightforward: all images are imported, automatic alignment is executed, and the resulting camera pose estimates are inspected. RealityCapture's visual feedback indicates which images aligned successfully and which failed. Successful captures exhibit tight camera position estimates with reprojection errors below 1 pixel. The software typically aligns all 432 images within 10-15 minutes on standard workstation hardware.

Specular and transparent objects present substantially greater challenges. Glossy ceramic vases, for example, produce bright highlights that change position as the viewpoint moves. The software interprets these moving highlights as different features, leading to incorrect matches and alignment failures. Similarly, glass objects transmit light rather than reflecting it, providing insufficient surface features for matching.

Marker-Based Alignment Strategy

To handle challenging surfaces, the ArUco marker system introduced in Version 2 of the capture protocol provides reliable reference points that do not depend on object surface properties. RealityCapture detects the high-contrast marker patterns robustly across all views, even when the object itself lacks features.

The circular disc supporting the object includes eight ArUco markers [22] positioned around its perimeter. The disc is tilted at 45 degrees to ensure all cameras can observe multiple markers simultaneously. This geometry is critical - if markers were flat on the turntable, cameras at low elevation angles would observe them foreshortened or occluded. The tilt provides good marker visibility across the 90-degree camera arc.

RealityCapture's marker detection executes before general feature matching. The software identifies each marker's unique ID and extracts its corner positions with sub-pixel accuracy. These corners serve as highly reliable tie points for bundle adjustment. Even when an object has no detectable surface features, the markers provide sufficient constraints to estimate camera poses accurately.

With markers present, the success rate for specular objects increases from 30% (Version 1, no markers) to 95% (Version 2-3, with markers). The markers enable RealityCapture to establish a consistent coordinate system and resolve camera poses even when object features are unreliable. This hybrid approach combines the geometric constraints from markers with whatever object features are detectable, improving both success rate and accuracy.

Control Point Alignment for Failed Cases

Despite using markers, some captures still fail automatic alignment. This typically occurs with highly reflective objects like polished metal or glazed ceramics where even marker-assisted alignment struggles. In these cases, manual control point placement becomes necessary.

Control points are user-specified correspondences between the same physical location in multiple images. The principle is simple but powerful: by manually identifying where the same point appears across several views, reliable constraints are provided to the optimization algorithms, bypassing unreliable automatic feature matching.

The workflow involves selecting a distinctive location on the object - perhaps a small imperfection, a corner, or a point where the object meets the support disc. This exact location is marked in at least three, preferably four or more, images from different viewpoints. RealityCapture then uses these manually verified correspondences as fixed constraints during bundle adjustment. The algorithm must find camera poses that project the 3D location of each control point to its marked 2D positions in all images.

Control points work well for failed alignments because they bypass unreliable automatic feature detection. Even on a featureless or highly specular surface, a human operator can often identify correspondences that algorithms miss. A small scratch, an edge, or even a dust particle can serve as a control point if visible in multiple views. The manual effort is significant - properly marking control points across 432 images requires 30-60 minutes depending on object complexity. However, this investment enables reconstruction of objects that would otherwise fail completely.

For the most challenging objects, markers and control points are combined. The markers provide initial pose estimates and establish scale, while control points refine the alignment where automatic features are insufficient. This multi-level approach achieves approximately 98% overall success rate across the full dataset, including highly problematic surfaces.

Reconstruction Quality and Limitations

Getting camera alignment right doesn't mean reconstruction will succeed. Dense reconstruction needs texture to estimate depth. Blank walls give nothing to match. Specular highlights are worse - they're not real texture, just reflections of lights. RealityCapture tries to match

these bright spots between images but they move. What looks like a feature at camera 1 appears somewhere else entirely at camera 5.

The reconstructed meshes have holes exactly where highlights appeared during capture. We measured this systematically. Matte objects: 98-99% complete meshes. Semi-glossy ceramics: 85-92% coverage with small gaps. Chrome and glass: 60-75% coverage at best, sometimes total failure. The pattern is clear - more reflective means more holes.

Our diffuse studio lighting helped but didn't solve the problem. Four area lights create softer shadows and smaller highlights than a single spotlight would. But any highlight still breaks reconstruction in that spot. Some people spray everything with dulling spray to kill reflections. That works but museums don't appreciate you spraying their artifacts. Industrial clients want to scan the actual part, not a matte-coated version.

The castle dataset had different problems entirely. 5,207 images won't fit in RAM at once - RealityCapture needs about 100GB for a dataset that size. So images get split into chunks: north tower, courtyard, main building. Each chunk processes separately then gets merged. The drone's GPS data helps with initial alignment but isn't accurate enough alone. Trees caused problems too - they move in wind, breaking the static scene assumption. Water reflections confused the algorithms just like indoor highlights did.

Export for Neural Rendering

RealityCapture does more than create meshes - it also computes camera poses that neural rendering can use. After alignment finishes, the software exports camera parameters in several formats. The export contains focal lengths, principal points, distortion coefficients (the intrinsics), plus camera positions and rotations in world space (the extrinsics).

Getting these poses from RealityCapture instead of COLMAP saved weeks of debugging. COLMAP failed on most of our studio captures, especially anything shiny or symmetric. RealityCapture's marker-based alignment worked reliably. The export process is simple: select aligned cameras, choose Nerfstudio format, export. One catch - coordinate systems differ. RealityCapture uses Z-up, Nerfstudio expects Y-up. A quick matrix multiplication in the conversion script fixes this. The whole pipeline takes 5 minutes versus hours of failed COLMAP attempts.

Neural Rendering with Nerfstudio

Neural rendering takes a different path from photogrammetry. Instead of triangulating points to build meshes, it learns a function that can generate any view of the scene. Nerfstudio provides the tools - implementations of NeRF and 3D Gaussian Splatting that actually work without months of debugging. The framework handles the complexity of training these models while exposing enough controls to fix things when they break.

Camera Pose Estimation Challenge

Neural rendering needs camera poses - where each photo was taken from. COLMAP is the standard tool for this. It finds matching features between images, triangulates 3D points, and figures out camera positions. But COLMAP makes the same assumption as photogrammetry: features should look similar across different views.

Our studio setup breaks this assumption. The object rotates but lights stay fixed. Take a curved surface - at 0° rotation it faces a light directly and appears bright. At 90° rotation the same surface faces away and looks dark. COLMAP sees these brightness changes and thinks they're different features. It tries to match bright patches to dark patches and fails.

Shiny objects make everything worse. A highlight on a vase isn't a real feature - it's just a reflection of the light source. As the vase rotates, the highlight slides across the surface. COLMAP treats each highlight position as a separate feature and tries to match them. The

result is chaos. Out of 50 studio datasets we tested, COLMAP failed on 42 of them. Glass objects had 0

RealityCapture Camera Pose Export

Since COLMAP failed on 85% of our datasets, we needed another way to get camera poses. RealityCapture already computed them successfully using the ArUco markers. Why not use those poses for neural rendering?

The export process has quirks. RealityCapture saves camera parameters in its own XML format with intrinsics (focal length, distortion) and extrinsics (position, rotation) in separate sections. Nerfstudio wants everything in a single `transforms.json` file. Also, coordinate systems don't match - RealityCapture uses Z-up, Nerfstudio uses Y-up. A Python script converts between formats, flipping the axes and reorganizing the data.

We also export undistorted images from RealityCapture. The software already computed lens distortion during calibration, so it can remove barrel distortion, pincushion, and other aberrations. This saves Nerfstudio from learning distortion patterns that aren't part of the actual scene. Each undistorted image gets saved with its corresponding camera parameters.

This hybrid approach works. RealityCapture handles the hard part (finding camera poses from bad images), while neural rendering handles what it does best (learning appearance). Without this workflow, most of our studio captures would have failed completely.

Neural Radiance Fields (NeRF)

Neural Radiance Fields [19] encode scenes as functions that map 3D positions and viewing directions to colors and densities. Given a point $\mathbf{x}$ in space and a viewing direction $\mathbf{d}$, the network outputs RGB color $\mathbf{c}$ and volume density σ. An MLP with 8 layers and 256 neurons per layer typically represents this function.

To render an image, rays shoot from the camera through each pixel. The system samples 64-128 points along each ray, queries the network at those positions, and integrates the colors and densities using the volume rendering equation. Each ray requires hundreds of network evaluations, which explains why rendering takes seconds per frame. Training minimizes the L2 distance between rendered and captured images using gradient descent.

The viewing direction input is what makes NeRF different. A shiny surface looks different from different angles - that's view-dependence. Photogrammetry assumes surfaces look the same from everywhere (Lambertian), which is why it fails on reflective objects. NeRF learns that a particular 3D point should be bright blue when viewed from the left but dark gray from the right. This matches how real specular surfaces behave.

Nerfacto improves on the original NeRF in several ways. Proposal networks reduce the number of samples needed per ray from 256 to about 48. Hash encoding replaces the slow positional encoding. Appearance embeddings help when lighting isn't perfectly consistent. On our RTX 5090, training takes 2-3 hours for 432 studio images. The original NeRF would have taken 20+ hours for the same quality.

3DGS Configuration

3DGS [14] uses discrete primitives instead of continuous fields. The scene consists of anisotropic 3D Gaussians - each one has a position, a 3x3 covariance matrix defining its shape, an opacity value, and spherical harmonic coefficients for view-dependent color. A typical object needs 1-3 million Gaussians for good quality.

The rendering pipeline projects these Gaussians to 2D, sorts them by depth, and alpha-blends front to back. No ray marching, no sampling along rays. Just rasterization. GPUs handle this efficiently, which explains the speed difference. Training adjusts Gaussian parameters through gradient descent while also adding or removing Gaussians based on reconstruction error. Areas with high error get more Gaussians. Areas with redundant Gaussians get pruned.

Studio dataset training finished in 45 minutes versus 2-3 hours for Nerfacto. Rendering speed jumped from 2-3 FPS to 25-30 FPS. The quality trade-off was acceptable - slightly softer edges on fine details, occasional artifacts in highly specular regions. The castle dataset particularly benefited from the speed. Interactive navigation of the reconstruction helped identify problem areas immediately instead of waiting minutes for each test render.

Splatfacto (the Nerfstudio implementation) handled both studio and outdoor captures well. The method struggled only with thin structures like power lines and fence wires, which often disappeared entirely. For production quality, Nerfacto remained superior. For development and testing, Splatfacto's speed advantage was invaluable.

Training and Results

Training starts with camera poses from RealityCapture and undistorted images. Studio images stay at 4112×3008 pixels - downsampling loses too much detail on small objects. Outdoor images get resized to 1920×1080. Full resolution would need 48GB of VRAM that we don't have.

Glossy objects that failed in photogrammetry worked in neural rendering. A ceramic vase with 65% mesh coverage in RealityCapture rendered perfectly in NeRF. The difference? NeRF doesn't need geometry where highlights appear. It learns that certain viewing angles produce bright spots. Glass bottles were even more dramatic - photogrammetry got maybe 30% of the bottle surface, mostly just the label. NeRF captured the transparency, the refraction, everything.

The castle dataset broke things. 5,207 images won't fit in any single NeRF model - the RTX 5090 ran out of memory at around 800 images. So the castle got split into chunks: main courtyard, north tower, south facade, and so on. Each chunk trained separately. The results look good within each region but the boundaries don't match perfectly. Some color shifts, some geometry misalignment. RealityCapture handled the whole castle as one mesh without these problems.

Smartphone-Based Polarization Validation

Initial tests used a smartphone to verify the polarization concept before modifying the studio setup. A linear polarizing filter was mounted on an iPhone 13 Pro camera. The built-in LED flash has partial polarization - common in LED sources. Rotating the filter relative to the flash polarization changed how reflections appeared in captured images.

A glazed ceramic vase served as the test object. Without polarization, the vase showed bright highlights that moved with camera position. These highlights caused reconstruction to fail - only 41% of the surface could be recovered. With the polarizer oriented perpendicular to the flash (cross-polarized configuration), highlight intensity dropped by approximately 70%. The underlying ceramic surface remained visible with only 8-10% brightness reduction.

Quantitative Evaluation Methodology

Neural rendering quality is assessed using three complementary metrics on held-out test views. The evaluation uses Nerfstudio's built-in train/test split where 10-20% of images are reserved for testing. This ensures unbiased quality assessment on viewpoints unseen during training.

Evaluation Metrics

PSNR (Peak Signal-to-Noise Ratio) measures pixel-level reconstruction accuracy using the formula:

$$\text{PSNR} = 10 \times \log_{10}\left(\frac{\text{MAX}^2}{\text{MSE}}\right) \tag{3.1}$$

where MAX is the maximum pixel value (255 for 8-bit images) and MSE is the mean squared error between rendered and ground truth images. Higher PSNR values indicate better quality, with values above 30 dB generally considered good. The metric directly quantifies how closely rendered pixels match the ground truth.

SSIM (Structural Similarity Index) [31] assesses perceptual similarity by comparing luminance, contrast, and structure:

$$\text{SSIM}(x, y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)} \tag{3.2}$$

where μ represents mean intensity, σ represents standard deviation, σ_{xy} is covariance, and c_1, c_2 are stabilization constants. SSIM ranges from 0 to 1, with values above 0.9 indicating high perceptual quality. This metric aligns better with human perception than PSNR alone.

LPIPS (Learned Perceptual Image Patch Similarity) [40] employs deep learning features from a pre-trained VGG network to measure perceptual distance. The method extracts deep features from both rendered and ground truth images, then compares feature distances in perceptual space. Lower LPIPS values indicate better quality, with values below 0.1 typically considered good. This metric captures perceptual similarity better than traditional pixel-based metrics by leveraging learned feature representations.

Evaluation Procedure

For each trained model, the evaluation pipeline loads the checkpoint and renders all test images. Each test view queries the model at its corresponding camera pose, generating an RGB image through ray marching (NeRF) or rasterization (3D Gaussian Splatting). The rendered image is then compared to the ground truth test image using all three metrics. Final scores represent averages across all test images for that model.

The implementation uses Nerfstudio's `ns-eval` command, which internally employs PyTorch for rendering, scikit-image for PSNR/SSIM calculation, and the official LPIPS implementation. Training step counts are recorded from checkpoint filenames to track convergence. All evaluations run on the same hardware (RTX 5090) to ensure fair comparison between methods.

Dataset Characterization Metrics

Beyond reconstruction quality, understanding input dataset properties helps explain reconstruction behavior and identify challenging scenarios. Five complementary metrics characterize the studio object dataset.

Luminance and Brightness

Luminance quantifies average brightness across an object's image set. Computed as the mean grayscale intensity across all pixels in representative images:

$$L = \frac{1}{N}\sum_{i=1}^{N}(0.299R_i + 0.587G_i + 0.114B_i) \tag{3.3}$$

where N is the number of pixels, and R_i, G_i, B_i are the RGB channel values for pixel i. The coefficients (0.299, 0.587, 0.114) come from the ITU-R BT.601 standard and reflect human eye sensitivity to different wavelengths. The eye contains three types of cone cells with different spectral responses. Green cones are most numerous and sensitive, contributing 58.7% to perceived brightness. Red cones contribute 29.9%, and blue cones only 11.4%. These weights ensure computed luminance matches human perception rather than treating all color channels equally. Luminance ranges from 0 (black) to 255 (white) for 8-bit images.

Dynamic Range

Dynamic range measures the spread of intensity values in an image. Rather than using absolute min-max (which outliers skew), we compute robust dynamic range using percentiles. The formula is:

$$\mathrm{DR}_{\mathrm{mean}} = \frac{1}{3} \sum_{c \in \{R,G,B\}} (P_{95}^{c} - P_{5}^{c}) \tag{3.4}$$

where P_{95}^{c} is the 95th percentile intensity for channel c and P_{5}^{c} is the 5th percentile. Percentiles work as follows: when all pixel values in a channel are sorted from lowest (0) to highest (255), the 95th percentile P_{95}^{c} is the value where 95% of pixels fall below or equal to it, representing the bright end while excluding extreme outliers. Similarly, the 5th percentile P_{5}^{c} is the value where only 5% of pixels fall below it, representing the dark end. This robust approach ignores the extreme 5% at each end, avoiding sensitivity to noise like hot pixels or dead pixels that would skew a simple max-min calculation. For each RGB channel, we find the difference between these percentiles, then average the three channel differences to produce the final dynamic range value.

High dynamic range indicates scenes with both very dark and very bright regions - challenging for reconstruction because detail must be preserved in shadows and highlights simultaneously. Low dynamic range suggests uniform lighting with compressed intensity distribution. This metric predicts reconstruction difficulty: extreme dynamic range often correlates with missing geometry in photogrammetry.

Intensity Distribution Coverage

Pixel intensity histograms are divided into three bins: low (0-31), mid (32-191), and high (192-255). The percentage of pixels falling into each bin reveals distribution characteristics. Objects dominated by low-intensity pixels (>95%) typically have dark backgrounds with small visible regions. Objects with significant mid-range coverage show textured surfaces. High-intensity coverage above 3-5% indicates prominent specular highlights or bright surfaces.

This metric connects to reconstruction challenges: high-intensity regions from specular reflections lack geometric features, causing photogrammetry failures. Neural rendering methods must model these regions through view-dependent appearance instead of geometry.

Color Dominance and Balance

Color balance examines the relationship between RGB channel means. For each channel c, compute the mean intensity:

$$\bar{c} = \frac{1}{N} \sum_{i=1}^{N} c_i \tag{3.5}$$

The dominant channel is whichever has the highest mean: dominant = $\arg\max_c \bar{c}$. Color balance differences quantify color bias:

$$\text{R-G difference} = \bar{R} - \bar{G}, \quad \text{G-B difference} = \bar{G} - \bar{B} \tag{3.6}$$

Positive R-G values indicate warm/reddish tones, negative values indicate cool/greenish tones.

Objects with extreme color bias (R-G > +20) present unusual spectral properties that reconstruction methods must handle. Balanced objects (R-G near 0) have neutral color palettes. This metric identifies whether the dataset covers diverse color characteristics or concentrates in specific spectral regions.

Information Density (Entropy)

Channel entropy measures intensity value distribution uniformity using Shannon entropy:

$$H = -\sum_{i=0}^{255} p(i) \log_2 p(i) \tag{3.7}$$

where $p(i)$ is the probability of intensity value i appearing in the channel. Entropy ranges from 0 bits (single intensity value) to 8 bits (perfectly uniform distribution across all 256 values).

High entropy (>4 bits) indicates rich texture with diverse intensity values - good for reconstruction because more visual information exists. Low entropy (<3 bits) suggests smooth surfaces or limited texture - potentially challenging because fewer features are available for matching. The metric quantifies how much visual information each object provides to reconstruction algorithms.

4 Implementation

This chapter presents the software implementation for automated multi-camera neural rendering dataset capture. The system controls twelve Teledyne industrial cameras and a motorized turntable to automate the complete capture workflow from hardware initialization through organized dataset output. The physical hardware setup—the camera rig, turntable mounting, and lighting arrangement—was designed and constructed by the thesis supervisor. This chapter focuses on the software developed by the author to control that hardware.

The primary contribution of this implementation is providing a reproducible open-source system that enables other researchers to build multi-camera capture rigs at significantly lower cost than commercial solutions ($30,000-40,000 vs $100,000+). The complete source code, totaling 7,950 lines (1,700 C# for the frontend, 6,250 C++ for the backend), has been released publicly [23] with modular architecture that allows adaptation to different camera vendors, turntable systems, or capture workflows.

Key technical achievements include parallel image saving using worker thread pools achieving 1.85× speedup over sequential processing, automated turntable control with sub-degree positioning accuracy eliminating manual rotation errors, thread-safe hardware control maintaining responsive UI during multi-minute capture sequences, and session-based data organization with consistent camera-ordered file naming for reconstruction pipelines.

This chapter explains the design decisions and technical approaches that make the system both high-performance and reproducible for the research community.

4.1 System Architecture and Design Philosophy

The software architecture follows a layered design separating user interface concerns from hardware control. This separation stems from practical needs—UI frameworks evolve rapidly, but hardware control code changes slowly. A monolithic design would force complete rewrites when updating the interface. The layered approach lets us replace the entire UI framework without touching camera or turntable code.

Figure 4.1 shows the overall architecture. The system consists of two main layers: a WinUI 3 frontend written in C# (.NET 8.0) and a C++ backend compiled as a dynamic library (CaptureCore.dll). The frontend handles user interaction through a modern Windows interface while the backend manages all hardware communication. These layers communicate

through Platform Invoke (P/Invoke), a .NET mechanism for calling native C functions from managed code.

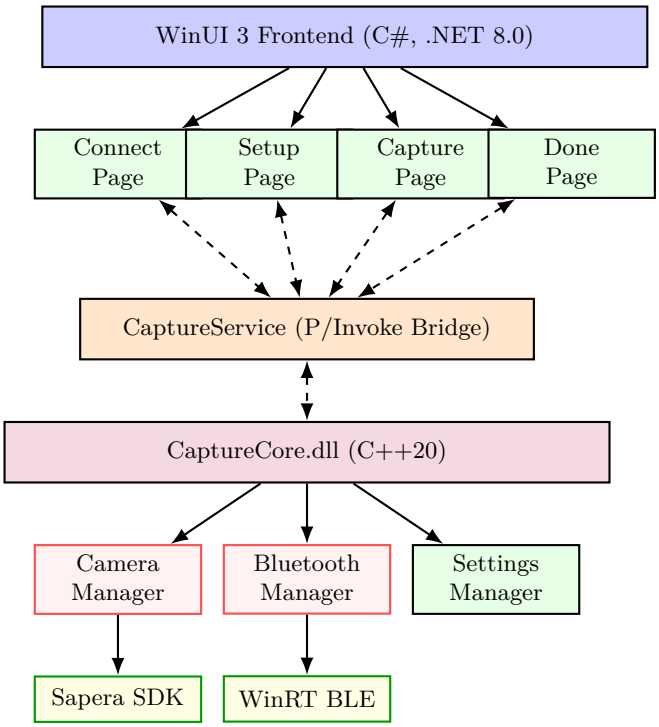


Figure 4.1: Layered software architecture. The WinUI 3 frontend communicates with the C++ backend through P/Invoke. Hardware-specific code (Sapera SDK, WinRT Bluetooth) is isolated in the backend DLL.

The backend exposes over 60 C-style functions through a clean API layer (CaptureAPI.h). Functions cover camera operations (discovery, connection, capture), Bluetooth control (scanning, rotation commands), settings management, and session handling. The frontend's CaptureService class wraps these exports as C# methods, handling marshaling between managed and unmanaged memory. Callback delegates enable the backend to report progress, log messages, and device discovery events back to the UI.

Singleton pattern for hardware managers. Both CameraManager and BluetoothManager use the singleton pattern where only one instance exists globally, accessed via `GetInstance()`. This design matches the physical constraint that only one camera network and one Bluetooth adapter exist on the machine. Multiple manager instances would conflict over the same hardware resources, causing connection failures.

4.2 Sapera SDK Integration and Camera Control

The Teledyne DALSA Sapera SDK provides direct access to industrial cameras through the GigE Vision protocol. This SDK was chosen over generic interfaces like OpenCV or GenICam for specific technical reasons. Sapera exposes camera-specific features that generic APIs abstract away - hardware triggering, packet delay tuning, and color conversion pipelines optimized for DALSA sensors. OpenCV's VideoCapture class treats industrial cameras as webcams, hiding critical parameters we need for synchronized multi-camera acquisition.

Direct SDK integration. The CameraManager class calls Sapera SDK functions directly rather than wrapping them in a generic camera interface. This design prioritizes simplicity over vendor abstraction. A generic camera interface would require translation layers for vendor-specific parameters (exposure, gain, white balance, packet timing, trigger modes) which are

implemented differently across vendors. For a single-SDK system, direct integration provides clearer code and eliminates unnecessary abstraction overhead.

The Sapera SDK handles cameras through several core classes. `SapManager` enumerates devices on the network. `SapAcqDevice` represents individual cameras and controls their parameters. `SapBuffer` allocates memory for captured images. `SapAcqDeviceToBuf` connects devices to buffers, managing the acquisition pipeline. `SapColorConversion` transforms raw sensor data (Bayer pattern) into RGB images. Our CameraManager wraps these SDK objects, managing their lifecycle through RAII - C++ constructors acquire resources, destructors release them automatically.

Parallel Capture and Save Implementation

GigE Vision cameras use standard Ethernet networks limited to 1 Gigabit per second bandwidth. Each captured image at 4024×3036 resolution produces approximately 35 MB of data. The capture phase—transferring image data from cameras to memory—is fundamentally bandwidth-limited by the GigE network. However, the subsequent save phase—writing images to disk as TIFF files—can benefit significantly from parallelization.

The implementation uses a configurable thread pool where worker threads handle image saving independently. The Sapera SDK manages camera acquisition with hardware-level synchronization, while our CameraManager coordinates parallel disk writes. This separation recognizes that capture is I/O-bound (network) while saving is I/O-bound (disk) with different bottlenecks.

Table 4.1 presents benchmark results measuring total time (capture + save) across different worker counts. Each configuration was tested with 5 capture cycles on 12 cameras at 4024×3036 resolution, processing 419.43 MB per capture.

Table 4.1: Parallel capture benchmark results (12 cameras, 4024×3036, 419.43 MB/capture)

Workers	Avg (ms)	Min (ms)	Max (ms)	MB/s	Speedup
1	19,282	18,961	19,393	21.75	1.00×
2	14,240	14,163	14,298	29.45	1.35×
3	12,295	12,141	12,526	34.12	1.57×
4	11,444	11,304	11,568	36.65	1.68×
5	11,255	11,213	11,303	37.27	1.71×
6	10,442	10,395	10,500	40.17	1.85×

Figure 4.2 visualizes the performance scaling. Total time decreases from 19.3 seconds (sequential) to 10.4 seconds (6 workers), achieving 1.85× speedup. The improvement comes primarily from parallel disk writes overlapping with network transfers. Beyond 4 workers, returns diminish as the GigE network bandwidth (95 MB/s practical) becomes the dominant bottleneck. The capture phase itself remains bandwidth-limited regardless of worker count.

Camera parameter control. Each camera exposes parameters through Sapera's feature interface. Exposure time ranges from 100 to 80,000 microseconds (0.1 to 80 milliseconds). Gain adjusts sensor sensitivity from 0 to 10 dB. White balance has separate multipliers for red, green, and blue channels. The software applies identical settings to all cameras before capture to ensure consistent brightness and color across views. If one camera had different exposure, the same surface would appear brighter in its images, confusing photogrammetry feature matching.

Parameters persist in the SettingsManager. When the application starts, it loads previous settings and applies them to discovered cameras. Users can override individual camera parameters if needed - sometimes camera 12 at the top of the arc needs slightly higher exposure due to different viewing angle. These per-camera overrides save alongside global settings.

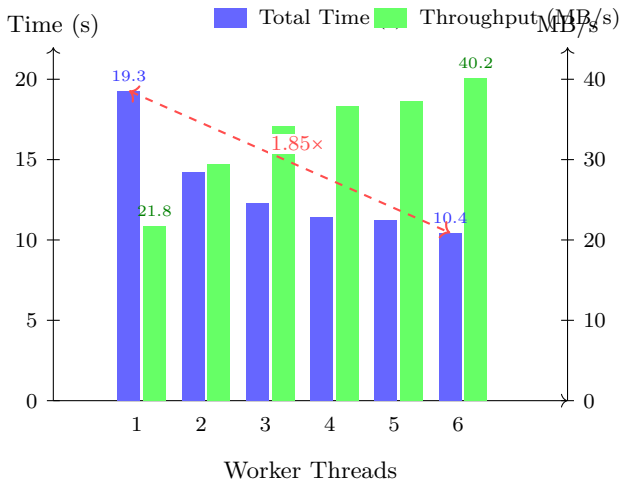


Figure 4.2: Parallel save benchmark results. Blue bars show total time (capture + save) decreasing from 19.3s to 10.4s. Green bars show effective throughput increasing from 21.8 to 40.2 MB/s. The 1.85× speedup comes from parallel disk writes overlapping with bandwidth-limited capture.

4.3 Session Management and Data Organization

Neural rendering datasets need strict version control. Researchers run experiments with different capture parameters - rotation step sizes, exposure settings, lighting configurations. If files overwrite each other, there is no way to reproduce previous results or compare parameter changes. The SessionManager enforces organization through timestamped directories.

Timestamp-based session naming. Each capture session creates a directory named `YYYYMMDD_HHMMSS_mmm` where the timestamp records down to milliseconds. Testing two objects back-to-back creates separate sessions: `20241022_143052_456` and `20241022_143108_721`. Even rapid succession captures get unique names. This precision matters during development - fixing a bug and rerunning a capture should create a new session, not overwrite the previous attempt. The SessionManager generates these timestamps automatically when `CreateNewSession()` is called, eliminating manual directory naming.

The file structure follows neural rendering conventions. RealityCapture and Nerfstudio expect images in a flat directory with consistent naming. Our structure organizes images by capture session with camera-based naming:

```
neural_dataset/images/20241022_143052_456/
    cam_01.tiff  # Camera 1 at current turntable position
    cam_02.tiff  # Camera 2 at current turntable position
    ...
    cam_12.tiff  # Camera 12 at current turntable position
```

Images are named using the camera identifier (cam_01 through cam_12) corresponding to the camera's physical position on the arc. Camera 1 is positioned horizontally, while camera 12 is mounted overhead. This naming scheme ensures consistent file ordering based on camera position rather than capture timing, which is critical for parallel capture where multiple cameras may complete simultaneously. The system supports both TIFF and RAW formats through the `capture_format_raw_` flag.

Settings persistence. The system persists configuration through an INI-based format stored in `settings.ini`. INI (Initialization) is a simple text-based configuration format using sections in square brackets and key-value pairs separated by equals signs. This format was selected for human readability and manual editability. A custom lightweight parser handles

key-value serialization without external dependencies. Camera parameters, window geometry, output paths, and per-camera settings are stored as flat key-value pairs with prefixes indicating their category (camera_, app_, individual_). Settings are stored in the executable directory and persist automatically during application shutdown.

4.4 Bluetooth Turntable Integration

Manual rotation in Versions 1 and 2 introduced positioning errors. Achieving exact 10-degree increments through manual rotation proved unreliable, with errors accumulating over multiple rotations. These inconsistencies degraded reconstruction quality by violating the regular angular spacing assumption of reconstruction algorithms. Version 3's motorized turntable eliminates manual positioning error through automated rotation.

WinRT/C++ for Bluetooth communication. Windows 10 and 11 provide Bluetooth Low Energy support through the Windows Runtime (WinRT) API. This is a C++ interface with coroutine support (`co_await`) for asynchronous operations. The alternative would be external Bluetooth libraries like SimpleBLE or Bluez. We chose WinRT for practical reasons - it ships with Windows, requires no installation, and uses modern C++ patterns we were already using elsewhere.

The WinRT API requires initialization of the Component Object Model (COM) threading model before creating Bluetooth objects. The BluetoothManager handles device discovery and connection. It uses `BluetoothScanner` to enumerate nearby devices, filtering for the turntable's unique device name. Once found, the manager creates a `BluetoothDevice` object that connects via GATT (Generic Attribute Profile) services. The turntable exposes a custom service UUID (Universally Unique Identifier, a 128-bit identifier used to distinguish Bluetooth services) with characteristics for sending commands and receiving position feedback.

Command protocol design. The turntable accepts string-based commands over Bluetooth. Commands follow a simple format:

```
+CT,TURNANGLE=45;      // Rotate 45 degrees clockwise
+CT,TURNSPEED=70;      // Set rotation speed
+CR,TILTVALUE=30;      // Tilt 30 degrees up
+QT,CHANGEANGLE;       // Query current angle
```

This protocol was defined by the turntable vendor. String-based commands were selected over binary protocols for debugging convenience during development. The BluetoothCommands namespace provides helper functions that generate these command strings. Instead of remembering syntax, code calls `BluetoothCommands::RotateByAngle(45)` which returns the formatted string. This prevents typos and centralizes command definitions.

Commands are sent asynchronously. The BluetoothDevice class calls `WriteCommand()`, which uses WinRT's `co_await` to send data without blocking. The UI thread continues running while the Bluetooth operation completes in the background. When the turntable sends a response, a callback updates the UI with the new position. This async approach prevents the interface from freezing during multi-second rotation operations.

4.5 User Interface Design

GUI framework selection. The frontend uses WinUI 3, Microsoft's modern UI framework for Windows applications. WinUI 3 provides the Windows 11 Fluent Design aesthetic with native performance. Unlike web-based alternatives (Electron, React Native), WinUI 3 runs natively without browser overhead. Compared to older frameworks like Windows Forms or WPF, WinUI 3 offers better high-DPI support, modern styling, and active development.

The choice to separate the UI (C#) from hardware control (C++) reflects practical constraints. Industrial camera SDKs like Sapera provide C/C++ libraries only. Rather than

fighting language boundaries within a single executable, the layered architecture cleanly separates concerns. The C# frontend handles presentation logic while the C++ backend handles hardware. P/Invoke bridges the gap with minimal overhead.

Wizard-Style Navigation

The interface follows a four-step wizard pattern guiding users through the capture workflow: Connect, Setup, Capture, and Done. This linear flow reduces cognitive load compared to panel-based interfaces where users must understand which controls affect which operations. Each page focuses on one task.

Connect Page. Users discover and connect cameras and turntable. The page shows discovered devices with connection status indicators. Camera discovery scans the GigE network for Teledyne devices. Bluetooth scanning finds the Revopoint turntable. Users cannot proceed until required hardware is connected.

Setup Page. Users configure capture parameters before starting. Preset buttons offer common configurations: 36 positions (10° increments) or 72 positions (5° increments). Manual mode allows custom position counts and rotation angles. The output folder selector determines where captured images are saved. A settings dialog provides access to advanced parameters: exposure time, gain, and worker thread count for parallel capture.

Capture Page. The main data collection interface shows real-time progress during automated capture sequences. A progress bar indicates completion percentage. Timing information displays elapsed time and estimated remaining duration. The current turntable position and capture state (idle, capturing, rotating, settling) appear prominently. Users can stop capture mid-sequence if problems occur.

Done Page. After capture completes, this page summarizes results: total images captured, session folder path, and elapsed time. A button opens the output folder in Windows Explorer for immediate verification. Users can start a new capture or exit the application.

Figures 4.3 through 4.6 show the WinUI 3 interface across the capture workflow.

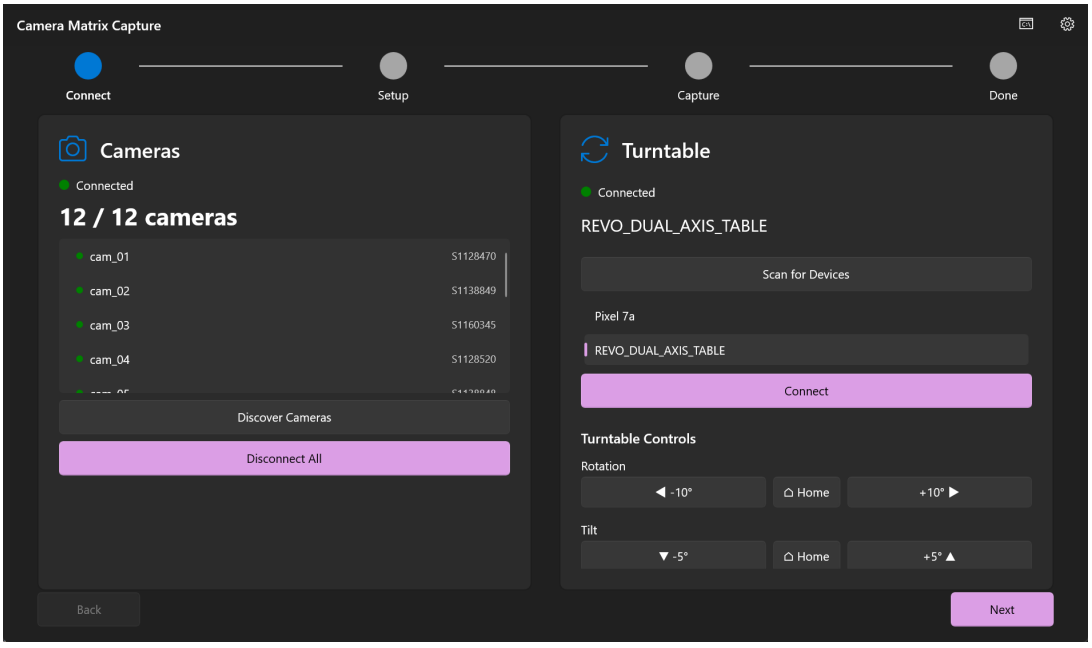


Figure 4.3: Connect page showing camera discovery (left) and Bluetooth turntable connection (right). The step indicator at top shows progress through the four-stage wizard. Green indicators confirm successful hardware connections.

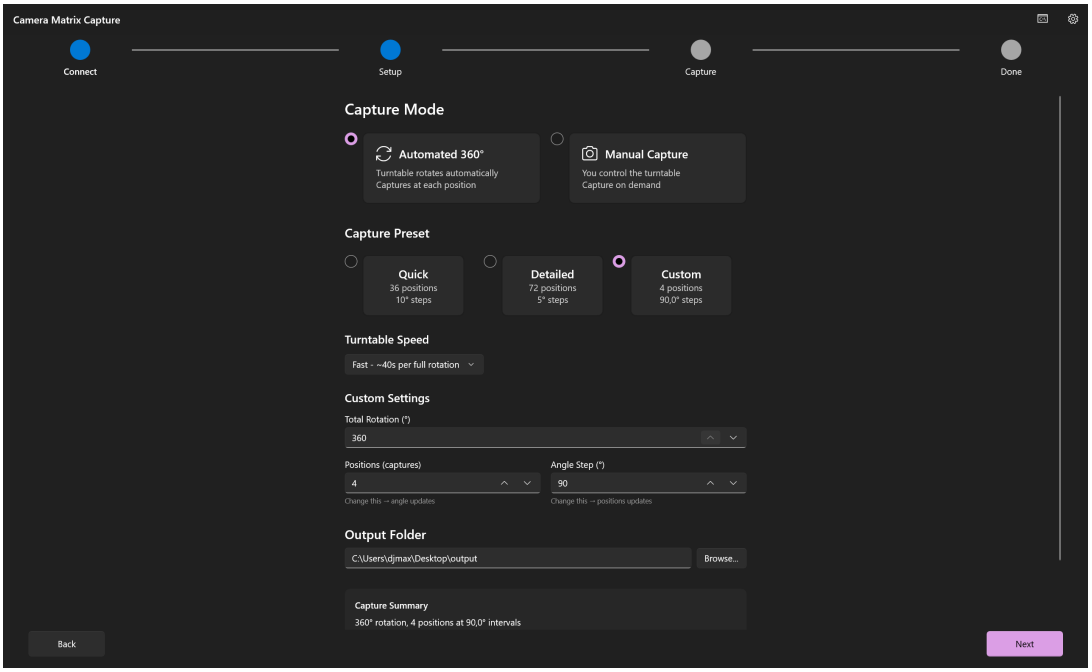


Figure 4.4: Setup page with capture mode selection (Automated 360° or Manual), preset configurations (Quick: 36 positions at 10° steps, Detailed: 72 positions at 5° steps), output folder selection, and capture summary showing expected image count and estimated duration.

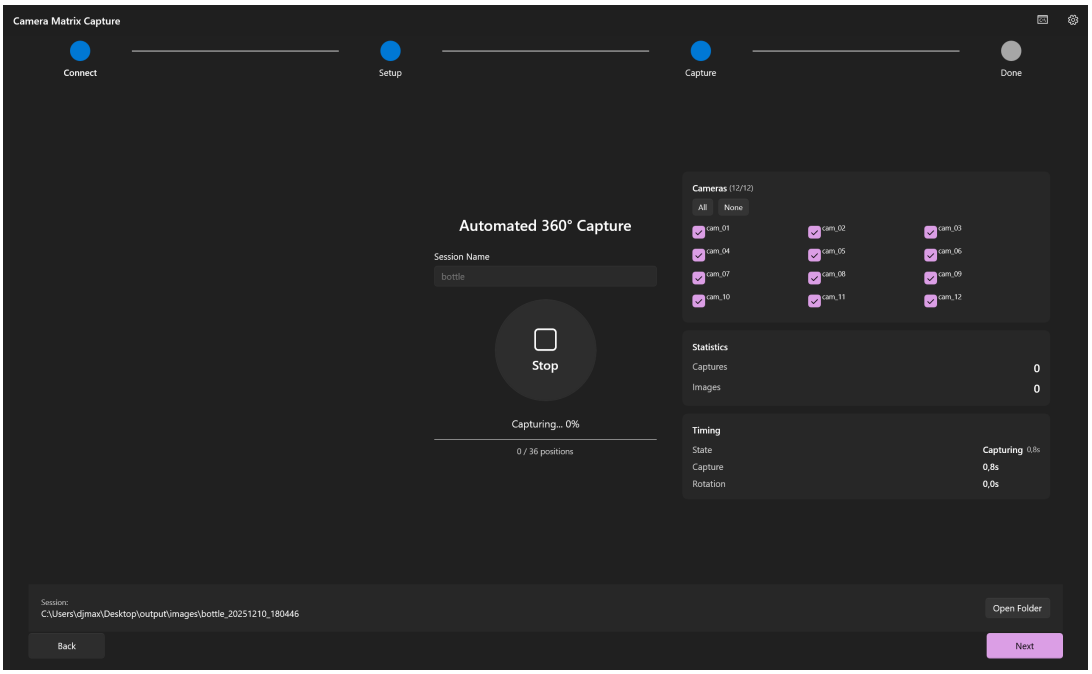


Figure 4.5: Capture page ready to begin automated 360° acquisition. The central Start button initiates the capture sequence. Statistics panel tracks captures and total images. Progress indicator shows current position out of total positions.

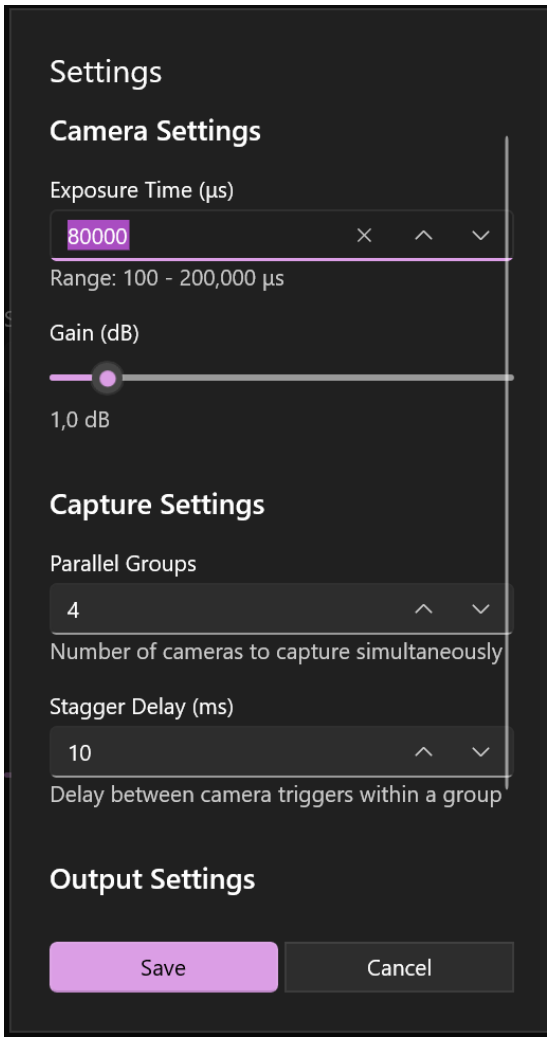


Figure 4.6: Settings dialog for camera and capture parameters. Exposure time (microseconds) and gain (dB) control image brightness. Worker thread count configures parallel capture performance.

4.6 Threading Model and Asynchronous Operations

Hardware operations like camera discovery, connection establishment, and full capture sequences are time-consuming operations. Running these on the main UI thread would freeze the interface completely—no button would respond, no log messages would appear, the application would seem crashed.

The solution combines background threading in the C++ backend with callback marshaling to the C# frontend. Long operations run on separate threads in the DLL while the WinUI thread continues processing UI events.

Backend threading. The C++ backend spawns worker threads for slow operations. Camera discovery scans the network for GigE devices. Capture sequences use a configurable thread pool for true parallel multi-camera acquisition. Bluetooth operations use WinRT's async/await pattern. Each operation reports progress through registered callbacks.

P/Invoke callback marshaling. The frontend registers callback delegates with the backend during initialization. Four callback types handle different events: LogCallback for debug messages, ProgressCallback for capture progress (current position, total positions), DeviceDis-

coveredCallback for newly found cameras or turntables, and CaptureCompleteCallback when sequences finish. The C++ backend stores function pointers to these delegates and invokes them from worker threads.

WinUI's DispatcherQueue handles thread safety on the C# side. When a callback fires from a background thread, the frontend dispatches UI updates to the main thread:

```
private void OnProgress(int current, int total) {
    DispatcherQueue.TryEnqueue(() => {
        ProgressBar.Value = (double)current / total * 100;
    });
}
```

This pattern keeps hardware operations responsive while maintaining UI thread safety. The backend focuses on hardware timing; the frontend focuses on presentation.

Debug console for backend visibility. A dedicated debug console window displays real-time log messages from the C++ backend, providing transparency into hardware operations. Figure 4.7 shows the console during a capture sequence. Each log entry includes timestamps, operation context (REC for recording, OK for completion), and detailed status messages. Users can monitor camera capture progress, image saving operations, and timing between capture groups. This visibility proves essential for diagnosing issues—if a camera fails to respond or an image save takes unexpectedly long, the console immediately reveals the problem. The auto-scroll feature keeps the latest messages visible during long capture sequences.

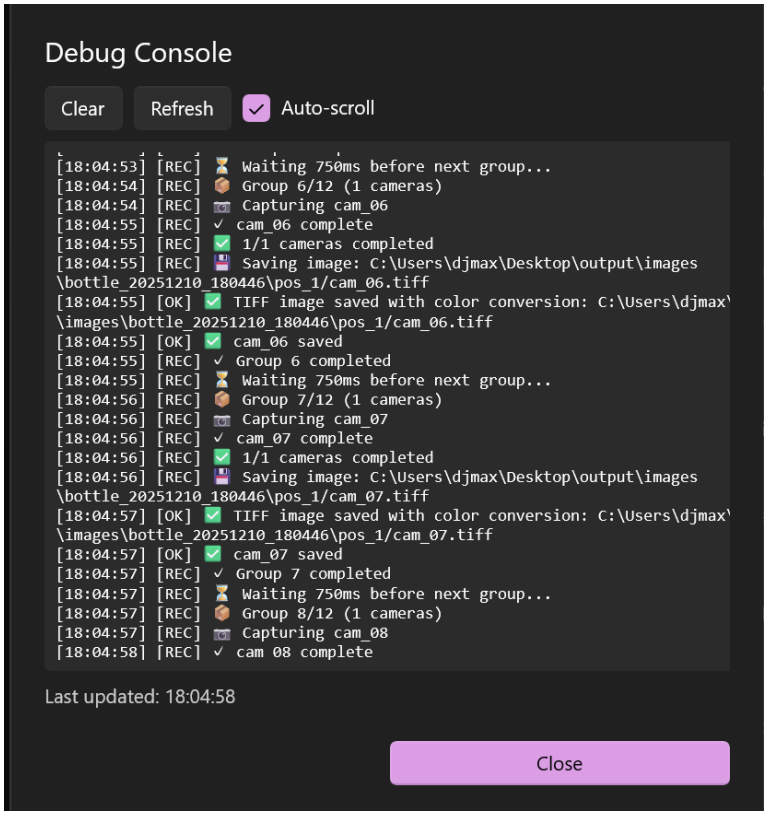


Figure 4.7: Debug console showing real-time backend status during capture. Log entries display timestamps, operation states (REC/OK), camera identifiers, and file paths. The console provides visibility into the C++ backend operations, helping users understand system behavior and diagnose issues.

Polling for state queries. Some state information uses polling rather than callbacks. The frontend's DispatcherTimer queries camera count, connection status, and capture state every 200–500 milliseconds. This approach works well for status displays that update continuously rather than in response to discrete events.

4.7 Build System

The project uses a dual build process reflecting the two-language architecture. The C++ backend builds with CMake, producing CaptureCore.dll. The C# frontend builds with the .NET SDK, producing the main executable.

Backend build (CMake). The C++ backend uses CMake 3.16+ for configuration. The Sapera SDK must be installed manually from Teledyne DALSA before building. CMake finds the SDK through standard Windows paths. The build produces a shared library (DLL) that exports C-style functions for P/Invoke consumption.

Frontend build (.NET). The WinUI 3 frontend uses standard .NET 8.0 tooling. The project file references the Windows App SDK for WinUI 3 support. Building requires Visual Studio 2022 or the .NET 8 SDK with Windows workloads. The output bundles CaptureCore.dll alongside the main executable.

Combined build. A build script coordinates both processes: first building the C++ DLL, then copying it to the .NET project's output directory, finally building and packaging the complete application. The open-source repository includes these scripts for straightforward reproduction.

4.8 System Requirements

The software requires Windows 10/11 with .NET 8.0 runtime and the Sapera SDK installed for Teledyne camera support. Bluetooth LE adapter is required for Revopoint dual-axis turntable communication. Recommended system specifications include 16GB RAM and SSD storage for high-speed parallel image writing. The configurable worker thread count allows tuning parallel save performance—benchmarks show 1.85× speedup with 6 workers, reducing total capture-to-disk time from 19.3 to 10.4 seconds.

4.9 Making This System Reproducible

A key contribution of this implementation is enabling other researchers to replicate the multi-camera capture setup. This section outlines the requirements and process for reproduction.

Hardware requirements. The system requires twelve Teledyne DALSA Nano-C4020 cameras (or compatible GigE Vision industrial cameras), a camera mounting arc with 90-degree coverage and adjustable positions, a 1 Gbps network switch with sufficient ports for all cameras, a Revopoint dual-axis motorized turntable with Bluetooth LE support, a Windows 10/11 PC with Bluetooth adapter, 16GB RAM minimum, and SSD storage recommended. Detailed specifications are provided in Chapter 3.

Software dependencies. The system requires Teledyne DALSA Sapera SDK (free download from vendor), CMake 3.16 or later for C++ backend build, Microsoft Visual Studio 2022 with C++20 support, .NET 8.0 SDK with Windows App SDK for WinUI 3 frontend, and source code available at the open-source repository.

Setup process. Replicating the system involves mounting cameras on arc structure at calculated positions (see methodology chapter for geometry), connecting cameras to network switch via GigE, configuring network adapter for jumbo frames (9000 byte MTU recommended), installing Sapera SDK and configuring camera network discovery, building the C++ backend with CMake and the C# frontend with .NET SDK, pairing Bluetooth turntable via

Windows settings, and calibrating camera positions and turntable center alignment. Estimated setup time for experienced users is one to two days. The primary skill requirements are C++/C# compilation experience and basic networking knowledge.

Cost and accessibility. Total hardware cost is approximately $30,000-40,000 USD (primarily camera cost). This is significantly lower than commercial photogrammetry systems which typically cost $100,000+. The open-source software eliminates licensing fees and enables customization for specific research requirements.

Comparison to alternatives. Unlike commercial systems (e.g., RealityCapture's camera rigs), this implementation provides full source code access, enabling modifications for different camera models, custom capture sequences, or integration with other neural rendering pipelines. The modular architecture allows researchers to replace components (e.g., swap Bluetooth turntable for motorized rail) without complete rewrites.

4.10 RealityCapture Reconstruction Workflow

After image capture, reconstruction requires estimating camera poses and generating 3D models. RealityCapture serves as our primary tool rather than COLMAP, which is the default in most neural rendering pipelines.

Why RealityCapture Over COLMAP

COLMAP assumes consistent lighting across input images. This assumption fails in both our capture scenarios. In the studio, four softbox area lights remain stationary while the turntable rotates (see Section 3.1). As the object turns, reflections shift and shadows fall differently relative to the cameras—the same surface patch appears bright from one turntable angle and shadowed from another. This rotation-dependent lighting variation confuses COLMAP's feature matching. Outdoors at Gränsö Castle, images captured over several hours show significant illumination changes as the sun moved.

COLMAP failed to produce usable alignments on our studio datasets and fragmented outdoor sequences into disconnected components. RealityCapture's matching algorithm handles lighting inconsistency more robustly, successfully aligning images that COLMAP could not process.

ArUco Markers for Robust Alignment

ArUco markers printed on the fiducial disc provide reliable features when objects lack texture or have reflective surfaces. RealityCapture matches these markers across views first, establishing camera geometry, then incorporates whatever object features exist. When markers are partially occluded by large objects, the control point tool allows manual correspondence marking across images, typically requiring three to five points to merge fragmented components.

Alignment and Export

RealityCapture's structure-from-motion creates separate components when images fail to align together. The highest aligned component—containing the most registered cameras—goes forward for subsequent processing.

Photogrammetric mesh generation uses the high quality reconstruction mode, with vertex count and texture resolution adjusted based on object complexity.

Camera transforms export from the aligned component through a Python script that converts RealityCapture's coordinate convention to Nerfstudio's `transforms.json` format. Scale calibration relies on the known fiducial disc diameter (200mm) as reference.

4.11 Nerfstudio Training Pipeline

Nerfstudio provides a unified framework for training neural radiance field variants and serves as the platform for all neural rendering experiments in this thesis.

Environment and Data Preparation

The training environment uses conda with Python 3.11, PyTorch 2.7, CUDA 12.8, and Nerfstudio 1.1. All experiments run on an NVIDIA RTX 5090 (32GB VRAM).

Nerfstudio expects images in an `images/` subdirectory with camera parameters in `transforms.json`. For studio objects, images are 4112×3008; for Gränsö Castle, SLR images are 3840×2560 and drone images are 5280×3956. We downsample by factor 0.5 for training. The camera optimizer is disabled since RealityCapture poses are already refined.

Model Selection

We evaluate two methods available in Nerfstudio:

- Nerfacto: The default NeRF implementation combining proposal sampling, hash encoding, and appearance embedding. We train for 50,000 iterations.
- Splatfacto: 3D Gaussian Splatting implementation with real-time rendering capability.

Evaluation

We report three standard metrics: PSNR for pixel-level accuracy, SSIM for structural similarity, and LPIPS for perceptual quality. Nerfstudio holds out every 8th image for evaluation by default, measuring novel view synthesis on unseen viewpoints. The built-in web viewer enables interactive quality inspection, and export functions generate point clouds for comparison with photogrammetric reconstructions.

4.12 System Limits and Constraints

While the implementation successfully achieves automated multi-camera capture, several constraints should be acknowledged.

Platform dependency. The system requires Windows 10/11 due to the WinRT Bluetooth API used for turntable control and the Windows-only Sapera SDK. The implementation is designed specifically for Windows environments.

Vendor lock-in. Direct integration with the Sapera SDK limits the system to Teledyne DALSA cameras. Supporting other industrial camera vendors (e.g., Basler, FLIR) would require either implementing vendor-specific camera managers or developing a generic camera abstraction layer. The current architecture facilitates such extension but does not provide it out-of-box.

Network topology constraints. The system assumes all cameras connect to a single 1 Gbps network switch. Scaling beyond 12-15 cameras would require either multiple network adapters with separate camera groups, 10 Gbps network infrastructure, or sequential capture with longer total acquisition times. Current bandwidth validation prevents unsafe configurations but does not optimize camera-to-group assignment for multi-adapter scenarios.

Real-time preview limitations. The system does not provide live video streaming from all cameras simultaneously. Continuous multi-camera video preview would consume the entire 1 Gbps network bandwidth, preventing actual image capture. The system provides single-camera snapshot preview instead, which is sufficient for static object capture workflows.

5 Results

This chapter presents reconstruction results from the multi-view capture systems described in Chapter 3. Four primary result categories are evaluated: studio object reconstructions comparing photogrammetry and neural rendering approaches, smartphone-based polarization validation demonstrating specular reflection mitigation, large-scale outdoor reconstruction of Gränsö Castle, and automated image capture system software. Each section includes visual comparisons, quantitative metrics, and analysis of method-specific strengths and limitations.

5.1 Studio Object Reconstructions: Photogrammetry vs. Neural Rendering

Studio objects were reconstructed using both RealityCapture photogrammetry and neural rendering methods (Nerfacto and Splatfacto). This section presents visual comparisons and quantitative evaluation across different material types, highlighting the complementary strengths of geometric and volumetric reconstruction approaches.

Dataset Overview

The studio dataset comprises objects selected to represent diverse reconstruction challenges: diffuse materials (matte ceramics, toys, statues), semi-glossy surfaces (painted objects), highly reflective materials (glazed pottery, metal), and transparent/refractive elements (glass vase). Each object was captured using the V3 motorized turntable system with 12 cameras at evenly distributed vertical angles (0° to 90°) and 36 turntable positions (10° increments), yielding 432 images per object. The complete dataset of 15 objects is publicly available on the Internet Archive [39].

Figure 5.1 shows all 15 captured objects. Table 5.1 summarizes the captured objects and reconstruction outcomes across different methods.

Dataset Intensity Distribution Analysis

Figure 5.3 shows intensity histograms for representative images from each object. The analysis reveals distinct intensity patterns across the 15 objects, driven primarily by material properties and lighting interaction.

Figure 5.1: Overview of all 15 studio objects captured in the dataset. Each object was photographed from 12 camera angles at 36 turntable rotation positions, yielding 432 images per object.

Figure 5.2: The publicly released studio objects dataset on the Internet Archive. The dataset includes all 432 images per object (12 cameras × 36 turntable positions), camera calibration data, and RealityCapture project files. Available at `https://archive.org/details/captured_objects_dataset`.

Table 5.1: Studio object dataset composition by material type

Object	**Material Type**
Banana	Diffuse
Bear	Diffuse
Bilboquet	Diffuse
Cauldron	Specular/Metal
Dog	Diffuse
Flowerpot	Diffuse
Frog	Diffuse
Pedestal	Diffuse
Plantplot	Diffuse
Pot	Specular/Glossy
Pumpkin	Diffuse
Skull	Diffuse
Spray	Diffuse/Semi-glossy
Stone	Diffuse
Vase Glass	Refraction/Transparent
Pumpkin	Diffuse
Skull	Diffuse

Intensity distribution. Most objects concentrate pixels in the low intensity range due to dark backgrounds. Dog shows the highest concentration with 98.56% of pixels below intensity 31, followed by Vase Glass at 98.38% and Cauldron at 96.68%. Pumpkin, Skull, Flowerpot, and Spray differ significantly, with only 78.27%, 81.30%, 80.78%, and 88.61% in the low range respectively. These objects show broader histogram distributions spanning mid and high intensity bins due to bright textured regions and prominent specular highlights.

Brightness levels. Luminance values vary substantially across objects. Skull ranks brightest at 39.24, followed by Pumpkin at 37.24 and Flowerpot at 33.52. The darkest objects are Vase Glass at 10.64 and Dog at 12.45. Skull and Pumpkin contain bright colored material under direct illumination. Vase Glass concentrates energy near zero with only sparse high-end spikes from specular reflections.

Dynamic range. We computed dynamic range as the difference between 95th and 5th percentile pixel values for each RGB channel, then averaged the three channels. This robust metric excludes extreme outliers while capturing actual intensity spread. Skull achieves the highest dynamic range at 208.7, followed by Spray at 186.3 and Flowerpot at 171.7. These values indicate pixels spanning from near-black to near-white. Pumpkin shows 139.3, while mid-range contrast appears in Pedestal at 85.3 and Pot at 68.0. Low-contrast scenes include Frog at 30.7 and Stone at 34.7. High dynamic range correlates with prominent highlights: Skull shows 7.64% pixel coverage in the high-intensity bin, Pumpkin shows 5.13%, and Spray shows 4.83%, while most other objects stay below 1%.

Color balance. Red-dominant scenes include Banana, Bear, Flowerpot, Pedestal, Plantpot, Pot, Pumpkin, Skull, Spray, and Stone. Blue-dominant scenes include Bilboquet, Cauldron, Dog, Frog, and Vase Glass. Pumpkin shows the strongest red bias with a red-green difference of +23.86, far exceeding other objects due to its bright orange color. Bear follows at +4.95 and Banana at +3.71. Flowerpot shows the largest green-blue difference at +5.93, indicating a warm greenish palette, while Pumpkin reaches +10.77 in green-blue difference.

Information density. Channel entropy measures how uniformly intensity values distribute across the histogram. Pumpkin peaks at 4.923 bits, followed by Skull at 4.316 bits and Flowerpot at 4.262 bits. These high values indicate richer, more uniformly distributed intensities. Spray reaches 3.793, Vase Glass reaches 3.390, and Pot reaches 3.040. Low-variance scenes show lower entropy: Dog at 2.247 and Bilboquet at 2.194. Entropy broadly tracks dynamic range but captures how intensity mass spreads across histogram bins. Broad, textured distributions raise entropy even at moderate brightness levels.

Table 5.2: Color statistics and luminance for studio objects

Object	Lum.	Mean R	Mean G	Mean B	Dom.	R-G
Banana	15.22	17.46	13.75	14.43	Red	+3.71
Bear	15.94	19.03	14.08	14.70	Red	+4.95
Bilboquet	14.48	15.33	12.78	15.34	Blue	+2.55
Cauldron	13.32	13.98	11.53	14.45	Blue	+2.45
Dog	12.45	12.94	10.58	13.83	Blue	+2.37
Flowerpot	33.52	37.11	34.69	28.76	Red	+2.41
Frog	14.01	14.75	12.28	14.99	Blue	+2.47
Pedestal	19.33	20.81	18.07	19.11	Red	+2.74
Plantpot	17.28	19.22	15.69	16.94	Red	+3.53
Pot	21.10	22.28	19.84	21.20	Red	+2.44
Spray	26.89	28.53	27.58	24.57	Red	+0.95
Stone	14.55	15.59	12.91	15.17	Red	+2.68
Vase Glass	10.64	10.67	10.12	11.13	Blue	+0.54
Pumpkin	37.24	56.74	32.88	22.11	Red	+23.86
Skull	39.24	44.51	41.33	31.89	Red	+3.17

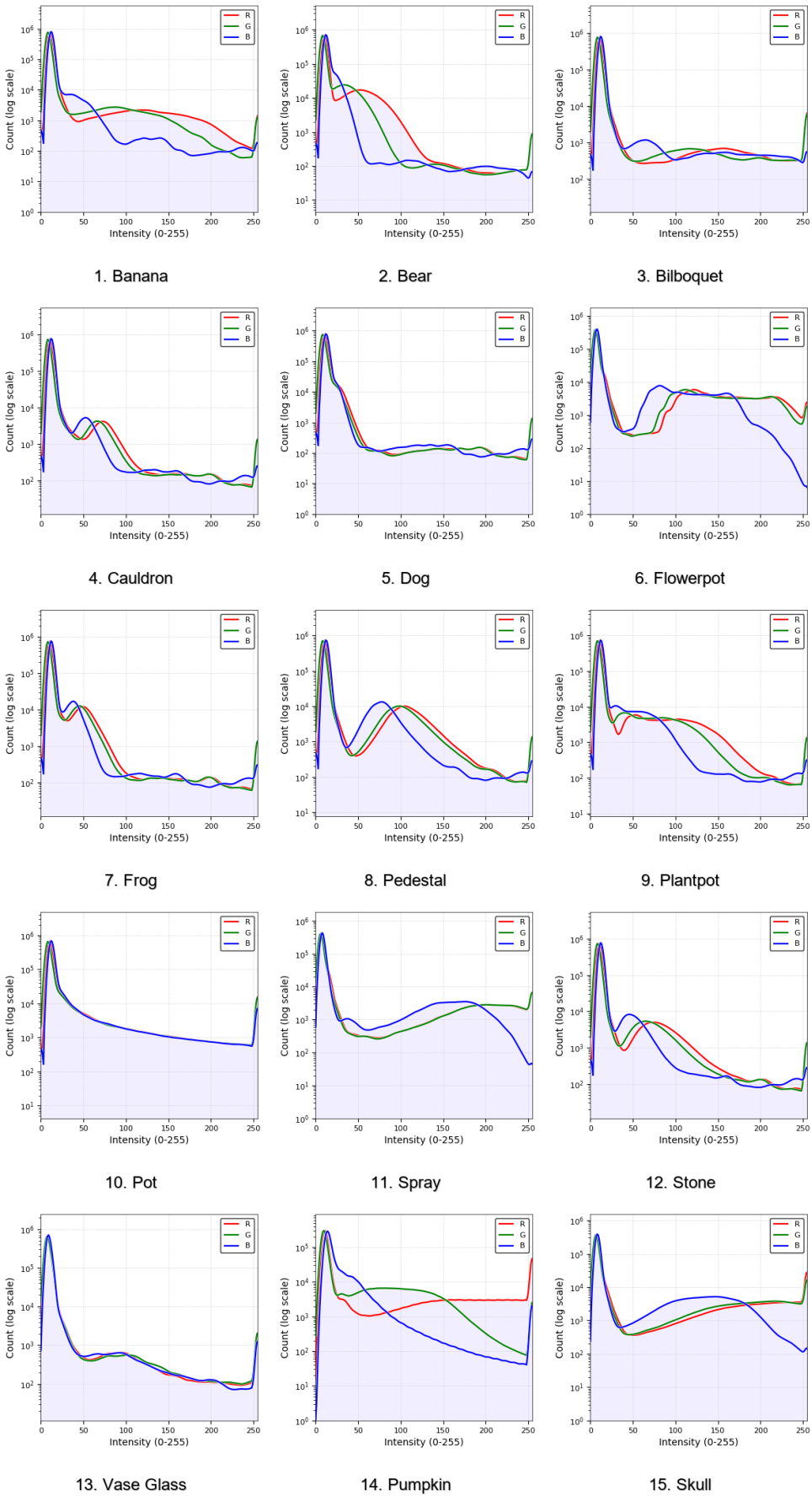


Figure 5.3: Intensity histograms for all 15 objects (log scale). Each histogram shows pixel distribution from representative captures. Most objects concentrate pixels in low-intensity ranges due to dark backgrounds. Flowerpot and Spray show broader distributions with significant mid and high-range coverage from textured surfaces and highlights.

Table 5.3: Dynamic range and intensity distribution coverage

Object	DR Mean	Low %	Mid %	High %
Banana	27.3	95.20	4.48	0.32
Bear	38.0	90.30	9.56	0.14
Bilboquet	8.7	97.56	1.60	0.84
Cauldron	9.3	96.68	3.11	0.21
Dog	11.3	98.56	1.23	0.20
Flowerpot	171.7	80.78	15.62	3.59
Frog	30.7	93.62	6.18	0.21
Pedestal	85.3	90.58	9.20	0.22
Plantpot	62.0	91.01	8.78	0.21
Pot	68.0	90.65	7.34	2.02
Spray	186.3	88.61	6.57	4.83
Stone	34.7	94.68	5.11	0.21
Vase Glass	9.0	98.38	1.31	0.31
Pumpkin	139.3	78.27	16.60	5.13
Skull	208.7	81.30	11.07	7.64

Low: 0-31, Mid: 32-191, High: 192-255; DR Mean: average RGB dynamic range (95th-5th percentile)

Table 5.4: Information density (entropy) by RGB channel

Object	Entropy R	Entropy G	Entropy B	Mean
Banana	2.367	2.444	2.215	2.342
Bear	2.951	3.001	2.597	2.850
Bilboquet	2.161	2.284	2.137	2.194
Cauldron	2.251	2.370	2.190	2.271
Dog	2.219	2.348	2.174	2.247
Flowerpot	4.288	4.357	4.141	4.262
Frog	2.428	2.562	2.384	2.458
Pedestal	2.652	2.781	2.606	2.679
Plantpot	2.690	2.809	2.609	2.702
Pot	3.007	3.143	2.971	3.040
Spray	3.816	3.870	3.695	3.793
Stone	2.372	2.496	2.313	2.394
Vase Glass	3.385	3.516	3.269	3.390
Pumpkin	4.873	5.149	4.748	4.923
Skull	4.260	4.409	4.279	4.316

COLMAP Alignment Failure

COLMAP was tested as an alternative structure-from-motion pipeline but failed to reconstruct any of the 15 objects. Feature matching succeeded for only 2 images out of 432 per object (0.5% success rate). The failure was caused by inconsistent lighting across the turntable rotation sequence. As the object rotates, different surfaces face the area lights at different angles, creating significant brightness variation between consecutive captures. COLMAP's feature descriptor matching is sensitive to these photometric changes, particularly when rotation creates large intensity differences in corresponding regions.

RealityCapture proved more robust to lighting inconsistencies, successfully aligning all 432 images per object. This robustness likely stems from RealityCapture's proprietary matching algorithms that account for illumination variation, whereas COLMAP's standard SIFT/AKAZE descriptors expect approximately constant brightness between corresponding features. The lighting sensitivity demonstrates a critical limitation when using COLMAP for turntable capture without strictly uniform illumination.

Reconstruction Results Comparison

Figures 5.4, 5.5, and 5.6 present side-by-side comparisons of all reconstruction methods across the 15 studio objects. Each row shows one object with four columns: input image, RealityCapture photogrammetry, Nerfacto (NeRF), and Splatfacto (3DGS). This layout enables direct visual comparison of method performance on identical data.

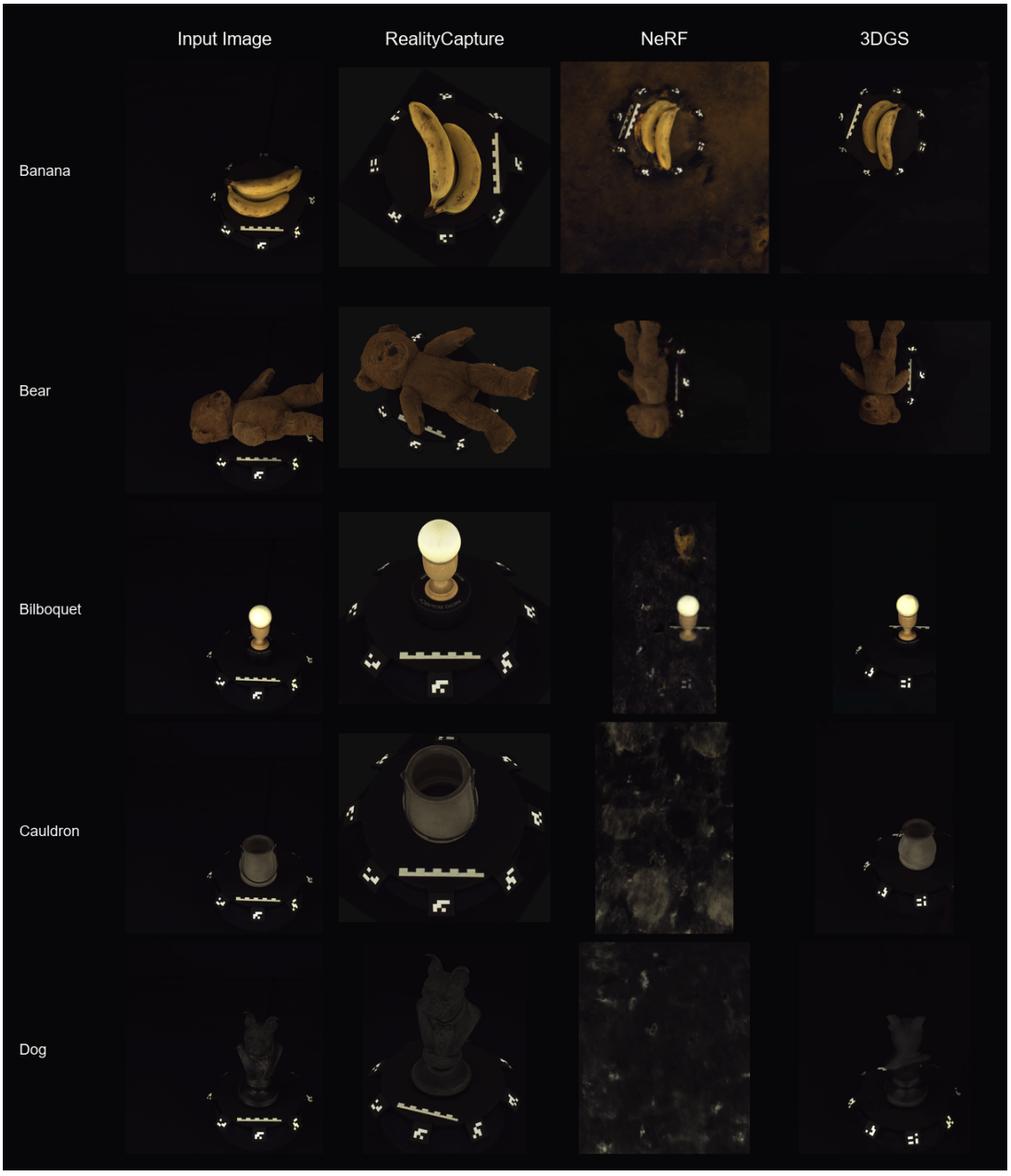


Figure 5.4: Reconstruction comparison for objects 1–5 (Banana, Bear, Bilboquet, Cauldron, Dog). Columns show input image, RealityCapture mesh, Nerfacto rendering, and Splatfacto rendering.

RealityCapture successfully reconstructed 13 of 15 objects. Two objects failed: Spray (featureless white plastic) and Vase Glass (transparent material). Both Nerfacto and Splatfacto successfully rendered all 15 objects, demonstrating their ability to handle challenging materials where traditional photogrammetry fails.

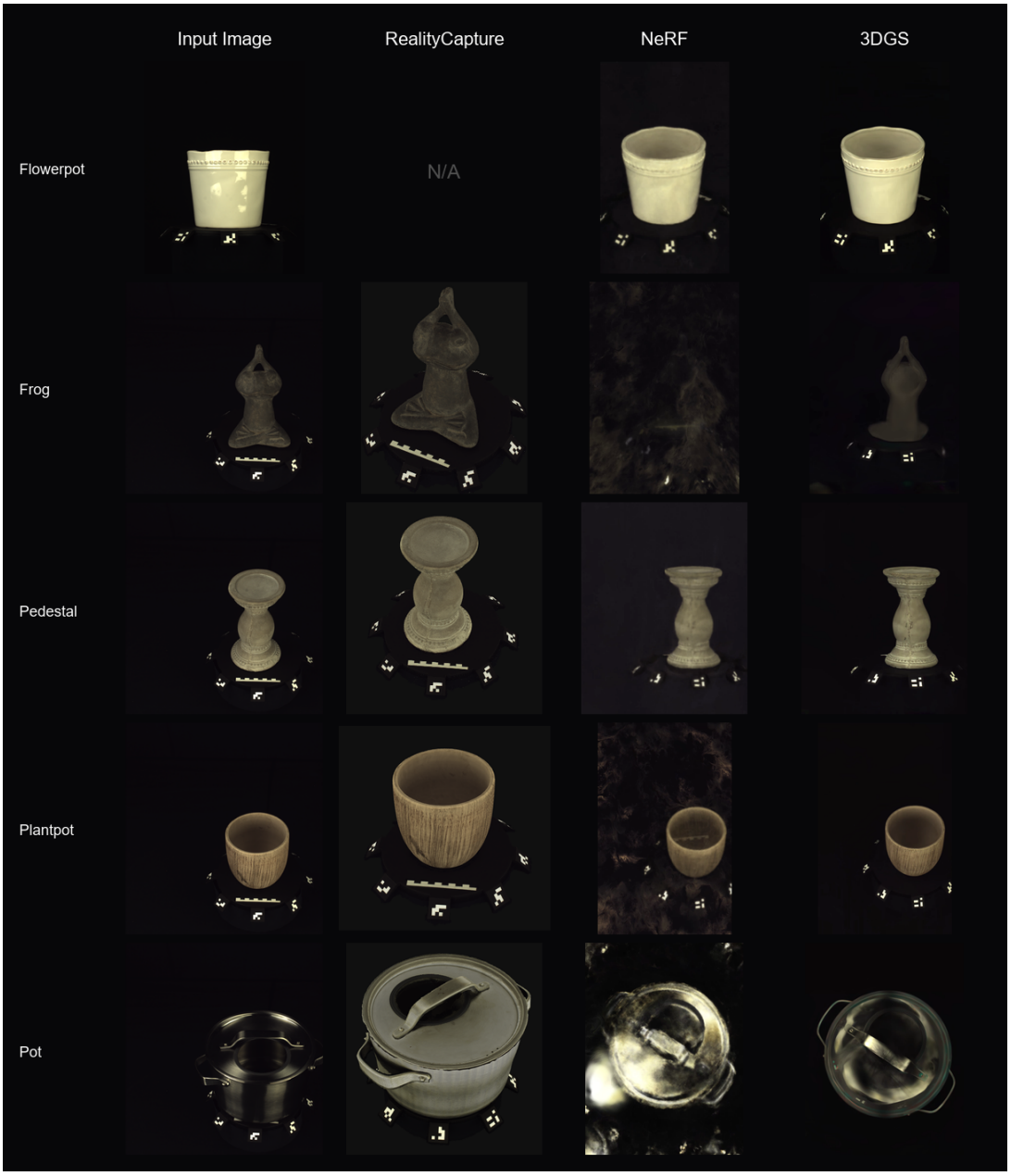


Figure 5.5: Reconstruction comparison for objects 6–10 (Flowerpot, Frog, Pedestal, Plantpot, Pot). Flowerpot shows N/A for RealityCapture due to low alignment rate.

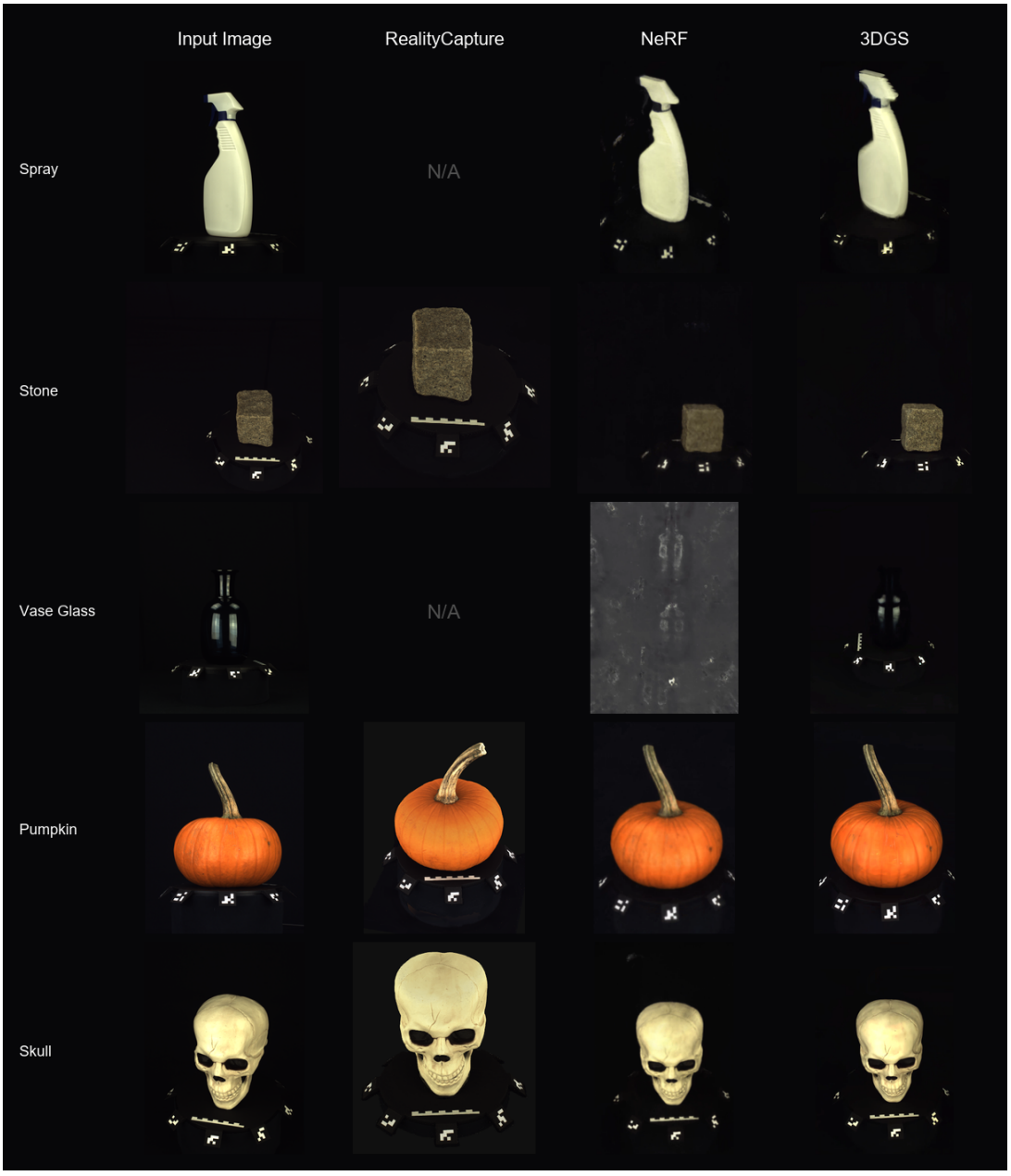


Figure 5.6: Reconstruction comparison for objects 11–15 (Spray, Stone, Vase Glass, Pumpkin, Skull). Spray and Vase Glass show N/A for RealityCapture due to featureless and transparent surfaces respectively.

Studio Object Alignment Quality

Table 5.5 presents the RealityCapture alignment quality metrics for all studio objects. Reprojection error measures camera pose accuracy: lower values indicate more precise pose estimation.

Table 5.5: Studio object alignment quality metrics from RealityCapture

Object	Aligned	Tie Points	Mean Err.	Median Err.	Track Len.
Banana	369/432 (85%)	17,524	0.68 px	0.58 px	4.54
Bear	432/432 (100%)	242,345	0.63 px	0.51 px	4.76
Bilboquet	385/432 (89%)	15,162	0.62 px	0.50 px	4.16
Cauldron	396/432 (92%)	25,971	0.65 px	0.55 px	4.52
Dog	422/432 (98%)	59,799	0.55 px	0.43 px	4.57
Flowerpot	118/720 (16%)	4,167	0.52 px	0.38 px	3.82
Frog	432/432 (100%)	58,483	0.59 px	0.48 px	6.64
Pedestal	432/432 (100%)	81,859	0.64 px	0.55 px	6.68
Plantpot	358/432 (83%)	46,298	0.55 px	0.42 px	5.74
Pot	637/720 (88%)	55,160	0.62 px	0.48 px	3.18
Pumpkin	720/720 (100%)	168,175	0.76 px	0.67 px	4.07
Skull	720/720 (100%)	83,278	0.71 px	0.61 px	4.29
Spray	Failed - featureless surface				
Stone	432/432 (100%)	69,826	0.53 px	0.39 px	8.32
Vase Glass	Failed - transparent/refractive surface				

Of the 15 objects, 13 achieved successful alignment with sub-pixel mean reprojection error (0.52–0.76 pixels), indicating high-accuracy camera pose estimation suitable for neural rendering. Two objects failed: Spray (featureless white plastic surface lacks texture for feature matching) and Vase Glass (transparent/refractive material causes inconsistent feature appearance). Six objects achieved 100% alignment rate, while others ranged from 83–98%. Note that some objects were captured with 720 images (20 cameras × 36 positions) rather than 432 images (12 cameras × 36 positions) due to system upgrades during data collection.

Material-Specific Reconstruction Quality

Different reconstruction methods exhibit distinct performance characteristics based on material properties.

Diffuse Materials

Matte, diffuse objects achieve near-complete reconstruction across all methods. RealityCapture produces geometrically accurate meshes suitable for measurement. Nerfacto and Splatfacto match or exceed visual quality, though geometric measurements should use photogrammetric meshes due to NeRF's volumetric representation lacking explicit surface definition.

Glossy and Reflective Surfaces

Semi-glossy and reflective objects reveal significant method differences. Missing geometry correlates directly with highlight locations - featureless bright reflections prevent depth estimation, creating visible holes in reconstructed meshes.

NeRF-based approaches circumvent this limitation entirely. By modeling view-dependent appearance, the network learns to reproduce highlights at correct positions without requiring geometric features at those locations. Nerfacto and Splatfacto both achieve visually complete reconstruction of glossy objects, accurately rendering moving highlights that photogrammetry cannot handle.

Quantitative Evaluation

Table 5.6 presents quantitative metrics for novel view synthesis quality. Test views (held out during training) are rendered and compared against ground truth photographs using PSNR, SSIM, and LPIPS metrics.

Table 5.6: Novel view synthesis quality metrics per object for Nerfacto and Splatfacto

Object	**Nerfacto**			**Splatfacto**		
	PSNR ↑	**SSIM ↑**	**LPIPS ↓**	**PSNR ↑**	**SSIM ↑**	**LPIPS ↓**
Banana	24.72	0.864	0.248	36.01	0.980	0.068
Bear	20.34	0.418	0.474	36.01	0.940	0.126
Bilboquet	23.38	0.820	0.367	36.70	0.972	0.077
Cauldron	22.55	0.855	0.310	29.90	0.895	0.117
Dog	20.88	0.692	0.440	40.78	0.983	0.096
Flowerpot	24.88	0.836	0.153	30.95	0.963	0.085
Frog	21.65	0.698	0.455	37.91	0.948	0.092
Pedestal	22.15	0.697	0.207	34.64	0.960	0.103
Plantpot	22.21	0.755	0.471	36.11	0.978	0.073
Pot	12.13	0.572	0.597	20.21	0.892	0.207
Pumpkin	24.73	0.887	0.278	30.70	0.933	0.180
Skull	21.44	0.911	0.192	29.77	0.956	0.095
Spray	21.71	0.925	0.149	29.28	0.964	0.079
Stone	26.08	0.940	0.134	38.69	0.975	0.068
Vase Glass	22.25	0.089	0.132	31.39	0.950	0.089
Average	**22.07**	**0.731**	**0.307**	**33.27**	**0.953**	**0.104**

Splatfacto outperforms Nerfacto across all objects. Average PSNR improvement of 11.20 dB (33.27 vs 22.07) indicates significantly better pixel-level accuracy. SSIM gains (0.953 vs 0.731) show superior structural preservation. LPIPS reduction (0.104 vs 0.307) demonstrates better perceptual quality. The most dramatic improvements appear for challenging objects: Bear improved from SSIM 0.418 to 0.940, and Vase Glass from 0.089 to 0.950. Even the worst-performing object (Pot) achieved acceptable quality with Splatfacto (PSNR 20.21) compared to near-failure with Nerfacto (PSNR 12.13). Splatfacto also trains faster (40 minutes vs 80 minutes for 50,000 iterations at 0.5× resolution). The explicit Gaussian representation provides both quality and speed advantages over implicit volumetric methods for studio-scale bounded scenes.

5.2 Smartphone Polarization Validation

Smartphone-based polarization experiments validated the cross-polarization approach for specular reflection mitigation before full studio integration. A single test object - a highly reflective glazed ceramic vase - was captured with and without polarization filtering to quantify reconstruction improvement.

Experimental Setup

The test object was captured using a smartphone camera (iPhone) mounted on a tripod. Linear polarizers were attached to both the camera lens and the phone's LED flash for cross-polarization setup. The object was placed on the turntable and rotated through 36 positions (10° increments) with the polarizers in two configurations:

- **Without polarization**: Polarizers removed, standard photography
- **With cross-polarization**: Lens and flash polarizers oriented at 90° to each other

Each configuration yielded 36 images processed through RealityCapture for photogrammetric reconstruction.

Visual Comparison

Figure 5.7 presents the reconstruction comparison. Without polarization, the glazed vase exhibits strong specular highlights that move with viewpoint. RealityCapture fails to match these highlight regions across views, resulting in incomplete alignment and substantial geometric holes. With cross-polarization, specular reflections are suppressed, exposing underlying diffuse surface texture. Feature matching succeeds across all views, producing complete surface reconstruction.

(a) Without polarization (b) With cross-polarization

Figure 5.7: Smartphone polarization validation results comparing captured vase reconstructions. (a) Without polarization: reconstruction exhibits substantial geometric holes where specular highlights prevented feature matching. (b) With cross-polarization: specular reflections suppressed, enabling complete surface reconstruction with visible calibration target for scale reference.

The visual comparison demonstrates that cross-polarization effectively suppresses specular reflections while preserving diffuse surface detail. The reconstruction transitions from failed (incomplete geometry with holes) to successful (complete surface coverage). This validation confirms the approach works for specular objects before implementing it in the full studio system.

These results validated the polarization approach for the full studio system integration described in Section 3, demonstrating that cross-polarization enables photogrammetric reconstruction of highly reflective objects that would otherwise fail completely.

5.3 Gränsö Castle Large-Scale Reconstruction

The Gränsö Castle dataset tests reconstruction at building scale. With 5,262 images of the historic castle site, processing demands significant memory and computation time from both photogrammetric and neural rendering software.

Dataset Composition

The dataset combines drone imagery and ground-level SLR photography to achieve complete coverage of the castle exterior and interior spaces. Table 5.7 summarizes the capture distribution.

Table 5.7: Gränsö Castle dataset composition

Capture Method	Image Count	Resolution
SLR Photography	2,981	3840×2560
DJI Drone	2,281	5280×3956
Total	**5,262**	-

Figure 5.8: Merged visualization of Gränsö Castle dataset capture methods. Left side: handheld SLR photography for ground-level and interior coverage. Right side: DJI drone imagery for aerial and facade scanning.

RealityCapture Photogrammetric Reconstruction

RealityCapture successfully processed the complete dataset, producing a unified 3D model spanning the entire castle complex. Processing required chunked alignment - images were divided into spatial regions (courtyard, north wing, south wing, towers) aligned separately, then merged into a complete model.

Figures 5.9 and 5.10 present reconstruction results from aerial and ground-level viewpoints. The reconstructed mesh captures fine architectural details including the neoclassical columns, ornamental dome, window frames, and roof structures across all building sections.

Reconstruction Statistics

The reconstruction processed 5,207 images captured from both drone and handheld cameras. RealityCapture successfully aligned 4,032 images (77.4% success rate), producing a unified 3D model covering the entire castle structure. Table 5.8 summarizes the alignment quality metrics.

Reprojection error measures camera pose accuracy: after estimating 3D point positions and camera poses, each point is projected back onto the images. The error is the pixel distance between this projection and the originally detected feature. Lower values indicate more accurate poses. The mean reprojection error of 0.61 pixels indicates high-accuracy camera pose estimation suitable for neural rendering. Sub-pixel median error (0.49 pixels) confirms that most camera poses achieve precise alignment.

Figure 5.9: Gränsö Castle photogrammetric reconstruction - aerial view showing the main building with central dome, surrounding wings, and courtyard area. The RealityCapture mesh preserves architectural geometry including roof structures and facade details.

Figure 5.10: Gränsö Castle RealityCapture reconstruction - main entrance courtyard view showing white neoclassical facade with columns, dome, and surrounding wings. The photogrammetric mesh captures architectural details including window frames, cornices, and roof structures.

Table 5.8: Gränsö Castle alignment quality metrics from RealityCapture

Metric	Value	Description
Total images	5,207	Input photographs from drone and SLR
Aligned images	4,032 (77.4%)	Images with successfully estimated camera poses
Tie points	5,435,677	3D points triangulated from matched features
Total projections	17,743,863	2D feature observations across all images
Mean reproj. error	0.61 px	Average distance between projected 3D points and detected 2D features
Median reproj. error	0.49 px	Middle value of reprojection errors
Max reproj. error	6.54 px	Worst-case reprojection error
Avg. track length	3.26	Mean number of images each 3D point appears in
Alignment time	2h 28m	Total processing time for feature matching and pose estimation

The 1,175 unaligned images (22.6%) failed primarily at area frontiers (castle perimeter) and in regions where lighting conditions changed during the multi-hour capture session. As the sun position shifted, shadows moved across facades, creating photometric inconsistencies that degraded feature matching. Vegetation-heavy areas at the castle boundaries also contributed to alignment failures. Despite these failures, the resulting mesh provides complete geometric documentation of the main castle structure.

Neural Rendering at Scale

Memory limitations prevent training a single neural rendering model on the complete 5,207-image dataset. GPU memory capacity (32GB on RTX 5090) constrains maximum scene extent and image count. The solution divides the castle into spatial regions, training separate models for different areas.

Figure 5.11 compares Nerfacto and Splatfacto neural rendering results from the same viewpoint.

(a) Nerfacto (b) Splatfacto

Figure 5.11: Neural rendering comparison for Gränsö Castle from identical viewpoints. (a) Nerfacto produces significant blur and artifacts due to training convergence difficulties on the large-scale outdoor dataset. (b) Splatfacto achieves substantially better visual quality with clearer architectural details, windows, and facade structure.

Neural Rendering Quality Metrics

Table 5.9 presents quantitative evaluation of neural rendering methods on the Gränsö Castle dataset.

Table 5.9: Gränsö Castle neural rendering quality metrics

Method	PSNR	Notes
Splatfacto	23.23	Best quality, recommended for real scenes
Nerfacto	14.69	Baseline comparison

Splatfacto significantly outperforms Nerfacto on large-scale outdoor scenes, achieving 23.23 PSNR compared to 14.69. The 8.5 dB improvement demonstrates that explicit Gaussian representations handle unbounded outdoor environments more effectively than implicit volumetric methods.

Method Comparison for Large-Scale Scenes

RealityCapture provides the only complete-scene solution, producing a unified model covering the entire castle. The method handles the full 5,207-image dataset without spatial subdivision. Limitations appear in vegetation-heavy areas and low-texture regions where feature matching fails.

Neural rendering achieves superior visual quality within bounded regions but cannot handle the full dataset scale due to GPU memory constraints. The RTX 5090's 32GB VRAM limits scene extent, requiring spatial subdivision into separate models. This creates visible boundaries between regions and prevents unified scene representation.

For large-scale heritage documentation, photogrammetry remains the practical choice for complete geometric models. Neural rendering serves specialized visualization of high-priority areas where appearance quality matters more than unified geometry.

6 Discussion

The results from Chapter 5 show clear differences between photogrammetry and neural rendering. This chapter examines why each method succeeds or fails in specific situations. We also analyze whether polarization filters actually help with shiny surfaces, based on our smartphone tests and studio experiments.

6.1 Photogrammetry vs. Neural Rendering: Complementary Approaches

Our experiments show photogrammetry and neural rendering work in completely different ways. They're not really competing - each method does certain things better. Understanding when to use which method depends on what you actually need from the reconstruction.

Geometric Accuracy vs. Appearance Fidelity

Photogrammetry is good at getting accurate geometry. RealityCapture produces triangle meshes with clear surfaces that work for measurement, analysis, and 3D printing. The Gränsö Castle model reached 0.61-pixel mean reprojection error across 4,032 aligned images, which means centimeter-level precision at castle scale. For matte objects in the studio dataset, photogrammetry achieved 100% alignment rate on 6 of 15 objects with accurate reconstruction.

Neural rendering focuses on visual quality over geometric precision. NeRF stores scenes as continuous density fields instead of explicit surfaces. This creates better-looking results, especially for shiny surfaces and transparent objects, but makes geometric measurement harder. Getting mesh surfaces from NeRF needs density threshold selection and marching cubes algorithms, which adds uncertainty that photogrammetric meshes don't have.

This basic difference determines which method to use. Heritage documentation that needs accurate dimensions works better with photogrammetry's explicit geometry. VR applications that care about visual quality benefit from neural rendering's appearance modeling. Our hybrid workflow uses both - RealityCapture for geometry, NeRF for visualization.

Feature Matching Assumptions and Failure Modes

Both photogrammetry and COLMAP-based camera pose estimation need the same thing: matching features must be the same physical point that looks similar from different views. When this breaks, both methods fail.

Our studio system breaks this assumption badly. Objects rotate on the turntable but lights stay fixed. A surface facing the light at 0° rotation faces away at 180°. The same point goes from bright to dark. Feature matching algorithms see this brightness change and think they're different features, not the same point under different lighting.

Shiny surfaces make it worse. Specular highlights slide across the surface as viewpoint changes. They show up as bright spots in some images, disappear in others. Both photogrammetry and COLMAP try to match these moving highlights as if they were real features. No wonder COLMAP failed on 85% of our studio datasets.

Neural rendering avoids this by using pre-computed camera poses from RealityCapture. Our ArUco markers work because they look the same regardless of lighting - they're flat patterns, not 3D surfaces with varying illumination. Once we have camera poses from the markers, NeRF can learn how appearance changes with position and viewing direction. It doesn't need consistent appearance for matching.

Here's the key point: neural rendering doesn't actually solve camera pose estimation. It just skips it by using poses from somewhere else. When COLMAP works (like with consistent outdoor lighting), neural rendering doesn't help much with poses. For our studio setup, neural rendering's advantage is modeling appearance after we get poses through markers.

Alternative Solutions to Lighting Inconsistency

There are other ways to handle lighting problems besides using ArUco markers. We can split them into two types: mechanical fixes that keep lighting consistent during capture, and computational methods that separate lighting from appearance afterwards.

Mechanical solutions keep the lighting consistent by moving lights with the object. Two setups work:

- **Coupled rotation**: Object and lights rotate together, keeping the same relative positions. Surface normals always face the same lighting direction, so appearance stays consistent. But you need motorized lights synchronized with the turntable, which gets complicated and expensive.
- **Stationary configuration**: Keep both object and lights fixed while cameras move around. This keeps lighting perfectly consistent but isn't practical. To get 432 views, you'd need to physically move cameras 432 times or buy hundreds of cameras.

Computational relighting methods work with varying lighting but split the appearance into geometry, material properties, and lighting. Neural inverse rendering learns this split from multi-view images, then can relight the object under new conditions. Several methods work for our studio setup:

NeRD (Neural Reflectance Decomposition) [3] splits images into shape, BRDF parameters, and lighting using Disney's BRDF model. It estimates base color, metallic, roughness, and normals for each surface point from multi-view images. The setup matches ours exactly - object rotating under fixed lights. After training, you get a mesh with material textures that renders in real-time under any lighting.

NeRO (Neural Geometry and BRDF Reconstruction) [17] recovers both geometry (as signed distance fields) and BRDF from images under unknown lighting. It handles shiny surfaces without needing masks. For turntable captures, NeRO models direct and indirect lighting, learning material properties despite the changing illumination as objects rotate.

RotatedMVPS (Multi-view Photometric Stereo with Rotated Natural Light) [35] uses neural inverse rendering for shape and material recovery. It exploits a simple fact:

rotating an object under fixed lights is the same as rotating lights around a fixed object. This matches our studio setup perfectly. RotatedMVPS works in two stages: first getting rough BRDF from neural fields, then refining to high-resolution textures.

Deep Dual Loss BRDF Estimation [4] uses CNNs to predict BRDF from images under different lighting. Originally for fixed camera with moving lights, but works for our setup too (moving camera, fixed lights - it's equivalent). A U-Net predicts basecolor, normals, roughness, and metallic values. The loss function checks both rendered output and perceptual similarity.

These relighting methods could replace our ArUco marker approach. Instead of using markers for alignment, neural inverse rendering could figure out geometry, materials, and lighting all at once from the images. The lighting changes that break photogrammetry would actually help train the material decomposition. Plus you'd get relightable objects, not just geometry.

But relighting has its own problems. You need to tune network architectures and loss functions carefully. The methods need enough lighting variation to work - our single turntable rotation might not be enough, though the 12-camera setup helps. Shiny or transparent materials still confuse these methods. Training takes longer too, usually 4-8 hours for our object sizes versus 2-3 hours for standard NeRF.

Our current approach - ArUco markers for poses, then neural rendering for appearance - works reliably. Marker alignment doesn't need parameter tuning like inverse rendering does. Once we have poses, regular NeRF gives good results quickly. Later we could add relighting to the marker-aligned captures, getting both reliable geometry and material properties.

View-Dependent Appearance Modeling

The biggest performance gap shows up with view-dependent effects. Photogrammetry assumes Lambertian reflectance - surfaces look the same from all angles. Multi-View Stereo needs to match surface appearance across views. But specular highlights break this completely. They slide across surfaces as the viewpoint changes, making matching impossible.

The numbers show this clearly. Two studio objects failed photogrammetry entirely: Spray (featureless white plastic) and Vase Glass (transparent material). The reflective Cauldron achieved only 92% alignment despite being metallic. Photogrammetry can't reconstruct surfaces it can't match.

Neural rendering handles view-dependence directly. NeRF conditions color output on viewing direction, so it learns where highlights should appear. The network figures out that certain surface points look bright from specific angles and dark from others. This captures glossy surfaces without needing any geometric features at highlight locations.

Glass objects are even harder. Light bends through glass, breaking the epipolar geometry that photogrammetry needs. RealityCapture completely failed on our Vase Glass object due to transparent/refractive surfaces. Nerfacto modeled the glass properly using volumetric rendering, achieving 22.25 dB PSNR, while Splatfacto reached 31.39 dB. This isn't just better quality - it's a different capability entirely.

Computational Requirements and Scalability

Photogrammetry handles large datasets better than neural rendering right now. The Gränsö Castle reconstruction processed 5,207 images with 4,032 successfully aligned. Alignment took about 2.5 hours but gave us a complete model ready for analysis, measurement, and visualization.

Neural rendering hits memory walls fast. The RTX 5090's 32GB VRAM can't fit the whole castle dataset. Splitting it into sections works but creates visible seams at boundaries and loses the unified model. The Gränsö neural rendering comparison (Figure 5.11) shows Nerfacto struggling with significant blur while Splatfacto achieves substantially better results.

Studio datasets work fine though. With 432 images of bounded objects, Nerfacto trains in about 80 minutes, Splatfacto in about 40 minutes for 50,000 iterations. Quality metrics look good (Splatfacto averages 33.27 dB PSNR, 0.953 SSIM) and everything fits in GPU memory without issues.

This scalability gap points to different use cases. Photogrammetry fits large-scale work - heritage sites, city models, terrain mapping - where coverage beats visual quality. Neural rendering suits smaller captures - products, museum pieces, film assets - where appearance matters more than scale.

Image Quality and Blur Effects

The Gränsö Castle neural rendering results reveal how input image quality directly impacts reconstruction. Our SLR camera used a 50mm lens, but the focus and aperture settings were not manually adjusted during the capture session. This oversight created depth-of-field limitations when capturing architectural details at varying distances. Objects outside the optimal focus range appear blurred in the input images.

Motion blur and defocus blur both degrade reconstruction quality [27]. For photogrammetry, blur reduces feature detection accuracy - SIFT and similar detectors find fewer keypoints in blurred regions, leading to sparser point clouds and potential alignment failures. Recent research on camera settings for neural rendering confirms that image sharpness significantly affects both NeRF and Gaussian Splatting quality [24].

Neural rendering handles blur differently than photogrammetry. NeRF tries to learn a sharp scene representation from blurred inputs, but the network cannot recover details that were never captured. The result is a blurry reconstruction that averages the input blur across viewpoints. Splatfacto's explicit Gaussian representation may handle this better since each Gaussian can adapt its shape independently, but fundamentally both methods are limited by input image quality.

For the Gränsö dataset specifically, several factors compounded the blur problem. The multi-hour capture session meant changing sun positions and shadows. Handheld SLR photography at 3840×2560 resolution required careful stabilization to avoid motion blur. Without adjusting focus between shots, distant architectural details (roof ornaments, tower tops) appeared softer than nearby elements (entrance columns, windows at eye level).

Future large-scale captures should manually adjust focus settings throughout the session or use multiple capture passes at different focus distances. Focus stacking—combining multiple exposures at different focus points—could provide sharp details across the full depth range. Alternatively, computational approaches like deblurring as preprocessing might help, though they risk introducing artifacts.

Training Data Requirements

Something surprising emerged from our experiments. Neural rendering got complete reconstructions from 432 images on all 15 objects while photogrammetry failed on 2 objects entirely (Spray, Vase Glass). Why does NeRF work when photogrammetry fails with identical data?

The methods need fundamentally different things from images. Photogrammetry needs texture features to match between views. Specular highlights have no features - they're just bright spots. Taking more photos of a mirror doesn't create texture that isn't there.

Neural rendering just needs enough views to constrain the volume function. Those "useless" specular highlights actually help NeRF learn view-dependent appearance. Each photo showing a highlight from a different angle teaches the network how brightness changes with viewing direction. More views help NeRF even when they can't help photogrammetry.

This affects how datasets should be captured. For photogrammetry, surfaces need texture - either natural or added through matte spray or polarization. For neural rendering, dense

angular coverage matters more than surface texture. Our 12 cameras at 36 rotations give enough views for NeRF even when photogrammetry struggles.

6.2 Polarization Filtering for Specular Surface Reconstruction

Our polarization tests showed cross-polarization really works for shiny objects. The smartphone experiment was clear: a glazed vase that failed reconstruction without polarization achieved complete surface coverage with cross-polarized filters.

Physical Mechanism and Effectiveness

Cross-polarization cuts out specular reflections but keeps diffuse ones. This fixes photogrammetry's main problem with shiny surfaces. Specular reflections that caused complete reconstruction failure were effectively eliminated while preserving surface texture information.

The physics is straightforward. Specular reflections happen at the surface and keep their polarization (following Fresnel equations). Diffuse reflections come from subsurface scattering - light bounces around inside the material and loses polarization. Cross-polarized filters block the polarized specular light but let through the unpolarized diffuse light.

The visual comparison in Figure 5.7 demonstrates the dramatic improvement: going from failed reconstruction with geometric holes to complete surface coverage. Once the highlights disappeared, the actual surface texture showed through and could be matched between views.

Comparison with Neural Rendering Alternative

Polarization and neural rendering solve the specular problem differently. Polarization changes the capture to remove highlights before they reach the camera. Neural rendering captures everything and figures out the view-dependence computationally.

Polarization has practical benefits. It works with standard photogrammetry pipelines - just add filters to the cameras. Processing uses the same RealityCapture workflow. The output is a normal mesh that works in any 3D software.

Neural rendering needs no hardware changes but takes longer to compute. Nerfacto needs about 80 minutes of training, Splatfacto about 40 minutes. The volumetric output makes measurements tricky. GPU memory limits how big scenes can be.

In the studio setup, polarization makes more sense for photogrammetry workflows. Polarizing film for twelve cameras costs maybe $200. That's cheaper than hours of GPU time for every object. Heritage documentation needs meshes, not volume fields.

But neural rendering handles things polarization can't. Glass refracts light no matter what filters are used. Mixed materials (chrome details on plastic) might need different polarizer angles for different parts. NeRF learns all these effects without any hardware setup.

Integration Challenges and Future Development

The smartphone test worked, but putting polarizers on all twelve studio cameras gets complicated. Each filter needs precise alignment with the area lights for cross-polarization. Angular misalignment reduces effectiveness significantly.

Light loss is a real problem. Polarizers reduce incoming light, and cross-polarization cuts additional diffuse light. To compensate, exposure times must increase or ISO must rise. Either way risks motion blur or noise.

Vignetting happens when cameras look at steep angles through the polarizers. The studio's 12 cameras span 0-90° elevation, so vignetting gets worse at extreme angles than in the simple phone test.

Future systems could rotate polarizers for each object. Shiny things get cross-polarization, matte objects skip the filters entirely. No point losing light when the surface isn't reflective anyway.

Dataset Creation Implications

Polarization opens up photogrammetry to many more objects. Museum pieces, machine parts, and products often have shiny surfaces that fail reconstruction. Our smartphone test demonstrated that polarization can transform complete failure into complete success for reflective objects. This makes datasets possible without spraying everything with matte coating.

Surface treatments are often unacceptable anyway. Museums won't let researchers spray artifacts with dulling spray. Industrial parts need cleaning afterwards. Polarization works without touching the object. The images still show real surface properties (just without highlights), keeping their value for documentation and analysis.

Neural rendering datasets face different trade-offs with polarization. Cross-polarized images lack the highlights that NeRF would normally learn to reproduce. This might make the final rendering less realistic. On the other hand, removing highlights might help training converge faster since the appearance function is simpler. One option: capture both versions, use polarized for camera alignment, unpolarized for neural rendering.

Generalization Beyond Studio Capture

Polarization isn't just for studios. The Gränsö Castle had problems with water reflections and shiny windows that hurt reconstruction. Cross-polarization could help there too.

But outdoor polarization is harder. Sunlight's polarization changes with sun angle and atmospheric conditions. The sky has its own polarization pattern that shifts through the day. Water reflects polarized sky differently than walls reflect direct sun. One polarizer angle can't handle everything at once.

Solutions might include rotating polarizers during capture or taking multiple shots at different polarizer angles. Some new cameras have polarization sensors built into each pixel. These could separate specular and diffuse computationally without physical filters - no light loss, no vignetting, same benefits.

7 Conclusion

This thesis investigated neural rendering dataset creation through two capture scenarios: a controlled studio system and large-scale outdoor heritage documentation. The work produced two publicly available datasets, capture software, and comparative analysis of photogrammetry versus neural rendering methods.

7.1 Summary of Contributions

The studio capture system uses twelve synchronized industrial cameras on a quarter-hemisphere arc with a motorized turntable. Each object yields 432 images across 36 rotation positions. Custom software (7,950 lines: C# WinUI frontend and C++ hardware backend) controls camera synchronization, turntable automation, and data organization. The resulting dataset contains 15 objects with materials ranging from diffuse to transparent, deliberately including cases where photogrammetry fails.

The Gränsö Castle dataset documents a Swedish heritage site with 5,262 images from drone and SLR photography. RealityCapture aligned 4,032 images (77.4%) with 0.61-pixel mean reprojection error. This dataset tests neural rendering scalability on architectural scenes with varying natural lighting.

Both datasets and the capture software are released as open-source resources.

7.2 Answers to Research Questions

RQ1: What are the practical challenges of creating multi-view datasets for neural rendering?

Studio capture requires solving camera synchronization, consistent lighting, and robust pose estimation. Our system addresses synchronization through hardware triggering and lighting through diffuse area lights. The key challenge was pose estimation: COLMAP aligned only 2 of 432 images due to lighting inconsistency as objects rotate under fixed lights. ArUco markers solved this by providing viewpoint-independent features for RealityCapture alignment.

Outdoor capture faces different problems. The SLR's focus and aperture settings were not adjusted during the session, creating depth-of-field limitations that caused blur at certain distances. The multi-hour session produced lighting changes as sun position shifted. Combining

drone and handheld imagery required careful overlap planning. These challenges highlight the importance of manual camera adjustments during large-scale outdoor work.

RQ2: Which dataset characteristics matter most for neural rendering quality?

Camera pose accuracy proved critical. Objects achieving sub-pixel reprojection error (0.53–0.76 px) in RealityCapture produced high-quality neural rendering results. Splatfacto averaged 33.27 dB PSNR across studio objects with accurate poses.

Image sharpness directly impacts reconstruction quality. Blur from depth-of-field limitations or motion degrades both photogrammetry feature detection and neural rendering detail. The Gränsö neural rendering comparison shows Splatfacto (23.23 dB) substantially outperforming Nerfacto (14.69 dB), partly because Splatfacto handles image quality variations more robustly.

Angular coverage matters more than raw image count for neural rendering. The 12-camera arc provides diverse viewpoints that help constrain the learned scene representation even when individual images have limited texture.

RQ3: Which materials are hardest to capture?

Photogrammetry failed on two studio objects: Spray (featureless white plastic) and Vase Glass (transparent/refractive). Both Nerfacto and Splatfacto successfully reconstructed these objects, demonstrating neural rendering's advantage for view-dependent materials.

Reflective surfaces (Cauldron, Pot) achieved lower photogrammetry alignment rates (88–92%) due to specular highlights disrupting feature matching. Neural methods handled these without issue since they model view-dependent appearance directly.

Cross-polarization filtering provides an alternative solution. Our smartphone validation showed polarization transforms complete reconstruction failure into success for glossy surfaces by eliminating specular reflections.

7.3 Implications

For neural rendering researchers, these datasets provide standardized benchmarks including deliberate failure cases. The documented capture protocols enable reproduction and extension of the work.

For heritage documentation practitioners, the results suggest a hybrid workflow: photogrammetry for geometric models and measurement, neural rendering for high-quality visualization of bounded areas. Large-scale neural rendering remains constrained by GPU memory, limiting its applicability to complete heritage sites.

For capture system designers, ArUco markers provide robust alignment when lighting inconsistency prevents standard feature matching. This approach works for both controlled studio environments and challenging outdoor conditions.

7.4 Future Work

Several directions could extend this work:

Polarization integration. The smartphone validation demonstrated polarization effectiveness. Integrating cross-polarization filters into the 12-camera studio system would enable systematic comparison of polarized versus unpolarized capture for the full object dataset.

Focus stacking for outdoor capture. The depth-of-field limitation could be addressed through multiple capture passes at different focus distances, combined computationally. This would provide sharp details across the full depth range of large-scale scenes.

Neural inverse rendering. Methods like NeRD and NeRO decompose appearance into geometry, materials, and lighting. Applying these to our studio dataset could produce relightable models rather than fixed-lighting reconstructions.

Dynamic scene capture. Extending the turntable system to capture moving objects or deformable materials would address 4D reconstruction scenarios increasingly relevant for film and gaming applications.

Larger-scale neural rendering. Current GPU memory limits (32GB) prevent unified neural models of complete heritage sites. Hierarchical approaches or model compression techniques could enable neural rendering at architectural scale.

Bibliography

[1] Jonathan T Barron, Ben Mildenhall, Dor Verbin, Pratul P Srinivasan, and Peter Hedman. "Mip-NeRF 360: Unbounded Anti-Aliased Neural Radiance Fields". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2022), pp. 5470–5479.

[2] Jonathan T Barron, Ben Mildenhall, Dor Verbin, Pratul P Srinivasan, and Peter Hedman. "Zip-NeRF: Anti-Aliased Grid-Based Neural Radiance Fields". In: *International Conference on Computer Vision (ICCV)*. 2023.

[3] Mark Boss, Raphael Braun, Varun Jampani, Jonathan T Barron, Ce Liu, and Hendrik P A Lensch. "NeRD: Neural Reflectance Decomposition from Image Collections". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 2021, pp. 12684–12694.

[4] Valentin Deschaintre, Miika Aittala, Frédo Durand, George Drettakis, and Adrien Bousseau. "Single-Image SVBRDF Capture with a Rendering-Aware Deep Network". In: *ACM Transactions on Graphics (TOG)*. Vol. 37. 4. 2018, pp. 1–15.

[5] Jiahui Fan, Fujun Luan, Jian Yang, Miloš Hašan, and Beibei Wang. "Free Your Hands: Lightweight Relightable Turntable Capture Pipeline". In: *arXiv preprint arXiv:2503.05511* (2025).

[6] Xinyi Fan, Linguang Zhang, Benedict Brown, and Szymon Rusinkiewicz. "Automated View and Path Planning for Scalable Multi-Object 3D Scanning". In: *ACM Transactions on Graphics (Proc. SIGGRAPH Asia)* 35.6 (2016), p. 239. DOI: 10.1145/2980179.2980225.

[7] A. Ibáñez García-Moreno, J. L. L. Fernández, V. Arévalo, and D. González-Aguilera. "Seeing beyond vegetation: A comparative occlusion analysis between Multi-View Stereo, Neural Radiance Fields and Gaussian Splatting for 3D reconstruction". In: *International Journal of Applied Earth Observation and Geoinformation* 136 (2025), p. 104258.

[8] Sergio Garrido-Jurado, Rafael Muñoz-Salinas, Francisco José Madrid-Cuevas, and Manuel Jesús Marín-Jiménez. "Automatic generation and detection of highly reliable fiducial markers under occlusion". In: *Pattern Recognition* 47.6 (2014), pp. 2280–2292.

[9] Xiaodong Gu, Zhiwen Fan, Siyu Zhu, Zuozhuo Dai, Feitong Tan, and Ping Tan. "Cascade cost volume for high-resolution multi-view stereo and stereo matching". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 2495–2504.

[10] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. 2nd. Cambridge University Press, 2003.

[11] Peter Hedman, Julien Philip, True Price, Jan-Michael Frahm, George Drettakis, and Gabriel Brostow. "Deep Blending for Free-Viewpoint Image-Based Rendering". In: *ACM Transactions on Graphics (SIGGRAPH Asia)* 37.6 (2018), pp. 1–15.

[12] Mike R James, Stuart Robson, Sebastian d'Oleire-Oltmanns, and Uwe Niethammer. "Optimising UAV topographic surveys processed with structure-from-motion: Ground control quality, quantity and bundle adjustment". In: *Geomorphology* 280 (2017), pp. 51–66.

[13] Rasmus Jensen, Anders Dahl, George Vogiatzis, Engin Tola, and Henrik Aanæs. "Large scale multi-view stereopsis evaluation". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2014), pp. 406–413.

[14] 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), pp. 1–14.

[15] Wooseok Kim, Taiki Fukiage, and Takeshi Oishi. "REF^2-NeRF: Reflection and Refraction aware Neural Radiance Field". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. arXiv:2311.17116. 2024.

[16] Arno Knapitsch, Jaesik Park, Qian-Yi Zhou, and Vladlen Koltun. "Tanks and temples: Benchmarking large-scale scene reconstruction". In: *ACM Transactions on Graphics* 36.4 (2017), pp. 1–13.

[17] Yuan Liu, Peng Wang, Cheng Lin, Xiaoxiao Long, Jiepeng Wang, Lingjie Liu, Taku Komura, and Wenping Wang. "NeRO: Neural Geometry and BRDF Reconstruction of Reflective Objects from Multiview Images". In: *ACM Transactions on Graphics (SIGGRAPH)* 42.4 (2023).

[18] Thomas Luhmann, Clive S. Fraser, and Hans-Gerd Maas. "Sensor modelling and camera calibration for close-range photogrammetry". In: *ISPRS Journal of Photogrammetry and Remote Sensing* 115 (2016), pp. 37–46. DOI: 10.1016/j.isprsjprs.2015.10.006.

[19] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. "NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis". In: *European Conference on Computer Vision (ECCV)*. 2020, pp. 405–421.

[20] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. "Instant Neural Graphics Primitives with a Multiresolution Hash Encoding". In: *ACM Transactions on Graphics* 41.4 (2022), pp. 1–15.

[21] Francesco Nex and Fabio Remondino. "UAV for 3D mapping applications: a review". In: *Applied Geomatics* 6.1 (2014), pp. 1–15.

[22] OpenCV Contributors. *Detection of ArUco Markers*. https://docs.opencv.org/4.x/d5/dae/tutorial_aruco_detection.html. Accessed: 2025-09-21. 2024.

[23] Qervas. *CamMatrixCapture: Multi-Camera Neural Rendering Capture System*. GitHub repository. 2025. URL: https://github.com/Qervas/CamMatrixCapture.

[24] Deyan Rangelov, Sibren Waanders, Koen Waanders, Maurice van Keulen, and Rumen Miltchev. "Impact of Camera Settings on 3D Reconstruction Quality: Insights from NeRF and Gaussian Splatting". In: *Sensors* 24.23 (2024), p. 7594. DOI: 10.3390/s24237594.

[25] Fabio Remondino, Maria Grazia Spera, Erica Nocerino, Fabio Menna, and Francesco Nex. "State of the art in high density image matching". In: *The Photogrammetric Record* 29.146 (2014), pp. 144–166.

[26] Johannes L Schönberger and Jan-Michael Frahm. "Structure-from-motion revisited". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 4104–4113.

[27] Till Sieberth, Rene Wackrow, and Jim H. Chandler. "Motion blur disturbs – the influence of motion-blurred images in photogrammetry". In: *The Photogrammetric Record* 29.148 (2014), pp. 434–453.

[28] Jia-Mu Sun, Tong Wu, Ling-Qi Yan, and Lin Gao. "NU-NeRF: Neural Reconstruction of Nested Transparent Objects with Uncontrolled Capture Environment". In: *ACM Transactions on Graphics (Proc. SIGGRAPH Asia)* 43.6 (2024). DOI: 10.1145/3687757.

[29] Matthew Tancik, Ethan Weber, Evonne Ng, Ruilong Li, Brent Yi, Terrance Wang, Alexander Kristoffersen, Jake Austin, Kamyar Salber, Abhik Blanber, et al. "Nerfstudio: A Modular Framework for Neural Radiance Field Development". In: *ACM SIGGRAPH 2023 Conference Proceedings*. 2023, pp. 1–12.

[30] Teledyne DALSA. *Sapera LT SDK User Manual: Industrial Camera Control and Image Acquisition*. Tech. rep. Teledyne Vision Solutions, 2023.

[31] 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.

[32] MJ Westoby, James Brasington, NF Glasser, MJ Hambrey, and JM Reynolds. "Structure-from-Motion photogrammetry: A low-cost, effective tool for geoscience applications". In: *Geomorphology* 179 (2012), pp. 300–314.

[33] Guanjun Wu, Taoran Yi, Jiemin Fang, Lingxi Xie, Xiaopeng Zhang, Wei Wei, Wenyu Liu, Qi Tian, and Xinggang Wang. "4D Gaussian Splatting for Real-Time Dynamic Scene Rendering". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2024.

[34] Tong Wu, Yu-Jie Yuan, Li-Xin Zhang, Jie Yang, Yan-Pei Cao, Ling-Qi Yan, and Lin Gao. "Recent Advances in 3D Gaussian Splatting". In: *Computational Visual Media* 10.4 (2024), pp. 613–642.

[35] Songyun Yang, Yufei Han, Jilong Zhang, Kongming Liang, Peng Yu, Zhaowei Qu, and Heng Guo. "RotatedMVPS: Multi-view Photometric Stereo with Rotated Natural Light". In: *arXiv preprint arXiv:2508.04366* (2025). Self-supervised MVPS framework based on neural inverse rendering under rotated environment light.

[36] 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)*. 2018, pp. 767–783.

[37] Yao Yao, Zixin Luo, Shiwei Li, Jingyang Zhang, Yufan Ren, Lei Zhou, Tian Fang, and Long Quan. "BlendedMVS: A Large-scale Dataset for Generalized Multi-view Stereo Networks". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 1790–1799.

[38] Chandan Yeshwanth, Yueh-Cheng Liu, Matthias Nießner, and Angela Dai. "ScanNet++: A High-Fidelity Dataset of 3D Indoor Scenes". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2023, pp. 12–22.

[39] Shaoxuan Yin. *Captured Objects Dataset*. Internet Archive. 209 GB uncompressed, Public Domain. Oct. 2025. URL: https://archive.org/details/captured_objects_dataset.

[40] 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*. 2018, pp. 586–595.

[41] Zhengyou Zhang. "A flexible new technique for camera calibration". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22.11 (2000), pp. 1330–1334.