Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

School of Computer Science and Statistics

Investigating factors that impact 2D - 3D Reconstruction Quality

Ciarán Coady

May 3, 2022

A dissertation submitted in partial fulfilment
of the requirements for the degree of
Masters in Computer Science

Declaration

I hereby declare that this dissertation is entirely my own work and that it has not been submitted as an exercise for a degree at this or any other university.

I have read and I understand the plagiarism provisions in the General Regulations of the University Calendar for the current year, found at `http://www.tcd.ie/calendar`.

I have also completed the Online Tutorial on avoiding plagiarism 'Ready Steady Write', located at `http://tcd-ie.libguides.com/plagiarism/ready-steady-write`.

Signed: _____ Date: _____

Permission to Lend and/or Copy

I, the undersigned, agree that Trinity College Library may lend or copy this thesis upon request.

Signed: _____ Date: _____

Abstract

Advancements in mobile and Augmented Reality technologies have both increased the complexity of their design and improved their accessibility. This shift has opened the door for more affordable AR based content creation through the use of 3D reconstruction tools, and images captured from modern smartphones. Using open-source software tools, this research investigates how different environmental factors impact 3D reconstruction quality. Existing literature was leveraged to identify the most prominent quality impacting factors, which include image resolution, number of images, distribution of images within the scene, object of interest texture and lighting conditions. From this knowledge, an automated framework for capturing and reconstructing a synthetic scene was designed and implemented. Using this testing framework, a diverse set of synthetic scenes were created to model the identified impacting factors. These scenes were captured and reconstructed, with their final reconstruction quality being evaluated. From this testing, it was found that increasing visible geometry and detailed texturing has the capability to drastically improve final reconstruction quality. It was confirmed that controlled lighting conditions with minimal hard shadows, such as on an overcast day, produces the best results for a given scene. Finally, it was determined that capturing images in an evenly distributed circular pattern around a focus point within the scene produces the optimal scene coverage and thus, reconstruction results. It is hoped that this research can be used to further the understanding of the 3D reconstruction quality impacting factors, enable automated synthetic scene evaluation and ultimately, inform and guide end-users on how best to capture their scene of interest.

Acknowledgements

I'd like to thank my supervisor Michael for his knowledge, help and support throughout our time working together. I'd like to thank Jan for his advice and time helping me to understanding the intricacies of capture studio setup and camera calibration. I'd like to thank Heather for her endless support throughout the last five years, it has made my college experience and unforgettable one. I'd like to thank my family and friends for their patience and kindness. I would like to thank the active contributors on the COLMAP project for their time and help solving issues I had with the software during implementation.

Contents

List of Figures

List of Tables

1 Introduction

Advancements in 3D graphics hardware and software has led to numerous improvements across many fields, resulting in more immersive games, enhanced assistive technologies and augmented learning environments. These advancements in technology are constantly pushing the boundaries of 3D graphics as a medium for expression and communication, while also making it more accessible than ever with more affordable and compact hardware. This increase in performance and creativity has raised user expectation and by extension the demand for more complex and detailed 3D graphics. This demand for detail and realism has resulted in more work for visual artists, designers and developers to build such applications within current hardware and time constraints. In search of realism in our virtual worlds, the desire to recreate faithful models of real world objects and scenes has come to the forefront.

Traditionally, to obtain a high quality 3D capture required an expensive capture studio and capture equipment, primarily using active scanning techniques such as LiDAR. In recent times, we have seen successful 3D reconstruction being performed for applications such as autonomous driving, where real-time robust scene understanding and perception is favoured over absolute quality. With the public eye taking greater interest in these applications, we see an increased demand for lower cost capture solutions that are more accessible. This enables the use of 2D to 3D reconstruction in applications like reconstructing historical artifacts for archaeological research or indie game development. Solutions leveraging hardware such as Microsoft's Kinect or smartphone sensors have greatly improved the accessibility of 3D reconstruction to the point where virtually anyone can perform a reconstruction with little to no prior knowledge using hardware they already have.

Driving this accessibility are passive 3D reconstruction techniques. These techniques are being used to bridge the hardware gap and digitize real world objects and environments from nothing more than a set of images. These passive methods allow users to take an unordered image set, captured by hand, and reconstruct a scene of interest digitally. There are numerous software solutions that facilitate this process, ranging from methods rooted in traditional computer vision to those using deep learning methods, which aim to improve and expedite the reconstruction process. Open source solutions such as COLMAP provide an

easy to use GUI compared with other approaches which are controlled from the command line.

Reconstruction is a non-trivial task and reconstructing an arbitrary scene is even more difficult due to varying environmental factors. The quality of the final reconstruction depends massively on the images supplied by the user, resolution, level of detail, amount of overlap, number of images, distribution within the environment etc. These are all factors that an inexperienced user will not be well-versed in. As such, while reconstruction software is more accessible than ever, there remains an ever increasing knowledge gap for users to overcome in order to achieve the best possible reconstruction quality. To better understand these factors and their impacts, we must first be able to evaluate the quality of a reconstruction. This evaluation is a non-trivial problem and varies drastically based on the target application, with there being no one size fits all approach to quality assessment.

In an ideal scenario, having ground truth information to compare against gives us the best chance of measuring quality. However, capturing high-quality ground truth data can be both expensive and time-consuming. Without this data it becomes difficult to quantitatively evaluate reconstruction performance. There is no definitive evaluation measure for these techniques unfortunately. Previous research often uses a fixed image-set to benchmark reconstruction methods against each other. With the computation expense of reconstructions, it makes sense to try and get it right the first time and only do the necessary amount of computation to get the desired quality. Capturing too few images will result in poor scene coverage, but equally capturing too many increases the run-time and resources required to perform the reconstruction, with little or no quality improvement.

In this research, I investigate how different environmental factors impact reconstruction quality, with the goal of better understanding how to mitigate negative impact and maximise positive impact. To do this, I create an automated framework for capturing and reconstructing a synthetic scene. By reconstructing a completely synthetic scene, we have an exact ground truth to which we can evaluate the performance of the technique in a controlled environment. In this way, not only can we gain an accurate measure of how well a reconstruction was performed, but also determine what factors play the most significant roles in quality. By better understanding these factors, we can better inform and guide users how best to capture their scene of interest. More formally, the research objectives are listed below.

Research objectives :

1. Understand 2D to 3D reconstruction techniques.
2. Investigate and understand the state of the art in Structure-From-Motion.
3. Determine what are the prominent quality impacting factors.

4. Design experiments to evaluate the selected reconstruction technique in the identified impacting areas.
5. Evaluate the obtained performance using pertinent metrics.
6. Form conclusions based on experimental results.

2 Background Research

2.1 3D Reconstruction

3D reconstruction is the procedure of recovering 3D information and understanding from an environment or scene. The 3D reconstruction process and its components are firmly rooted in both the fundamentals of computer vision and computer graphics, with both working in unison to produce results which underpin many of the digital experiences we take for granted today such as visual effects in movies [1][2]. At its core, 3D reconstruction is a balance of application needs, available hardware and environmental factors. The type of reconstruction method used is influenced by the application area being targeted, whether it is real-time navigation or high quality model capture. To meet these application requirements there are many approaches to 3D reconstruction, each with their own limitations in terms of hardware, time to compute and quality of the result obtained. These techniques and their intended applications broadly fall under the categories of real-time and offline reconstruction.

2.1.1 Real-time applications

In real-time applications, 3D reconstruction is primarily used to gain scene understanding. These reconstructions generally do not directly produce a 3D model or mesh, but rather form an estimation that can be used for depth perception, object segmentation and localisation. In these applications the scene is dynamic and changing relative to the capture device, with the goal for the capture system being to understand and react to these changes. Two popular application areas that employ real-time 3D reconstructions are autonomous vehicle navigation and Augmented Reality (AR). For navigation applications, scene understanding feeds algorithms which are designed to safely maneuver city streets, roadways, car parks and numerous other scenarios. For AR applications gaining scene understanding facilitates overlaying 3D objects accurately within the scene for greater user immersion. These applications have deeply penetrated society today, with examples spanning from assisted-driving and self-driving cars from Tesla and Google, to gaming with Pokemon Go, to social media such as Snapchat. Simultaneous Localisation and Mapping (SLAM) is an example of such a real-time 3D reconstruction technique which can utilize a multitude of

hardware sensors or be solely image based (Visual-SLAM) to facilitate the needs of these applications [3] [4] [5].

2.1.2 Offline applications

In contrast to real-time applications, offline applications are primarily focused on achieving the highest possible reconstruction quality, even if it comes at high computational expense and long run-times. Interest in offline 3D reconstruction spans many fields including Medical [6], Search and Rescue [7], Aerial Imaging [8] [9], Cultural heritage & tourism [10], Geoscience [11] and AR applications [12]. For these applications, the desired outcome is a high quality 3D point cloud or 3D mesh capturing an area of interest. A well understood and researched method of reconstruction for these scenarios is Structure-from-Motion (SfM). In contrast to SLAM, SfM focuses on reconstructing a static scene captured from many viewpoints, with scene understanding being used to recover intricate geometry and minute details to produce a model. With applications becoming more complex by the day and computation power increasing, we have seen the desire for higher detail in areas such as gaming with high profile examples like Microsoft Flight Simulator, pulling in satellite imaging to perform a global scale 3D reconstruction.

The alternative to reconstruction for these applications is to model 3D assets by hand. This process requires a high level of skill, creating a barrier to entry and for those with the skill-set, it is a time-consuming process. Reconstructing objects bridges the gap between virtual and physical worlds with real world objects being scanned and ported to the virtual world in high fidelity.

2.2 Category of Reconstruction Technique

To facilitate both real-time and offline application targets, a number of varying reconstruction techniques have been developed. These techniques fall under the categories of active and passive reconstruction.

2.2.1 Active Reconstruction

Active reconstruction techniques employ methods to probe the scene of interest, measuring distances, distortion properties and discovering occlusions. One well known example of an active reconstruction technique is Light Detection and Ranging (LiDAR). LiDAR capture works by emitting light pulses into the scene and measuring the time of flight for these pulses to return to the device. Using this information we can gain an understanding of the scene and produce an accurate dense point cloud representation of its geometry in real-time. In the past, LiDAR has been used effectively to produce accurate reconstructions of large

scale environments such as Dublin City [13]. Similarly, in real-time applications it is used for navigation in autonomous vehicles using algorithms such as SLAM. LiDAR is a compelling option for high quality real-time applications, however it is not without its limitations. Objects must reflect light to be effectively captured by LiDAR and as such LiDAR struggles with transparent objects. More detail on LiDAR can be found here [14] [15].

Structured light is another active reconstruction technique, which uses similar principles to LiDAR but on a smaller scale. By emitting a known light pattern using an IR dot emitter into the scene and capturing how it distorts and deforms, we can capture scene structure. Depending on the resolution of the dot emitter, it becomes possible to form accurate reconstructions. This technology is positioned as a cheap compact alternative to the expensive bulky LiDAR equipment, while still being capable of complex sensing applications such as performing facial recognition in modern smartphones or pose tracking in games using Microsoft's Kinect. This type of reconstruction does have limitations, such as the inability to be used effectively outdoors due to other IR light emissions from sources such as the sun. IR capture also has a limited capture distance when compared with other active solutions such as LiDAR. In ideal conditions it has been shown that this type of hardware can be used to great effect in performing real-time reconstruction tasks [16]. However, while suitable for real-time reconstruction and understanding of dynamic scenes, this technique can often only produce moderate quality results when creating models. This is due to the characteristics of most available consumer hardware devices, which do not target this use-case and thus do not capture in a high enough resolution. These factors make it difficult to effectively reconstruct larger scenes such as buildings and other outdoor environments.

With careful planning, LiDAR can be used to great effect to perform large scale, accurate 3D reconstruction and equally can be used for supporting complex real-time applications. Cheaper IR backed solutions offer a compelling alternative should the environment and application be suitable. However, with active techniques there are concerns when capturing delicate and sensitive scenes. Objects such as archaeological artifacts may be sensitive to light emission and thus could be damaged by active scanning techniques. Situations like this provide the need for alternative reconstruction approaches.

2.2.2 Passive Reconstruction

Passive reconstruction techniques use observed scene attributes to perform reconstruction. These observed attributes are commonly captured by 2D image collections. This makes them ideally suited to reconstruct delicate historical artifacts and other such sensitive scenes where unwanted radiation emission should be minimised for fear of damage or degradation. Owing to the fact that these reconstruction techniques use images as their input, it opens

up a huge opportunity for the masses to perform 3D reconstructions using nothing more than a digital camera. The amount of information gathered by these passive techniques are inherently more compressed when compared with active techniques. Points of interest must be matched between images and then processed in such a way that they can be then reprojected back in to 3D space, rather than just directly obtaining the 3D points themselves as done in active methods. In this way, there exists a much higher computational load, which lies in the software rather than the hardware. This means purely passive techniques in general struggle to balance both computation efficiency and accuracy, depending on the targets of the application. However, there exists camera systems that help in this process, beyond just a standard perspective camera.

Light field cameras are a recent area of research, which hold significant promise when retrieving depth information from a scene and are particularly useful when creating a depth map [17]. However, the cameras themselves are quite expensive and many of the more affordable variants of these cameras capture at a lower resolution, thus impacting reconstruction quality and their potential. Similarly, stereo camera systems mimic characteristics of the human vision system and utilize spatial disparity along a common axis to perceive depth. There are limitations to these stereo systems, such as the amount of disparity possible before overlap no longer occurs. However, using stereo pairs have been shown to perform well when reconstructing scene depth information. Despite their limitations, the knowledge gained from the development of these systems can be used to improve elements of the SfM process.

Similar to active reconstruction techniques, the scene of interest plays a key role in reconstruction. Due to the fact that passive reconstruction solely relies on matching points of interest within the scene, it is then imperative that the scene contains enough identifiable points of interest. This translates to well textured surfaces, favourable lighting conditions, minimising reflections etc. While there is the potential for less planning than LiDAR when performing a capture session, there still exists best practices to ensure you achieve the absolute best capture quality given the scene capture scenario.

Using mobile phone cameras for reconstruction has become an area of research interest. Phones are shipping with an increasing number of sensors, which can improve reconstruction accuracy. One such example is geolocation data. This type of tagging on images has been successfully used to improve reconstruction quality and decrease the impact of phenomena such as drift [18].

With the high cost of many alternative reconstruction solutions and the accessible nature of image capture, there is an increasing interest surrounding passive image based

reconstruction. This accessibility has led to more inexperienced end-users attempting to perform reconstruction without any prior knowledge. This poses the question, how can we best inform end-users to get the best captures? Having analysed the available reconstruction techniques, I decided that for this research I will be focusing on the offline passive reconstruction methods, particularly Structure-From-Motion (SfM).

2.3 General Purpose Structure-from-Motion

2.3.1 Problem

Structure-from-Motion (SfM) is the process of reconstructing 3D structure from its projections into a series of images [18]. This reconstruction process involves first understanding the characteristics of the images and how they were captured. To do this we can analyse each of the three components making up the capture system. When an image is captured, this system takes points in 3D space and projects them onto a 2D planar image. With 3D reconstruction we wish to reverse this process using the information available to us from the image.

The first component of the capture system to consider is the camera calibration, which is composed of intrinsic and extrinsic parameters. For a perspective camera model, intrinsic parameters include focal length, principle point, lens distortion and skew factor. This defines the transformation and distortion through which points from the scene will undergo as they are projected onto the final image. The camera extrinsic parameters describes the location and pose of the camera within the 3D environment. Camera extrinsic parameters can be described by euclidean coordinates and a rotation vector or equally be described by translation and rotation matrices. Camera extrinsic parameters are crucial for registering and arranging images in 3D space so that we can recover the spatial information of the scene and the structures within it. The second component is the projected point in the image, this is where a point of interest in the scene is located on the planar 2D image that is captured. Finally, the third component is the location of the points of interest in 3D space.

This capture system can be described by the equation $p = C * P$. p is the projection of the point in the image. C is the calibration of the camera, both intrinsic and extrinsic. P is the 3D coordinates of the point. Understanding this capture system and using this equation in the case of reconstruction we want to recover P, the points in 3D space. Depending on the variation of the problem, we will have one or two elements that we are trying to solve simultaneously. In most cases, we cannot assume the camera intrinsic parameters as there is manufacturing variance in most lenses and potentially this metadata may not be available to use. In the best case, the camera intrinsic information supplied by image metadata can be

used as a starting point for later optimisation. We know that camera parameters are impacted by environmental factors, such as heat, thus this optimisation is necessary. With this in mind, the problem we wish to solve is finding P when both C and P are unknown, only using the projected image point p.

Homogeneous coordinates also referred to as projective space coordinates are used here to simplify the solving of this equation. When using homogeneous coordinates translation, rotation and scaling operations can all be performed via matrix multiplication. This enables us to simplify the solution of the p = C * P equation to matrix multiplication operations. More info on homogeneous coordinates can be found here [19] [20] [21].

When reversing the image capture system to perform reconstruction, there are two general problems we are trying to solve, estimating the camera parameters, both intrinsic and extrinsic, and the second problem is then reprojecting observed image features back into 3D space. In the next subsection, I will discuss some of the approaches taken to solve this complex problem.

2.3.2 Early Work

Photogrammetry

Photogrammetry was first used to describe an area of research exploring manual reconstruction of 3D environments. In the earliest work, it was found that by capturing overlapping images from a camera with known parameters and through the use of trigonometry, it is possible to determine accurate measurements of features without the need for manual measurement of each feature point [22]. This came as a particularly interesting proposition for cultural heritage and forestry applications where obtaining accurate measurements may not always be possible, but highly desirable [23] [24]. Today photogrammetry encompasses all image based 3D reconstruction techniques, such as those later mentioned in this paper. The greater accessibility of these techniques and an evolving list of application areas has grown a thriving online community, with projects of interest ranging from preservation of historical artifacts to reconstructing household objects [25] [26].

Linear and Factorisation Approaches

From these early works, the desire to improve upon techniques and automate the reconstruction process lead to the development of linear and factorisation approaches. Longuet-Higgins et al. [27] proposed a linear method based on point correspondences, *the eight point algorithm*. This approach estimates relative camera motion between pairs of

cameras through the use of point correspondences and epipolar geometry. In this approach, the coordinate system of one camera is fixed in place and used as an anchor, with the relative position of the second camera being estimated and then used to establish a global coordinate system within which 3D points can be triangulated. Using eight point correspondences between the captured images and by utilising epipolar constraints, the SfM problem can be solved and the 3D feature points can be located within the coordinate system. Later, C. Tomasi and T. Kanade [28] proposed a factorisation method which utilised orthographic assumptions about the object of interest and showed that by ensuring objects of interest are far away from the camera, orthographic projection assumptions can be made. Using this assumption they went on to show, depending on the amount of noise present, that we can obtain camera parameters and 3D world coordinates via factorisation and through the use of singular value decomposition. The findings from these early linear and factorisation approaches, while having limitations in terms of noise tolerance and the assumptions made, paved the way for future more general purpose reconstruction techniques.

Bundler

While SfM techniques were gathering interest in academia, a major breakthrough for more widespread interest in SfM was the paper titled *Photo Tourism* [29]. This paper expands on the approaches discussed above and surrounds them with complimentary computer vision and computer graphics components to make the Bundler reconstruction pipeline. In this work the authors target a large scale of reconstruction with 150k input image-set, containing a diverse set of camera calibrations gathered from internet image sharing sources. One of the key components they employ to achieve this target is non-linear optimisation of the reprojection error also known as bundle adjustment [30]. Bundler extends the camera calibration estimation that is possible with previous SfM techniques and makes it possible to produce a dense 3D point cloud. This goes on to form the first of it's kind in a general purpose SfM based reconstruction pipeline for an arbitrary image-set. This pipeline produces a dense point cloud and interactive system, which can be used as a user friendly medium to browse related photos, while also gaining an appreciation of the 3D structure. For the first time general purpose reconstruction was not only handled at greater scale but more importantly it opened the idea of a more accessible solution, as anyone could assemble an image-set. This could be used to explore image-sets in a new way, or indeed go on to create 3D models. At this time, there existed viewpoint interpolation and topology generation and exploration, but not a method that combined these with the spatial exploration of an image-set. Bundler combined this with a GUI that enabled the exploration of this reconstruction in a way not previously seen. These developments paved the way for SfM, its modern use and the implementations we have available to us today.

2.3.3 Reconstruction Pipelines

Today's reconstruction techniques take the approach developed by Bundler, refining it and improving upon it. By substituting and adjusting components of the reconstruction pipeline and using them together with additional computer vision and computer graphics components, updated pipelines strive to produce a general purpose solution suitable for the vast majority of applications. While we have discussed the SfM process thus far, using SfM alone will only produce a set of camera parameters and a sparse point cloud representation of the recovered 3D points. However, in most cases this sparse point cloud is not sufficient to generate an accurate 3D model and as such other supporting techniques must be used to produce a 3D model end to end. Commonly Multi-View Stereovision (MVS) is used to create a dense 3D point cloud of the scene of interest, using the camera parameters estimated during SfM. This dense point cloud can later be processed and meshed for use in 3D graphics applications. Both SfM and MVS are non-trivial problems to solve and by nature are computationally intensive. In this way, the major challenges for this type of reconstruction has been balancing computational efficiency and generality. As such these reconstruction pipelines have been developed to balance these factors out of the box. This often comes in the form of a recommended configuration provided by the authors, which they have found to have the most balanced results.

Optimisation attempts have been made to improve this issue, for example Bundler reduced the run-time of the reconstruction pipeline by scaling the problem to run efficiently across a cluster of machines with a novel distributed matching and reconstruction algorithm. Since then several other approaches have been taken to optimise the reconstruction process such as the use of different formulations of the SfM process including incremental, hierarchical and global. State of the art open-source reconstruction pipeline COLMAP performs optimisation through the use of CUDA GPU acceleration, particularly for the later stages of its reconstruction pipeline [18] [31].

Beyond these optimisations, machine learning and deep learning has been used to improve elements or indeed the entirety of these traditional techniques, with approaches such as NeRF [32], volumetric video [12] showing promising results in these areas. These deep learning based approaches abstract much of the reasoning we have behind certain reconstruction outcomes and in most cases also require a significant amount of training data to perform well. Many of these approaches are closer to academic research than commercial use, requiring manual setup and programming knowledge to configure. As such they are less accessible to users. In this way it was decided to focus on the traditional approaches for this research as they are more widely used in industry and by hobbyists. I hoped that results from this research can be extended to these machine learning approaches and be researched

further in the future. For these reasons I will be focusing on the traditional reconstruction techniques in this research.

There are many reconstruction softwares available today. Examples of commercial software solutions include: ArcGIS by ESRI, ContextCapture by Bentley Systems, Correlator3d by Simactive, Inpho by Trimble, iWitnessPRO by Photometrix, Metashape by Agisoft, Pix4dMapper by Pix4D, PF-Track by ThePixelFarm, RealityCapture by Epic Games, ReCap by Autodesk, Zephyr by 3DFlow. Being commercial and closed source gives us limited insight into the inner workings of these reconstruction pipelines and their component parts. However, there are many open source solutions including: COLMAP, AliceVision Meshroom, MicMac, MultiViewEnvironment (MVE), VisualSFM, OpenMVG, Theia, OpenDroneMap, Regard3D (based on openmvg and openmvs). Each of these balance functionality with user friendly GUI. Some such as OpenMVG strive to be understandable and maintainable over everything else, AliceVision Meshroom tries to bridge the gap between academia and industry through its use of a GUI. However, all these pipelines can be described as traditional reconstruction pipelines, as they use fundamental and well understood computer vision techniques to achieve their results, with one of the core steps being SfM. In the next section, I will discuss the steps in a traditional SfM based reconstruction pipeline and discuss how some of these open source solutions differ in their implementation to accommodate different application needs.

2.4 Components of a Reconstruction Pipeline

A reconstruction pipeline can be defined as a combination of many computer vision and graphics techniques, which ultimately will produce a dense 3D point cloud or 3D mesh. Broadly speaking, most pipelines break this reconstruction process into five main stages:

1. Feature detection
2. Feature matching
3. Structure-from-Motion
4. Multi-View Stereovision
5. Meshing

In the following subsections, I will discuss each of these steps in more detail using open-source solutions as examples, mainly Bundler, COLMAP and AliceVision Meshroom, henceforth referred to as Meshroom. Bundler represents the first attempt at an open-source general purpose reconstruction pipeline and both Meshroom and COLMAP represent the

state of the art in SfM based reconstruction.

2.4.1 Feature Detection

The first step of the reconstruction process involves gathering as much information about the scene as possible from the supplied images. From this we hope to obtain information about common points of interest, the camera calibration, 3D geometry of the object or scene and ultimately a sparse representation of this 3D geometry. Feature detectors are used to extract this information from the image. The function of a feature detector is to detect distinct keypoints, also referred to as feature points within an image. A feature point is a salient point of interest which contains useful geometric information within the image [33]. This is a foundational step for many computer vision operations, with there being an extensive body of research in this area. Many feature detectors have been developed over the years ranging from handcrafted detectors for corners and edges such as Harris [34] to machine learned detectors such as FAST[35]

The accuracy of the SfM techniques used in these pipelines is based on the number of keypoint correspondences which can be correctly identified between images. Especially in techniques which are simultaneously calculating camera parameters and refining keypoint detection [36]. One popular detector is Scale Invariant Feature Transform (SIFT) [37], SIFT is robust and resilient to changes in scale and rotation. This makes it an ideal choice for matching feature points across images with varying viewpoints, hence its widespread use in many SfM reconstruction pipelines.

Increasingly, we have seen a move away from hand-crafted feature detectors and towards machine-learned ones, as they can provide more robustness to things like changes in lighting conditions. To improve results, generally a number of feature detectors can be used in tandem to find different feature points across images For example, Meshroom includes a number of feature detectors including SIFT, AKAZE and CCTAG. This allows for customisation and fine tuning by the user to use one or multiple feature detectors to suit their application needs. The reader is directed to these sources for a more in depth review of feature detectors [33] [38].

In addition to the computer vision feature extraction, it is also possible to pull metadata from the image files themselves. Most modern cameras will output useful data such as its model number, focal length, resolution etc to the image EXIF metadata. This information can be useful for establishing initial estimates of camera intrinsics. Both Alicevision and COLMAP will use any available metadata to establish an initial camera model, with EXIF information only being used as an initial estimate as it can sometimes be inaccurate.

2.4.2 Feature Matching

In the matching stage, feature descriptors that have been identified in the previous stage are taken one by one and matched across images. Each feature descriptor should match exactly one feature descriptor in the other image, if this does not occur it is not a match and is discarded. Multiple matches for a single feature point commonly occurs with repetitive structures, such as the face of buildings, in this step those ill-matched feature points get discarded.

Feature matching can be a time-consuming process depending on the approach adopted. While approaches like exhaustive image matching can give the absolute best matches for the image-set, it scales quadratically with the number of images supplied as every images is compared to every other image in the dataset. Vocabulary trees can be used to reduce this run-time by quantising the feature descriptors, as discussed in this paper [39]. Vocabulary trees and are commonly used when dealing with larger image-sets and have been show to do so with great effect. However, for speedup to be gained the tree must be relatively balanced, otherwise the gains disappear over exhaustive approaches. COLMAP, Meshroom and Bundler all provide both exhaustive and vocabulary tree feature matching approaches.

2.4.3 Structure-from-Motion

Having detected and matched features, we can now begin estimating the relative positions of cameras to each other using SfM. At this stage, we hope to estimate both the intrinsic and extrinsic camera parameters for each of the images. Supplying an image-set which captures a diverse set of overlapping views from within the scene will have a notable effect on the final quality when compared to image-sets with a narrower focus. If the displacement between cameras is too small, triangulation of points becomes difficult and ill-conditioned as there is not enough disparity between the images to perceive depth. Well-distributed cameras are more likely to provide sufficient coverage of a scene and thus give us more keypoints, which can be used to triangulate 3D scene information [36].

After feature points have been matched between two images additional filtering occurs to ensure that they are geometrically consistent. Two-view epipolar and planar homography constraints can be used to help perform verification and filter outliers. Random sample consensus (RANSAC) is commonly used here to further filter outliers and improve the quality of matches retained. This geometric verification results in a scene graph, relating image matches to each other and sets up everything for the SfM reconstruction process. SfM can be split into 3 different types of technique, incremental SfM, hierarchical SfM and global SfM. Each of these techniques differ in the way they register new views and apply bundle adjustment and as such have differing characteristics.

Incremental Structure From Motion

This type of SfM starts off with an initial pair of seed images as the base for reconstruction. These seed images can be selected based on the number of keypoint matches they contain, randomly or manually. This step greatly impacts the quality of the reconstruction and could result in non convergence or failure to register future images if poorly selected. Reprojection and optimisation is a non-convex problem and as such it is possible to converge on local minima rather than a the global best solution. This is what makes image pair initialisation so key to the final results. Taking the image pair, point correspondences are reprojected into 3D space and then optimised using bundle adjustment, with the goal being to reduce the reprojection error.

After the first image pair has been optimised, the next closest image in the scene graph in terms of keypoint matches is registered to the pair and the camera parameters updated accordingly and validated using bundle adjustment. This processes is repeated for each image to be registered. COLMAP, Alicevision and Bundler all support incremental SfM. While sensitive to initialisation, incremental SfM is useful for multi stage reconstructions. In the use case of field capture, you can add more views to the scene without requiring the entire reconstruction to be recomputed. This allows users to return to a capture environment after the first reconstruction is completed if they have areas that require better coverage. Equally with the use of large online photo collections via social media and other sources, one could potentially enhance a reconstruction as more images become available.

Hierarchical Structure From Motion

Hierarchical SfM differs from incremental SfM in the matching stage where it partitions the problem in to a hierarchical tree structure. It does this by matching and organising the images with the highest number of correspondences together. This has the potential to decrease run-time when compared to incremental in cases where a balanced tree can be created. In cases where this is not possible, the performance advantage falls away. Once hierarchical partitioning is complete during feature matching the SfM process and view registration is then guided by the hierarchical structure.

Each cluster gets reconstructed in a similar fashion to incremental SfM, we then merge clusters together. Bundle adjustment is then performed after the merging of two clusters to reduce the reprojection error. This means we are performing bundle adjustment on smaller clusters which runs faster. This partitioning into separate subproblems gives hierarchical SfM desirable parallelizable properties, which can help with computation efficiency. However, it is worth noting that merging the submodels can introduce artifacts and require several rounds of bundle adjustment to filter them out.

Bundler draws ideas from hierarchical SfM and first reconstructs a skeletal set with the most similarity and then adds to this with the less densely covered areas of the scene. This means we can exploit dataset sparsity, which is a characteristic of unordered internet photo collection. However, something that has more evenly distributed coverage of a scene will not have such sparsity to exploit and thus we will not get the same sort of speed up improvements.

Global Structure From Motion

Global SfM calculates an initial solution by considering all the images in the image-set at once, then using bundle adjustment the accuracy of this estimate is improved. This type of SfM is much more susceptible to noise because all of the images are considered at once, meaning there is less opportunity to filter noise. Thus, noisy images will have a greater impact on the end result. Global SfM benefits computationally by only having one bundle adjustment, which is usually the most computationally intensive part of SfM. This improves over incremental SfM where we have bundle adjustment after each new image registration. Theia is an example of a global reconstruction pipeline [40].

Bundle Adjustment

After individual keypoints have been triangulated into 3D space, geometric verification is performed to ensure these reprojected points are consistent with all registered cameras and their intrinsic and extrinsic parameters. Due to inaccuracies in correspondence estimation, this is not perfect and we get a re-projection error. The error observed in this verification process is known as reprojection error. It is desirable to minimise the reprojection error to achieve the most accurate set of camera parameters for each camera. This joint minimisation of the reprojection error is known as bundle adjustment. This is a non-linear minimisation process in high-dimensional space as we are refining both camera intrinsic and extrinsic parameters and 3D point positions simultaneously. This reprojection error can be thought of as a cost function, which bundle adjustment minimises and in so doing, improves the accuracy of the reprojected 3D points and the camera parameter estimates[41].

The largest time component of the SfM stage is bundle adjustment, which scales with the number of images. Thus, having several bundle adjustments has a detrimental effect to run-time. As can be seen there are many factors that impact the selection of SfM formation whether it is dataset size, noise, sparsity or incomplete with views being added.

2.4.4 Multi-View Stereovision

MVS algorithms have the goal of producing a geometrically consistent dense representation of a 3D scene. This process is often closely coupled with surface meshing to produce a final 3D model. MVS approaches take the camera parameters estimated during SfM and use them to first produce a depth map for every image in the supplied image-set. To do this, images are undistorted by using the camera intrinsic parameters thus determining what factors influenced the perception of the scene as it was being projected onto the 2D planar image. Initial depth values are then taken from the sparse reconstruction produced during SfM and then refined and optimised. This refinement is conducted in "patches", where portions of each image are projected into adjacent images in search of consistency amongst the various views and their depth maps. These per image depth maps are used to perform surface reconstruction.

Once the per image depth maps have been finalised, the input images are sampled and combined to make a visually consistent texture for use when colouring the 3D points, which make up the reconstruction surface. This is a non trivial problem due to illumination changes and visual discrepancies from image to image. It also should be noted that the resolution and detail of these textures will depend on the number of images captured and their resolution. The surface reconstruction and texturing processes are computationally intense and account for a large chunk of the time required for the overall reconstruction process to complete. Finally, the textured surface points are then fused together to form one globally consistent dense 3D point cloud. For more information on the MVS process and the details of its use in COLMAP, the reader is directed towards these papers [31] [42].

2.4.5 Meshing

Meshing is the final stage of the reconstruction process and aims to produce a 3D mesh from the dense point cloud. During meshing, dense 3D points are joined to create faces and ultimately a 3D surface. Not all reconstruction pipelines provide meshing functionality, such as Bundler. However, once a dense point cloud has been produced, meshing can be completed in other utilities, such as MeshLab [43]. In this way meshing does not necessarily make up the core reconstruction pipeline, but does serve to produce a final reconstruction in the form of a 3D mesh, which is desirable for most 3D applications. COLMAP does provide meshing functionality in the form of poisson and delaunay meshing. Alicevision also provides meshing functionality and utilises a delaunay meshing approach. The reader is directed towards both the COLMAP and Alicevision papers for more detail on their use of these meshing techniques [18] [44].

2.5 Quality Impacting Factors

Having established a clear understanding of SfM based reconstruction techniques, we now turn our focus towards the factors which impact final reconstruction quality. These factors have been identified as common threads throughout research in this area, with it possible to link each impacting factor to a particular phase in the reconstruction pipeline. Often these factors cause a cascading effect where errors become more pronounced and influential as the reconstruction process progresses. While many of these impacting factors are environmental and potentially out of the control for a user capturing a scene, many others can be attributed to the camera setup or capture process. These controllable factors are what pose the most interest, as better understanding them could guide users to better capture and reconstruct arbitrary scenes.

2.5.1 Image-set Properties

Scene capture is the main input data source to the entire reconstruction pipeline and directly impacts the quality of the final mesh, so it is crucial to capture high quality images. The quality of an individual image and by extension an image-set, can be attributed to several properties that they possess. The first and most obvious property is the resolution of the captured image. Capturing the highest number of pixels possible is desirable to ensure the maximum amount of detail about the scene is retained in the final image and not lost to pixel quantisation. Similarly, capturing the maximum amount of information about the scene is directly related to the number and distribution of images captured. Both of these factors have a bearing on the amount of visual overlap achievable between images. Too few images will result in not only poor reconstruction results, but also failure to reconstruct at all in some cases. A sparse image-set has the potential to challenge the reconstruction pipeline both during SfM and MVS. SfM will struggle to estimate camera parameters with a low number of correspondences and images. While MVS will struggle to reproject 3D points if there is little recognised detail between the images and also poor camera calibration estimations cascading from SfM.

Due to the nature of keypoint matching and reprojection, parallax or disparity between images is a desirable image-set property. Camera parallax and disparity can be achieved via a change in capture location rather than just rotation alone. This is essential to enable triangulating points detected back into 3D space. Both horizontal and vertical parallax are desirable, as it uncovers the maximum amount of depth cues available within the scene. However, it is important to note there lies a balance between view disparity and visual overlap. Too small a disparity and the captured views do not differ enough for us to infer depth or gauge any depth cues. But too far apart and we start sacrificing visual overlap

between images. An even distribution of cameras around the scene and at a high density is an ideal configuration to achieve this. While parallax and disparity between images is preferable, images which can be described by homography, where only the camera rotation differs still remain useful in filtering outliers and views during feature matching [18].

2.5.2 Environmental Factors

While optimising capture configuration can have a considerable positive impact on reconstruction quality, environmental factors remain to be where the most variance in capture quality is observed. Even the same scene but captured at different times of day can vary drastically based on factors such as lighting and specularities. Outside of a studio or indoor environment, lighting becomes a prevalent factor to contend with. Changes to the lighting and exposure during image acquisition can have a detrimental effect in both SfM and MVS. In SfM, feature matching can struggle or even fail as keypoints may have a differing appearance under changed lighting conditions. However, the degree to which this impacts quality depends on the type of detector or detectors used.

During MVS, changed lighting conditions can impact the per vertex colouring of the final mesh, as the colour values can differ from shot to shot. While not necessarily impacting the reconstructed geometry, it can result in visually inconsistent texturing. Intense lighting can cast both soft and hard shadows, which should be avoided if possible. These shadows can introduce phantom geometry to the reconstruction and equally prevent feature detection in those areas as they tend to be under exposed in comparison to the rest of the scene. Items which vary drastically depending on the lighting are those which are highly reflective or transparent. These objects have the potential to reflect something different from the environment depending on the viewpoint from which they are captured, adding noise to the reconstruction. An overcast day is usually preferable when performing image acquisition for these reasons, as generally lighting is almost equal in all directions, similar to that of ambient lighting in a synthetic scene, giving good visibility of the objects within.

Texture plays an equal role to lighting as it can have some of the same side effects in terms of keypoint detection and matching. Weakly textured objects and surfaces such as those with just a solid homogeneous texture, will result in very few keypoints being detected for matching and tracking during SfM. In this way, we learn very little about the geometry of that object or surface. This type of texturing can also increase the number of outliers introduced due to ill-identified keypoints, similar to regions under shadows cast by lighting. RANSAC can be used to reduce these outliers by sampling a random set of keypoints from the image-set and constructing a probabilistic model, which can be then used with other observations to predict and remove potential outliers. It is effective in filtering out these

outliers. Due to the patch matching process used during MVS, large homogeneous regions will result in matching failures due to the lack of details. This will lead to large empty regions in the final reconstruction. Placing highly textured objects within the scene can help with this. We know this from their use in calibrating capture studios.

Some scenes are inherently more difficult to capture than others due to their characteristics. Drift is a problem experienced by hierarchical and incremental SfM techniques when tracking scenes with closed loops, such as capturing all sides of buildings. As discussed, SfM methods operate by matching local features across images to track camera motion. However, this can be difficult in scenes where there are many objects similar in appearance with repeating patterns such as the face of buildings or geometric symmetry, for example. These factors can result in major errors in structure recovery, with completely different images being matched together incorrectly even when no correspondences are present [36]. The accumulation of such false matches can contribute to the drift phenomenon in both incremental and hierarchical SfM. Global SfM is less susceptible to drift as the accumulation of reprojection errors does not happen as it does in the others because all views are considered at once. Image geo-tags and additional sensor information can be used to help reduce the likelihood of drift happening, as discussed in these sources [45] [18] [46], but it remains a problem in large scale reconstruction scenarios.

2.6 Evaluating Reconstruction Quality

Evaluating reconstruction quality is a non-trivial problem to solve. Techniques for evaluating quality vary significantly depending on the final application, as there is no standard method which fits all targets. For example, when evaluating a new SfM technique, the number of features detected, matches made and run-time are all interesting metrics to consider and compare to other methods. Whereas when evaluating the end quality of a mesh, it may be appropriate to use a quantitative measure or even a subjective assessment. Two commonly used assessment measures can be categorised as relative and distance measures.

2.6.1 Relative Measures

Many of the measures in existing research rely on the use of a common dataset such as the Rome image-set used to evaluate Bundler [39] and COLMAP [18]. Using a well known dataset as a common benchmark, reconstruction techniques can be compared to each other at each phase of the reconstruction process. The number of features detected, number of images matched, 3D points successfully reprojected, reprojection error and run-time commonly fall under scrutiny [18]. In some cases where a specific improvement to the reconstruction pipeline is made, that aspect of the pipeline is then compared like for like to

other methods. For example, viewing the final reconstruction top down and analysing its shape has been previously used to indicate improvements to drift [46].

Such relative measures have brought to light many characteristics of large scale image reconstructions not limited to a specific reconstruction pipeline. These metrics are used time and again throughout the literature with Meshroom choosing the number, faces and vertices in the final mesh and run-time as their primary evaluation measures. The authors of Bundler found that when reconstructing large scale image-sets, the number of images, the complexity of the scene and the distribution of the images have a considerable impact on performance, particularly run-time [39]. This was shown when they reconstructed two cities, Rome and Dubrovnik, with different scene characteristics. In the Rome image-set, the images were mostly focused around popular landmarks featuring simple geometry with good visibility and image coverage. In the Dubrovnik image-set, the images were more evenly distributed and captured intricate side streets in more detail, with difficult lighting conditions. This increased the difficulty of the reconstruction process even though the number of images in the image-set was lower than the other reconstructions they tested.

While factors such as run-time and the number of keypoints reprojected are interesting to benchmark when releasing a new reconstruction pipeline, they do not wholly speak for the final reconstruction quality. In the examples mentioned, ground truth information is not necessarily available to compare against. Hence, relative performance serves the purpose of ranking reconstruction pipelines given the same data. However, when ground truth data is available, it becomes desirable to measure differences between it and the reconstruction.

2.6.2 Distance Measures

While relative measures have proven useful in gauging performance across a wide set of techniques, they do not indicate the amount of noise present nor accuracy of a reconstruction. For example, we could develop a new reconstruction technique, which reprojects significantly more keypoints than previous work, but there is no guarantee these are useful keypoints and not noise. When ground truth data is available for a scene of interest, both performance and final reconstruction quality can be measured. In many cases, techniques such as LiDAR are used to collect an accurate ground truth to which a reconstruction can be compared [47]. Having an available ground truth means attributes such as noise can be viewed in more detail and other mesh properties such as volume can be analysed.

One use of available ground truth mesh data, is to use its vertices and surfaces as

comparison points for reconstructed meshes. This approach enables us to discover what areas of a final mesh have been reconstructed accurately and others which have not. It also acts as a way to determine what points have been spuriously introduced throughout the reconstruction process. One measure that can be used for this assessment is the Hausdorff distance measure [48]. With this measure we take every point in the reconstructed model and then sample the ground truth model to find the nearest point. The distance between these two points then gives us a measure of accuracy, with a lower distance indicating higher accuracy. Using this kind of analysis, it becomes possible to represent the overall accuracy as a singular value, the Root Mean Squared (RMS). It also becomes possible to generate a heatmap over the surface of the mesh, highlighting areas above and below the ground truth surface, which were reconstructed inaccurately [12].

While these measures can give a more accurate view of final mesh quality, they are sensitive to outliers. The introduction of outliers to the final reconstruction will result in skewing to the final RMS value and the heatmap generated. As such, these measures are less suitable for large scale reconstruction of urban scenes where outliers are not uncommon. However, in smaller scale settings where a single object of interest is being reconstructed with the intention of meshing, these measures become particularly useful. Ultimately, the nature of the application and object of interest still play the deciding role in the quality assessment. For example, a reconstructed teapot which is 95% accurate, may be acceptable for use as a background asset in a 3D game, but the same level of accuracy when reconstructing a human for a VR application may be insufficient. The nuances of each application means no one measure or approach can be considered in isolation. These factors are what make generic quality assessment for an arbitrary scene such a complex problem to solve.

2.7 Outline my work

The list below summarises my research objectives with this dissertation:

1. Understand 2D to 3D reconstruction techniques.
2. Investigate and understand the state of the art in Structure-From-Motion.
3. Determine what are the prominent quality impacting factors.
4. Design experiments to evaluate the impacting factors for the selected technique.
5. Evaluate the obtained performance using pertinent metrics.
6. Form conclusions based on experimental results.

From this background research, I have established an understanding of 2D to 3D reconstruction techniques, the available approaches used in the field and have investigated

the state of the art in these techniques with particular focus on SfM. Finally, I discussed the quality impacting factors and identified what metrics are commonly used to measure this quality beyond subjective assessment.

Traditionally, these SfM techniques have been tested and developed with the reconstruction of large scale urban environments in mind, but how do they perform in a more generic environment?

3 Design

This section discusses the methodology followed to evaluate how the factors identified in section 2.5 impact final reconstruction quality. To achieve this objective, an automated experimental testing framework was developed. The goal for this experimental testing framework is to automate the end to end reconstruction process, so that it is possible to evaluate reconstruction quality from image acquisition to final reconstruction. Informed by background research it was chosen to split this process into the following steps: scene creation, scene capture, reconstruction and quality assessment. Figure 3.1 presents a high level illustration of this framework.

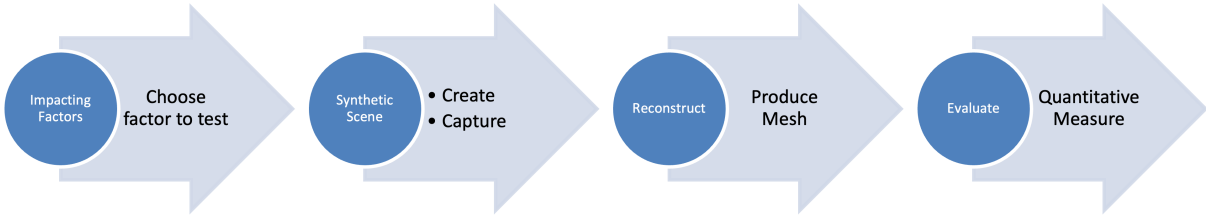

Figure 3.1: Experimental testing framework

3.1 Ethics Approval

Ethics approval was not required for the experiments conducted as part of this dissertation research as they did not involve human participation nor the collection, analysis and storage

of personal data.

3.2 Quality Impacting Factors

Based on findings during background research, a total of five potential quality impacting factors were selected for analysis:

1. Image resolution
2. Number of images
3. Distribution of images within the scene
4. Object of interest texture
5. Lighting conditions

These factors were selected to be representative of elements that the user can control about the capture, but also model environmental factors which the user has little control over. In this way, we can get a good representation of an arbitrary scene and determine how much a user can impact their reconstruction quality even in non-ideal situations.

3.3 Scene Creation

While it is possible to capture and perform reconstruction of a wide range of objects in varying environments, for this research it is desirable to construct a controlled environment in which the factors being investigated can be altered individually and in a measured way. To achieve this level of control, inspiration was drawn from how setup and calibration is performed in capture studio settings [49]. These capture studios are configured with cameras oriented towards a main focus area, usually a central point. These studios are designed to capture an object or objects of interest within the scene in the highest possible quality. This pattern of capturing with a central focus point closely models a standard reconstruction use-case where a user may be trying to capture an object of interest.

To achieve granular and isolated control over each of the identified impacting factors, the use of a synthetic scene is proposed. By using a synthetic scene we can control each known factor independently, even if that stretches what is achievable in reality. This will allow us to measure and observe how much a reconstruction is impacted by each factor, or combination of factors, with minimal external influences. Similar testing could be performed in a controlled studio environment, however there still remains several advantages in using a synthetic scene. Firstly, assembling a suitable capture environment or gaining access to a studio can be quite expensive and limits the reproducibility of the results, unless that exact setup is replicated. Secondly, even in a studio environment many factors can be difficult to

precisely control such as lighting or camera calibration, which may be impacted by external factors [50]. In contrast, a synthetic scene can be defined using code and reused for any number of capture sessions with zero variance to the setup. Thirdly, sourcing and producing objects with the exact material properties we wish to test has the potential to be a difficult and time consuming process. When using a synthetic object of interest, all geometry and material properties can be easily varied in a matter of minutes. Finally, the acquisition of precise ground truth data when using real-world objects is a difficult issue to solve. 3D printing offers a compelling middle ground, with the original digital model acting as a ground truth, but this limits the material properties that can be tested to the available printing filaments. For these reasons, it was decided to use a synthetic scene over a controlled real-world test environment. Using a synthetic capture environment, a model of interest can be placed within the scene in the focus region and then be captured accurately, with the model itself remaining available for easy comparison later.

3.4 Scene Capture

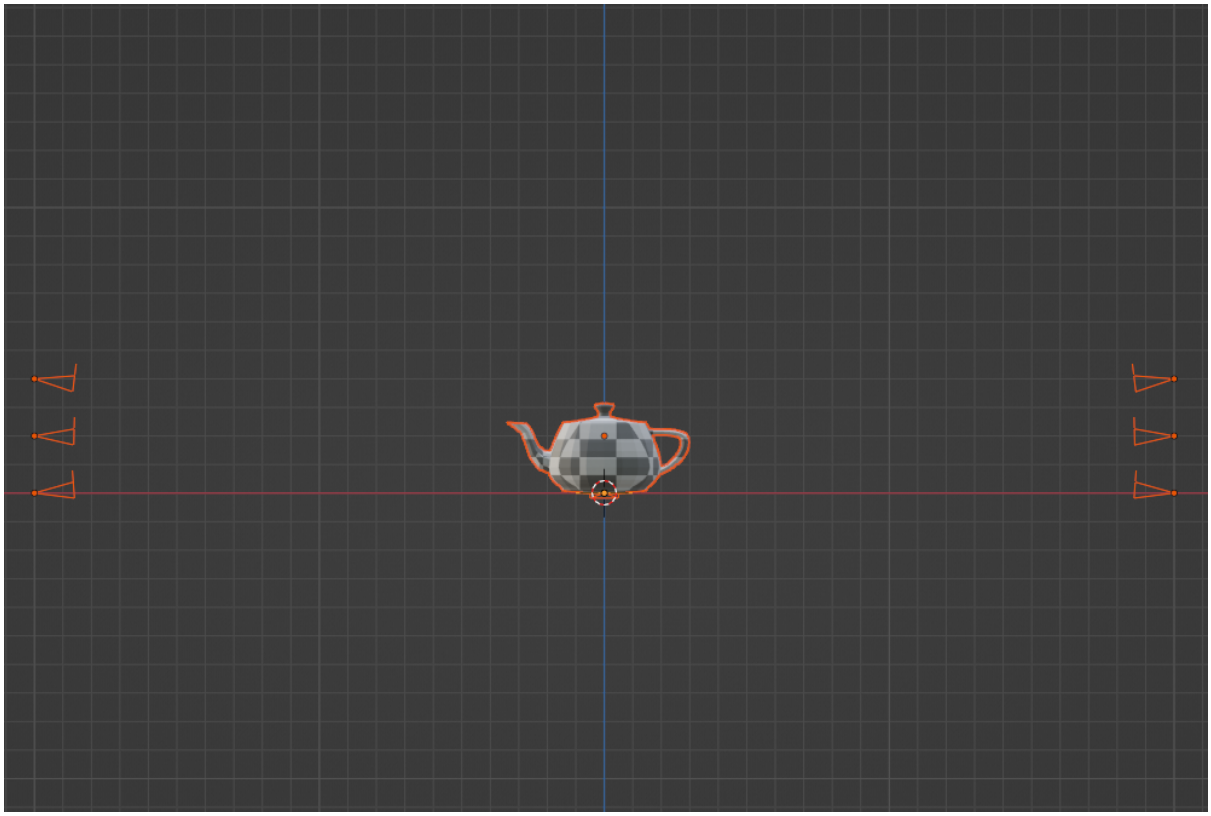

Figure 3.2: Side on view of various camera placement heights

As discovered during background research, both the distribution and number of images within the capture scene play key roles in the final reconstruction quality. These factors are tightly coupled as both influence the amount of overlap between images, a factor which we know to impact feature extraction and matching. Thus, having the ability to test various

configurations for both distribution and number of images is desirable. This can be achieved by placing virtual cameras around the synthetic scene in varying patterns and frequency. For these experiments, I decided to investigate two different camera placement patterns, circular and square. These patterns were selected to model what an end user could achieve when capturing out in the field. Following this model, it would be expected that a user would attempt to capture images from low and high vantage points for the best scene coverage. In the synthetic setup, cameras were placed in set circular or square orbits around the object of interest, representing ground level, eye level and above the head capture positions. Figures 3.2 and 3.3 show illustrations of these placement patterns with a sample model acting as the focus point. For the virtual cameras, a perspective camera model with minimal distortion

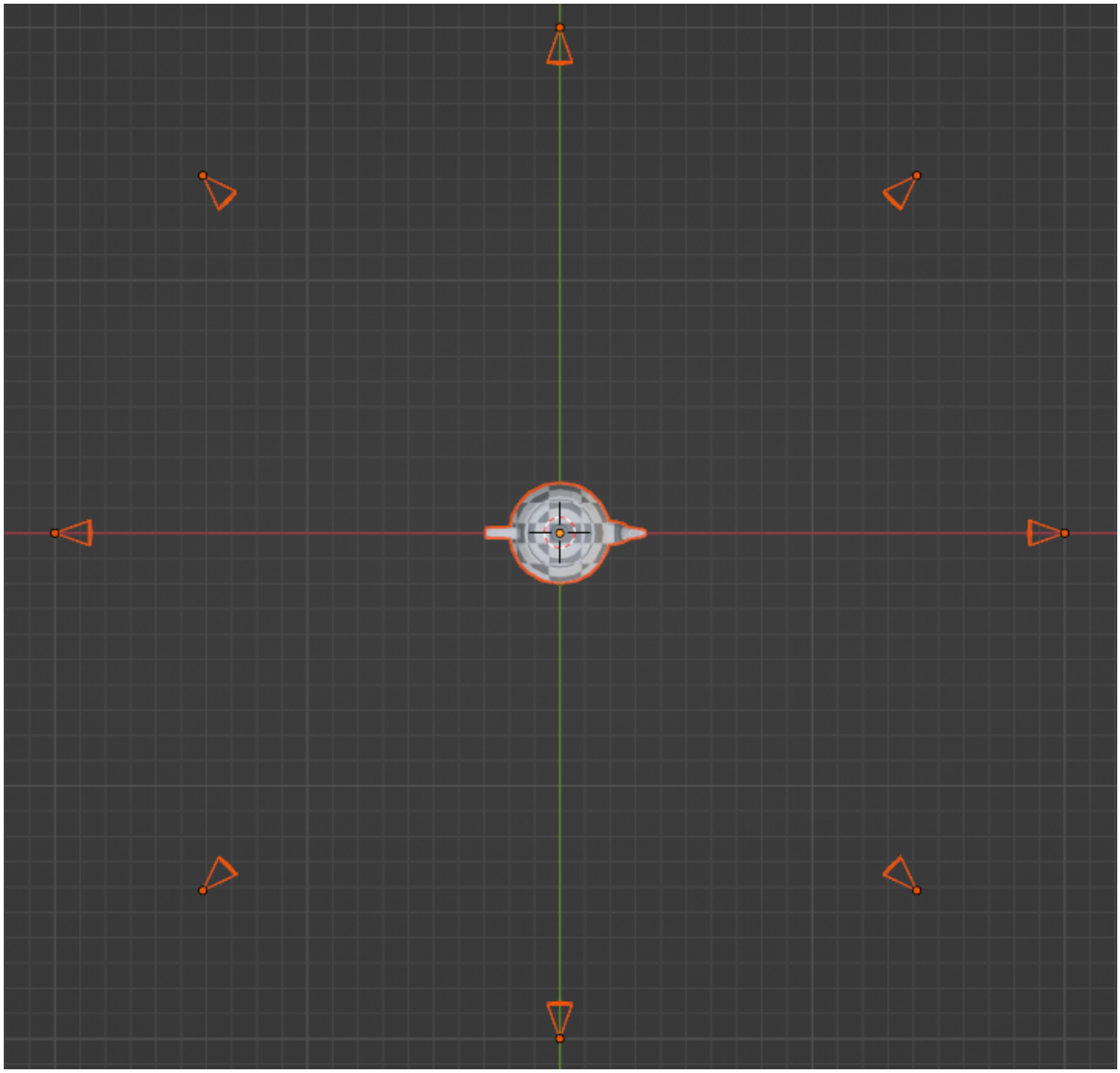

Figure 3.3: Top down view of circular camera placement pattern

was selected. This selection was made as it most closely models the setup present in modern smartphones. By using the synthetic scene, full control over the camera parameters is

available and as such it is possible to precisely track camera position and pose. Using this available control, cameras will be positioned with pose facing directly towards the object of interest. Having full access to all camera parameters makes it possible to evaluate the accuracy of camera placement estimates made by the reconstruction algorithm. Once in place, a virtual camera can be used to capture the scene visible from it's viewport at a chosen resolution. Figure 3.4 is an example of the final image captured using this configuration. The resultant image-set from this capture process will be then be used as input to the reconstruction phase.

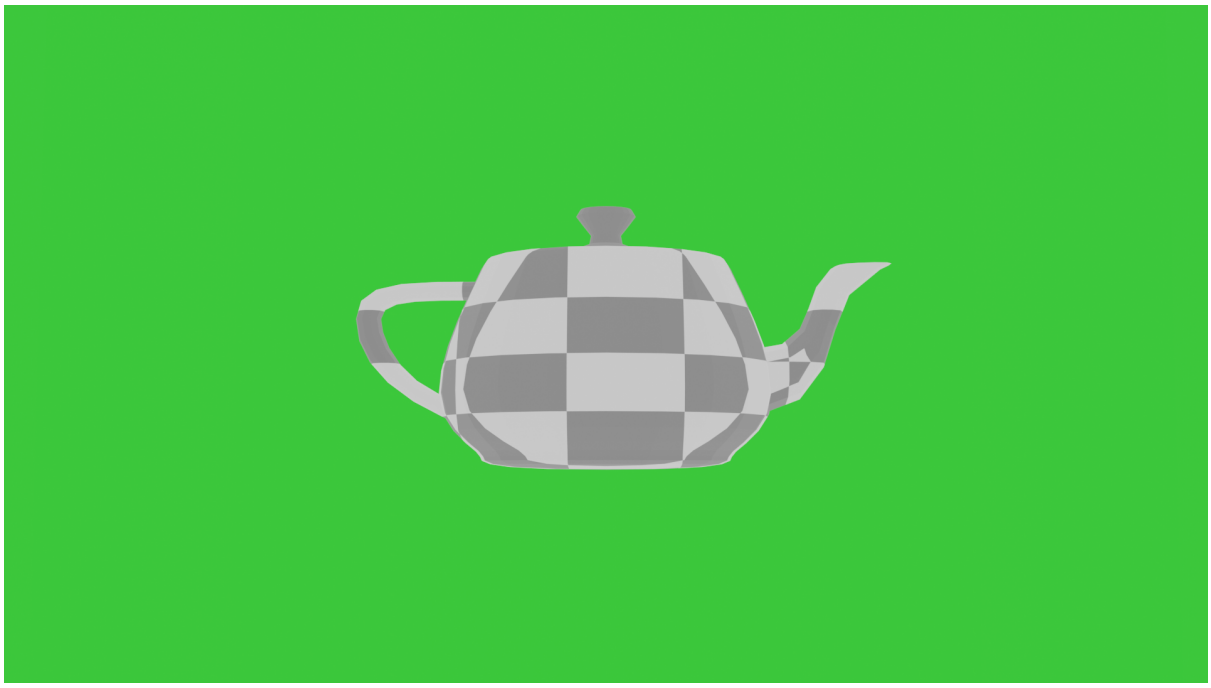

Figure 3.4: Sample image captured using a virtual camera

3.5 Reconstruction

With the image-set and ground truth data obtained, we now come to the reconstruction phase. At this stage, the selected reconstruction algorithm is used to produce a 3D mesh from the supplied image-set. Initially, it was thought that solely performing the SfM process could produce results in which we could analyse. By taking estimated camera positions and comparing them to the ground truth and measuring the error, it could indeed be possible to infer how well certain portions of the scene could be recognised in steps like feature extraction and feature matching. However, this still would not be representative of the final impact on quality for most use cases which is in the production of 3D mesh. As such the use of 3D reconstruction in its entirety was used. The recommended reconstruction configuration as suggested by the authors is used in this case as we want results to be representative of an inexperienced user using the the reconstruction software. The selected

reconstruction software and its configuration should then remain constant for all tests ran, ensuring a consistent ground on which to compare results. COLMAP was selected as the tool for reconstruction as it represents the state of the art in SfM reconstruction. In the future, different reconstruction techniques could be substituted into the testing framework and their performance measured.

3.6 Mesh Quality Evaluation

Having reviewed the literature, it became apparent that there is not one standard way to evaluate reconstruction quality. However, it was found that a number of quantitative measures do exist and could be used for assessment purposes. Using these measures, I propose the analysis of the reconstructions and the generation of quality metrics. Two measures were selected for this, SfM results statistics and Hausdorff distance based mesh quality assessment. The SfM metrics are obtained during the reconstruction process, at roughly the half way mark. The number of keypoints identified in images, the number of images registered and the number of 3D points triangulated are examples of these metrics.

The Hausdorff distance based assessment is performed using the final reconstructed mesh. This will show us how well the output model resembles the ground truth and what portions of the model were reconstructed well or likewise poorly. From these measures we can hypothesise what impacted the quality and how it can be remedied. This measures will allow us to not only to have a quantitative percentage representing how similar the meshes are, but it will also enable the visualisation of quality heatmaps. From these heatmaps it can be visually analysed to what extent mesh deformation has occurred and what areas of the mesh have been reconstructed to greater or lesser extent.

It must be recognised that no single quality measure is not the definitive answer for overall reconstruction quality, and that application specific concerns remain a top priority. However, in this case where we want to generalise how environmental factors impact quality, having a controlled set of assessment metrics means we can measure deviations and changes relative to other scene setups and thus, gain insight into the impact these changes have.

4 Implementation

This section first discusses the tools and libraries chosen to construct an experimental testing framework and the reasoning behind their selection. After this the implementation of an end-to-end reconstruction testing framework using these tools is discussed in detail.

4.1 Tools

4.1.1 Blender

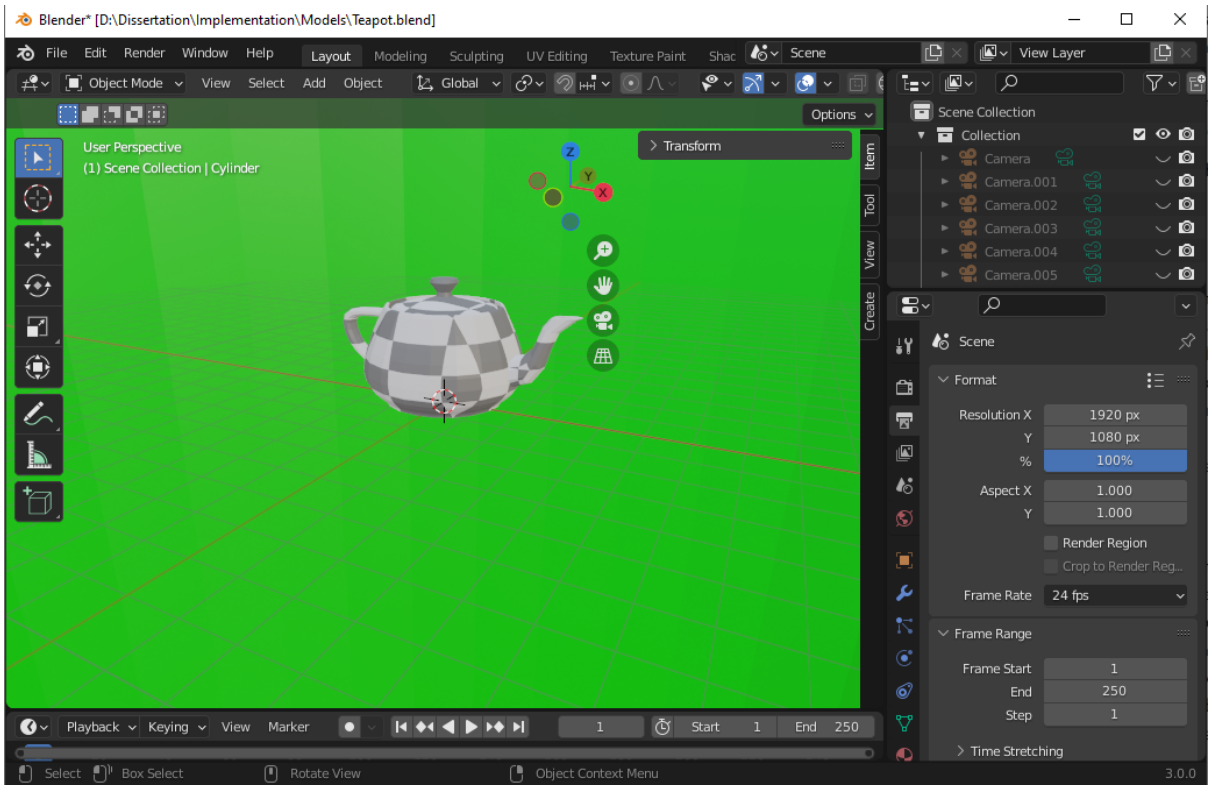

Figure 4.1: Blender modelling interface example

Blender is an open source 3D modelling suite which supports all of the scene setup requirements for this research. This includes model creation, image capture, photo realistic rendering and model exporting [51]. Blender enables the construction of a synthetic scene in

which it is possible to control all environmental factors, such as those identified and discussed in sections 2 and 3. These factors include image resolution, number of images, distribution of images within the scene, object of interest texture and lighting conditions. This enables us to create a synthetic scene which we can completely control and alter from run to run, targeting particular factors we wish to test in each experiment. Additionally, Blender has rich automation support, with every action available in the GUI being fully scriptable using python. For these reasons it was chosen as the scene creation and capture tool for this project.

Blender Photogrammetry Add-on

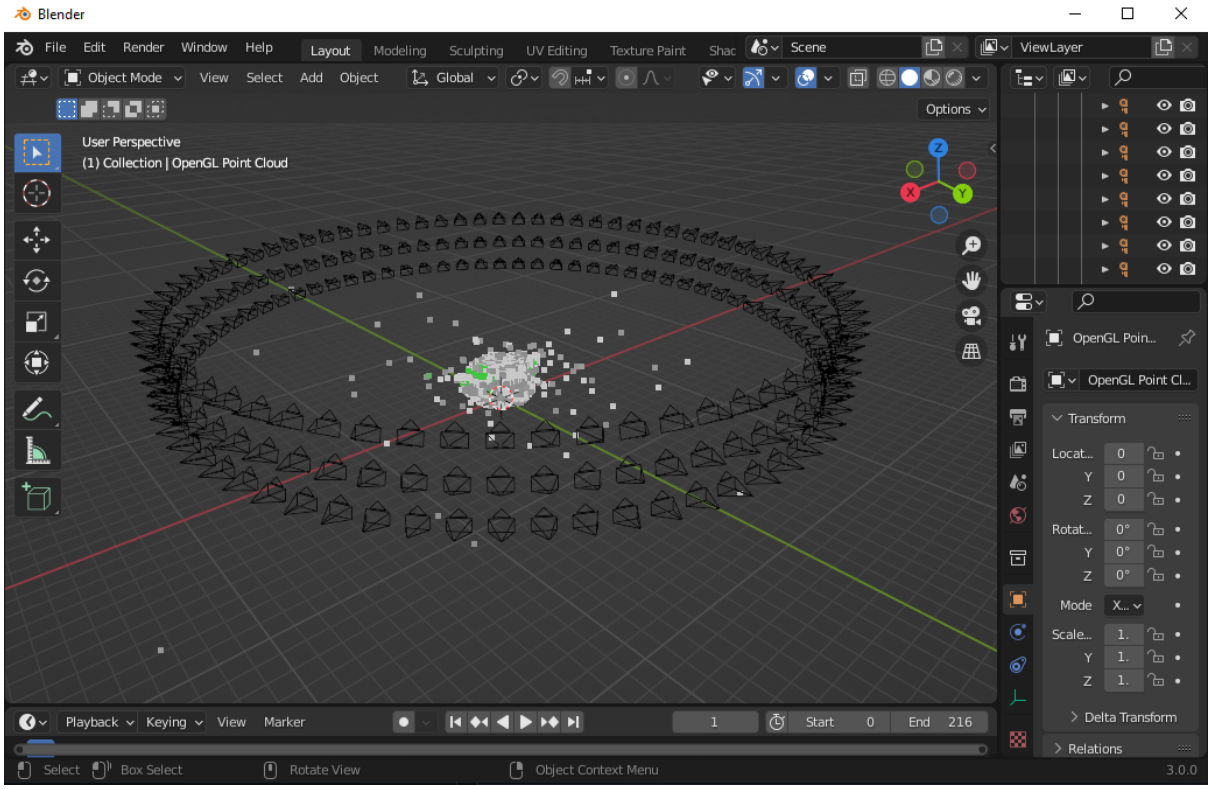

Figure 4.2: Blender photogrammetry add-on imported COLMAP workspace example

This open source Blender add-on allows many additional formats to be imported to and exported from Blender. For example, we can use this add-on to directly import the resultant point cloud produced by COLMAP into Blender. This opens up the opportunity to evaluate the model produced by COLMAP and visually inspect its quality. While the export features from COLMAP allow us to export a .ply model that can also be used in Blender, this add-on allows us to directly import the entire sparse point cloud that is produced by COLMAP, complete with camera poses and positions. In this way, with some adjustment, we can directly compare these to the original cameras and models in a visual way. This was helpful in the initial on-boarding to both Blender and COLMAP and was useful as an extra resource when evaluating results [52] [53].

4.1.2 COLMAP

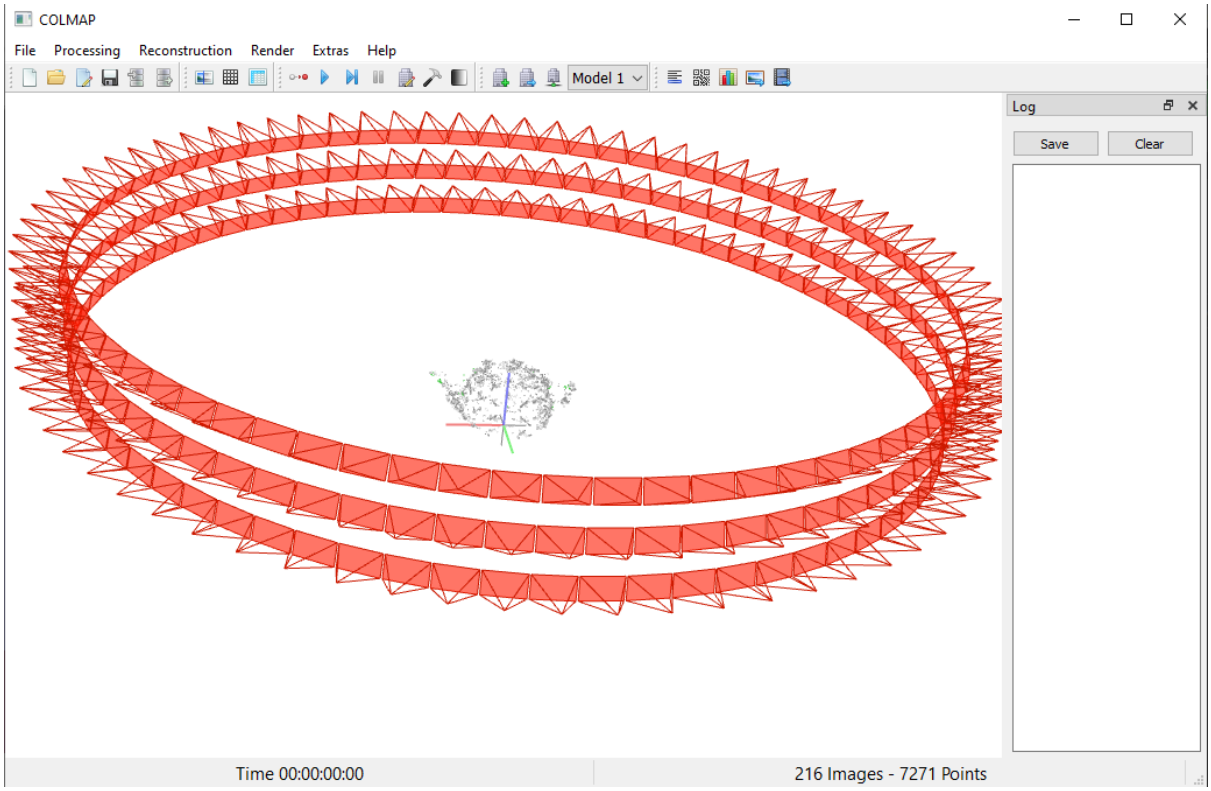

Figure 4.3: COLMAP UI example

As discussed in sections 2 and 3, COLMAP is a general-purpose Structure-from-Motion (SfM) and Multi-View Stereo (MVS) pipeline with a graphical and command-line interface [18] [31]. It represents the state of the art in SfM based reconstruction techniques and offers a complete end-to-end reconstruction pipeline. This enables us to take an arbitrary image-set as an input and produce a 3D model. Thanks in part to its GUI and preconfigured automatic reconstruction preset, COLMAP is an accessible reconstruction technique. Figure 4.3 shows a preview of the COLMAP GUI displaying an active reconstruction.

4.1.3 MeshLab

Meshlab is an open source utility for processing and editing 3D meshes [43]. It provides a rich set of granular mesh comparison functions and readily supports directly comparing meshes, using measures such as the Hausdorff distance measure [48]. It also supports outputting visual representations of comparison measures, with heatmaps being an example of such a visual representation. This not only allows us to quantitatively assess the reconstructed model against the ground truth, but also enables us to visually inspect all aspects of the reconstruction. As with all the tools selected, Meshlab has rich automation support via a python api named pymeshlab [54].

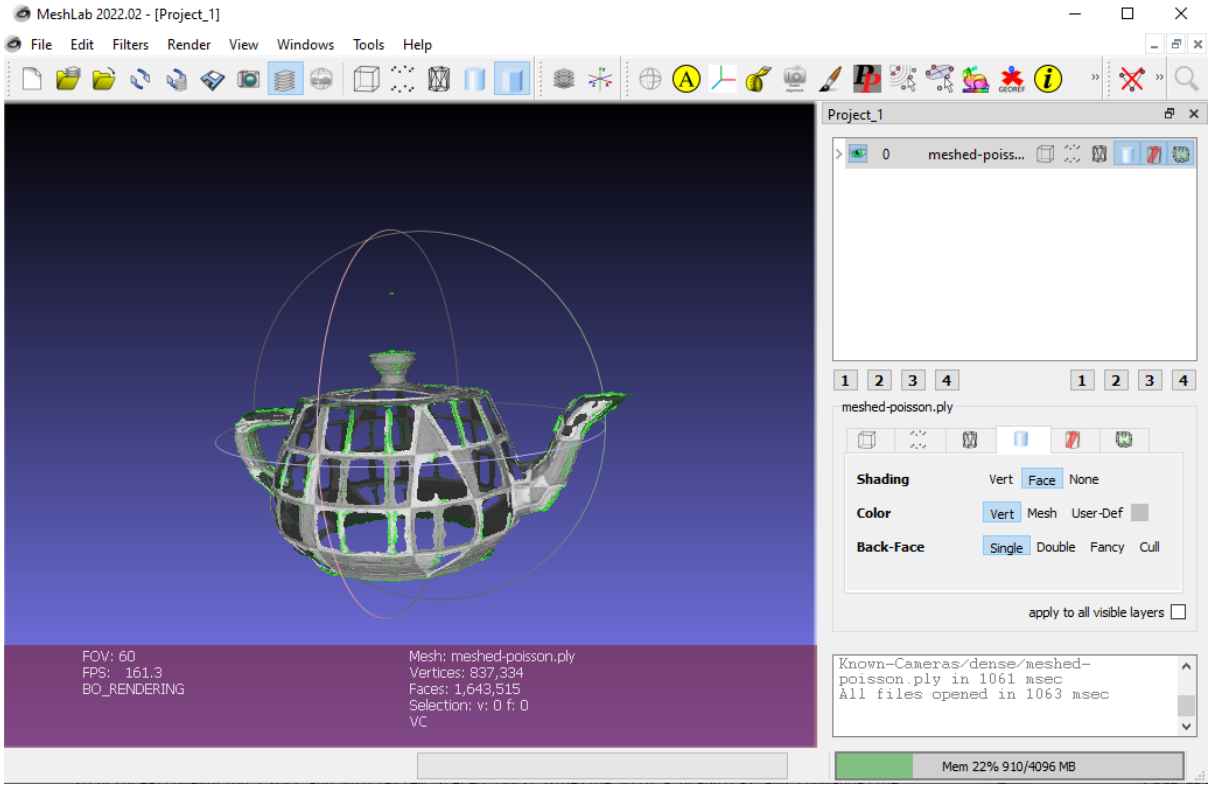

Figure 4.4: Meshlab UI example

4.2 Hardware

As discussed earlier in section 2, the reconstruction process is computationally intensive. Adding to this, computation requirement is the scene creation and capture phases designed in section 3. This amounted to many experiments taking multiple days to finish both scene capture and reconstruction. For this reason, I decided to run experiments across a selection of available devices to maximise the number of experiments that could be completed within time constraints. This resulted in the use of four different machines over the course of testing. Table 4.1 provides a list of the hardware specifications for each of these machines.

Name	CPU	RAM	GPU	OS
Machine 1	Intel Core i7 6950x	64GB	Nvidia Geforce GTX 970	Windows 10
Machine 2	Intel Xeon E3-1240v3	32GB	Nvidia Geforce GTX 980	Windows 10
Machine 3	Intel Core i5 7300HQ	8GB	Nvidia Geforce GTX 1050	Windows 11
Machine 4	Apple M1	16GB	Apple M1	MacOS Monterey

Table 4.1: Hardware used to perform reconstruction experiments.

Due to the varied hardware from run to run, it was no longer possible to take the

reconstruction time into consideration as part of testing. Should time and resources have permitted, the use of homogeneous hardware across test runs would have been preferred. Finally, to facilitate the sharing of data, each of these machines operated on the same network using a common network attached storage pool.

4.3 Experiment Pipeline Summary

Implementing the design discussed in section 3, gives us the multi-stage pipeline below:

1. Create a synthetic scene with model centred around the origin.
2. Vary the impacting factor of interest such as texture, lighting, geometry etc.
3. Place cameras in the scene following the selected pattern.
4. Capture images from the placed cameras and output them to disk, along with other ground truth metadata.
5. Perform a reconstruction with the outputted images, using COLMAP's automatic reconstruction option.
6. Analyse reconstruction statistics such as number of images registered and number 3D points reprojected.
7. Compare final meshed reconstruction to the ground truth model using Meshlab.

To orchestrate these steps in a automated way, a number of scripts were developed. This automation played a key role in ensuring the level of control and reproducibility desired for these experiments was reached. As mentioned throughout section 4.1, each chosen utility supports some form of automation. Blender has full python integration, COLMAP supports command line usage and Meshlab comes with a python api. These facilities made it possible for a fully automated testing environment to be created. This automation was achieved through the use of three primary scripts, a capture script, a Blender automation script and a reconstruction script. These scripts will be discussed in more detail in sections 4.5 through 4.8.

4.4 File Structure

To ensure interoperability between each of the pipeline phases, it was necessary to construct a common directory structure. I decided to extend the default COLMAP directory structure, which is generated when performing an automated reconstruction. When a scene is being captured, a top level directory is created and named using a unique identifier for that scene. For example, Teapot-Checkerboard-4k-Ambient-12-Circle indicates a scene centered around

```
Teapot-Checkerboard-4k-Ambient-12-Circle/
├── dense/
│   ├── images/
│   ├── sparse/
│   ├── stereo/
│   ├── fused.ply
│   └── meshed-poisson.ply
├── images/
│   ├── progcam0.png
│   ├── progcam1.png
│   └── progcam...
├── ground_truth/
│   ├── cameras.txt
│   ├── images.txt
│   └── points3D.txt
├── sparse/
│   ├── cameras.bin
│   ├── images.bin
│   └── points3D.bin
└── database.db
```

Figure 4.5: Workspace directory structure.

the teapot model, with a checkerboard texture, image resolution of 3840x2160px also known as 4k, ambient lighting only and captured from 12 cameras in a circular pattern. This top level directory is referred to as the workspace directory within the created scripts.

Below the workspace directory, there are directories to store the captured image-set, sparse reconstruction, dense reconstruction and finally the ground truth. COLMAP populates these directories with various data files throughout the reconstruction process, with the exception of the ground truth. The ground truth directory was added to store additional camera metadata outputted from Blender during the capture phase, to be used by COLMAP during reconstruction. Further discussion on the use of this ground truth data during reconstruction is given in section 5. Figure 4.5 shows an illustration of this directory structure, populated with files from both the capture and reconstruction processes.

4.5 Scene Creation

4.5.1 Model Selection

While the primary focus of scene creation lies with the environmental factors such as lighting, there remains interesting insights to be gained by selecting varying models for reconstruction. Initially, it was thought the use of simple models with distinct features would be sufficient for testing all environmental impacts. The teapot fits this criteria and

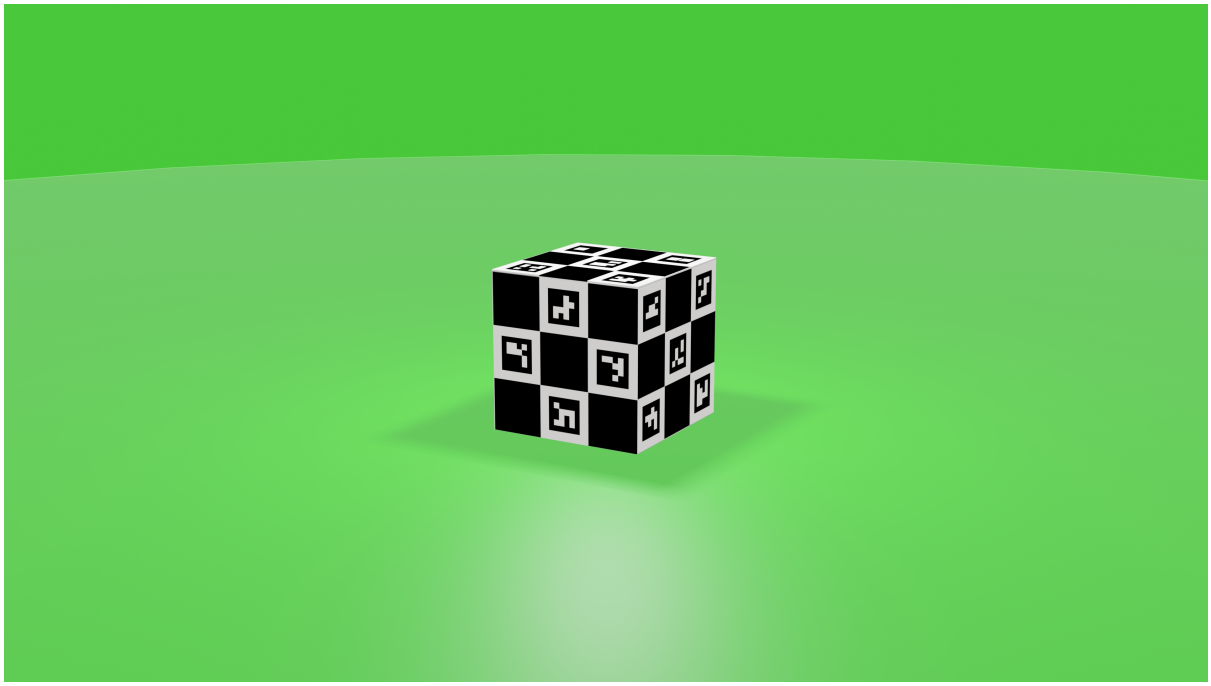

Figure 4.6: Cube model example

represents a relatively simple geometric object with intricate details and symmetry, all factors which make it challenging to reconstruct. This indeed was the case, with several reconstruction attempts outright failing after feature matching. For these reasons additional models were selected for reconstruction. The final models tested were a cube, teapot and wall like structure henceforth referred to as medieval wall [55]. Figures 4.6, 4.7 and 4.8 are images captured of these models.

4.5.2 Scene Setup

Lighting and Texture

In all cases, the model of interest is located at the world space origin of the Blender modelling environment. The model is then scaled so that it fits comfortably within the field of view of the cameras, which will be placed during the capture phase. This will ensure we get good image coverage for each capture session. After the object has been placed, an invisible point is added to the scene at the object's center of mass. This will act as a tracking point for cameras in the capture phase, as without it the cameras will point directly at the origin and thus, the object of interest will not be in the center of the frame. Beyond this, the remaining factors to vary at this point include lighting and texture. In terms of lighting, all scenes are lit using global lighting. This is achieved by setting the global scene lighting to the colour white, with a value of one. This emits an equal intensity of white light from all directions within the scene. Alternatively, a point light source can be added to the scene. This point light source will emit a similar white light, but from one direction within

Figure 4.7: Teapot model example

the scene and thus, cast shadows. The type of shadow cast from a point light source can be observed in figure 4.6.

For each scene, the model texture is varied using Blender's material properties control panel. This enables the use of image textures and also generated surface properties using Blender's built in material shader options. For the teapot model, the built in material shaders were used to produce checkerboard, gradient and solid colour surfaces on the teapot. The resultant model produced through use of these material shaders can be seen in figure 4.9. For the cube and medieval wall models, image texturing was used. These image textures were mapped to the models using Blender's UV editor. Each model was unwrapped during this process to allow for accurate texture placement. Figure 4.10 shows a unique, handcrafted calibration texture being applied to each face of the cube model.

Render Engine and Background Geometry

In the literature discussed in section 2, it is common for the reconstruction scene of interest to be a large scale urban landscape with buildings, plenty of background detail etc. The images captured from these scenes deal with a myriad of factors, such as transparency, reflections, different lighting conditions, time of day differences, weather etc. To mimic some of these factors, both the Blender render engine and the use of background geometry were proposed. For initial testing it was thought to create a controlled studio like capture environment, with ambient lighting and a homogeneous green screen background. This was tested as it was believed this kind of scene would be easier to reconstruct. However, these

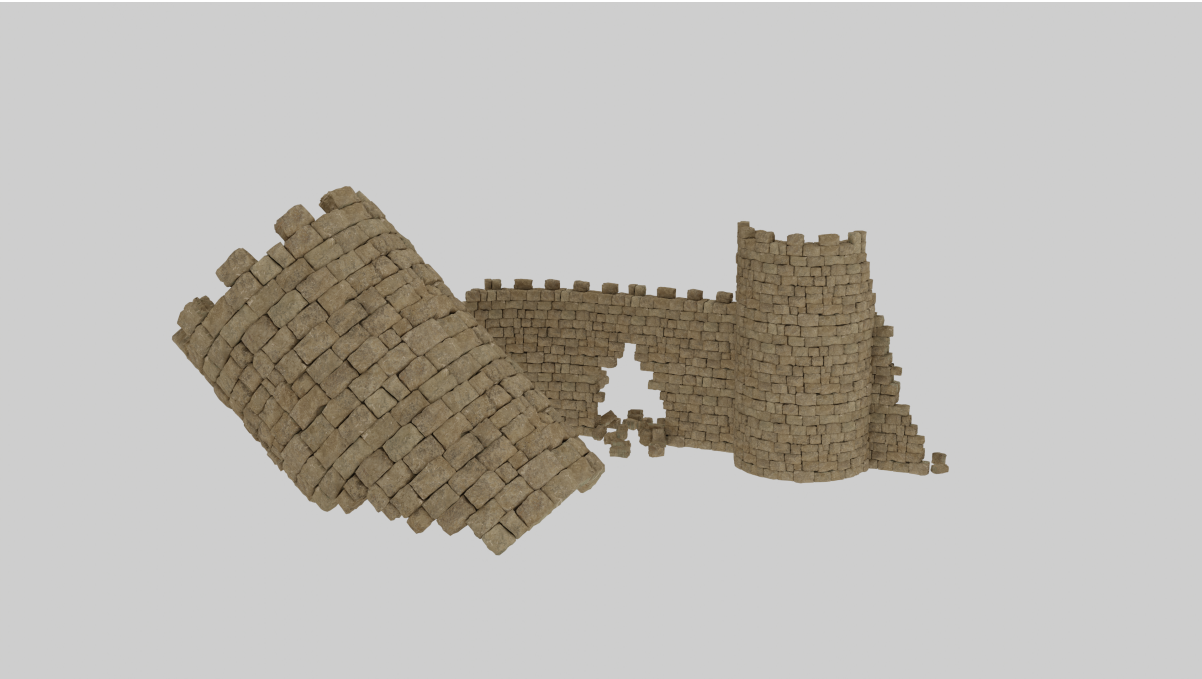

Figure 4.8: Medieval wall model example

Figure 4.9: Teapot material shader examples. Left to right checkerboard, gradient, solid colour.

initial scenes posed problematic when performing reconstruction and even when optimised still produced underwhelming results. From this it was determined that mimicking the conditions present in previous research could improve performance.

To recreate urban scenes, gaining as much photoreaslism as possible becomes a key target. During investigations in this area, it was discovered that Blender provides two main render engines, Evee and Cycles. Evee focuses on speed and interactivity within the Blender environment, whereas Cycles targets the highest level of photorealism and quality possible. To meet these targets, Evee does not perform raytracing and as such does not produce photorealistic results. Likewise, to meet its targets, Cycles does perform raytracing, which sacrifices performance but produces photorealistic results. This distinction means that by varying the render engine used, we gain additional control over the lighting conditions within the scene. Accompanying the change of render engine, the presence of extra models was thought to make the scene closer resemble an urban environment. To vary the amount of background detail present in the scene, a cityscape model was added. This cityscape model was scaled and positioned around the model of interest. This resulted in further background

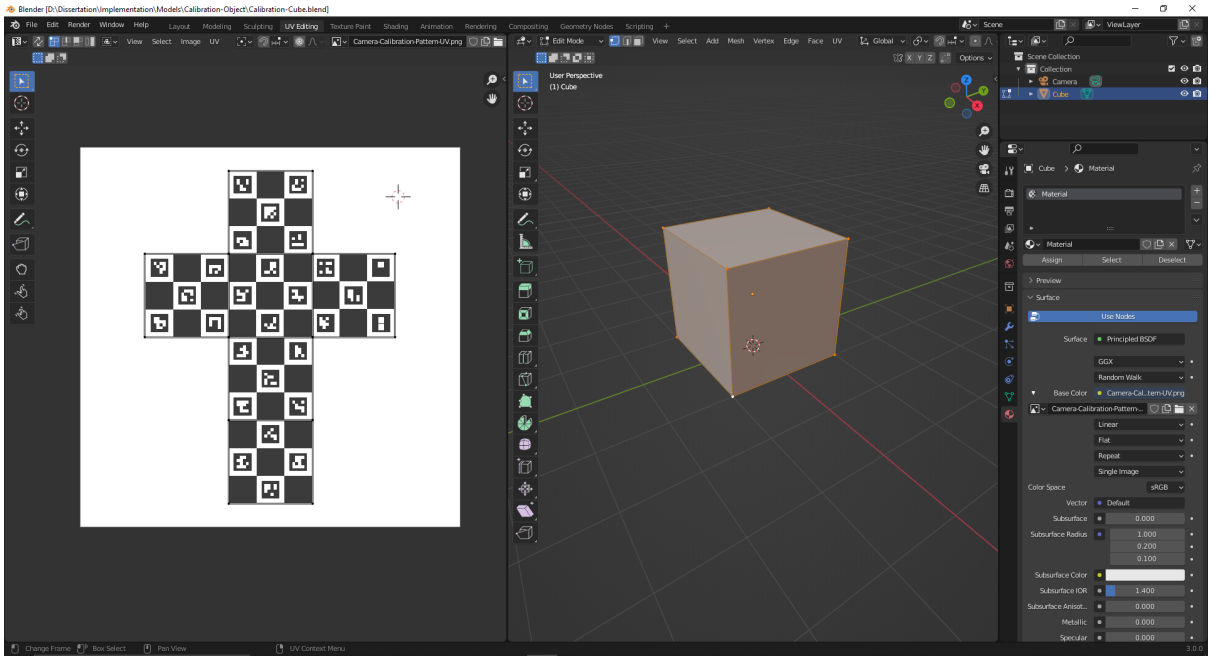

Figure 4.10: Applying image texture to cube model.

information being captured in all images, unlike previous tests with a homogeneous background. Figure 4.11 shows a preview of the cityscape scene setup in Blender and figure 4.12 shows a checkerboard teapot model being captured in the cityscape scene and rendered using Cycles.

Once all the desired scenes have been successfully created, their respective .blend files are saved to a common directory. This directory is then used as the source directory for batch scene capture.

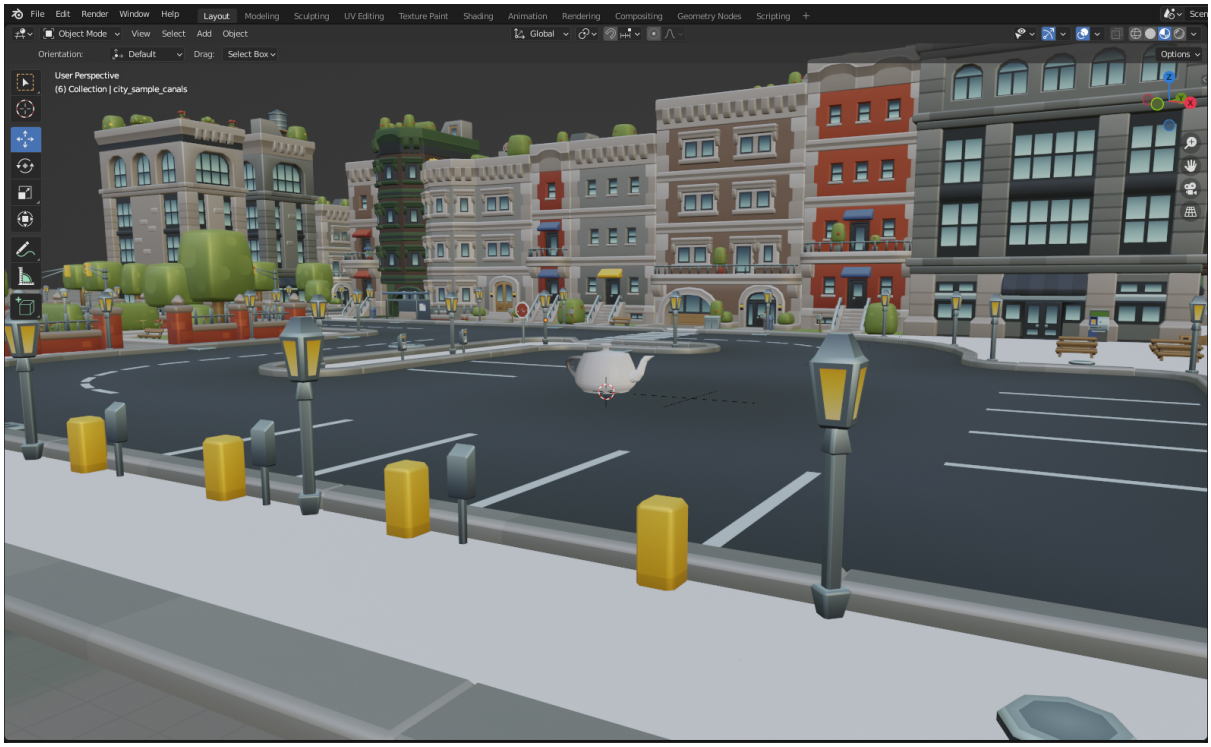

Figure 4.11: Gradient teapot in cityscape scene.

Figure 4.12: Teapot model captured in the cityscape using cycles render engine.

4.6 Scene Capture

After the scene has been created the next phase is capturing images of the object of interest. For this virtual cameras are placed around the scene within Blender. When placing these virtual cameras, we have full control of both the intrinsic and extrinsic parameters with 100% accuracy. For the purposes of these experiments it was decided to use a fixed set of intrinsic parameters for all cameras. A perspective camera model was chosen with focal length of 50mm and sensor width of 35mm. The only factors which varied about the cameras between experiments were the extrinsic parameters and the capture resolution. Due to the virtual nature of the cameras, we were not subject to physical manufacturing discrepancies and thus, we did not need to worry about other intrinsic camera parameters such as skew factor.

4.6.1 Bash Batch Capture Script

The first step to capture images of the various scenes, begins with the use of the capture.sh bash capture script (Appendix A1.2). This script takes a directory of .blend files prepared during scene creation as input, and kicks off the camera placement and image capture logic contained in automation-script.py python script (Appendix A1.3). capture.sh iterates through each .blend file within the supplied directory and starts a capture session. The configuration of each capture session was encoded within this script and could be changed from run to run. Once a .blend file has been selected, a Blender session is created and the desired configuration parameters are supplied using the following command:

```
blender -b "${modelPath}" \
    --python "${captureScriptPath}" \
    -- "${modelOfInterestName}" \
    "${numberOfCamerasPerOrbit}" \
    "${orbitHeights}" \
    "${cameraPlacementPattern}" \
    "${outputDir}"
```

This command starts a Blender session with the selected .blend file and passes through a python script to be executed with Blender's included python runtime. The parameters after the second "–" are ignored by Blender and get passed directly as arguments to the python script. Using this approach in conjunction with the python script, enables different camera placement patterns and frequencies to be tested in the given scene.

4.6.2 Python Capture Script

Camera Coordinate Generation

Once invoked from the bash script, the automation-script.py latches into the Blender environment using the available python api. With this api every action that is available in the GUI, can also be completed programmatically. Once the passed arguments have been parsed, the necessary directories are created and the camera placement logic begins. The first phase of this logic is the placement pattern, for which three options have been developed, circle, square and random. With each of these patterns there is the concept of an orbit height. All cameras, regardless of placement pattern are arranged in orbits around the object of interest. These orbits are in the vertical direction, which in the case of Blender is the z axis. After selecting a placement pattern and set of orbit heights, a set of 3D world space coordinates will be generated for each of the cameras. Following this logic, the number of cameras placed is defined on a per orbit basis, with each orbit containing an equal number of cameras. For example, a full set of coordinates can be defined as: orbitHeights=[0,4,8], cameraPlacementPattern=circle, numberOfCameras=8. This configuration will place a total of twenty-four cameras, arranged in three orbits at z heights zero, four and eight respectively, with each orbit containing 8 equally spaced cameras following a circular pattern. The logic for square and random placement patterns follow the same principles. This camera coordinate generation process is completed by the generate_camera_positions_circle(), generate_camera_positions_square() and generate_camera_positions_random() functions (Appendix A1.3).

Camera Rigging and Capture

After generating a set of camera positions, the camera objects themselves must be instantiated. This logic is encapsulated in the capture_single_camera() function (Appendix A1.3). Regardless of placement pattern, all cameras are instantiated with a set of x,y,z coordinates and are rigged to point directly at the object of interest using Blender's object tracking constraint. The invisible tracking point placed during scene creation is used for tracking rather than the object of interest mesh, which ensures a centered image is obtained. Once a camera is placed, it then captures an image from its view of the scene, under the influence of its intrinsic parameters. Figures 3.2 and 3.3 show an example of a circular placement pattern using this process. This image is then output to the images folder, following the previously mentioned directory structure. Each image is given the same name as the virtual camera that captured it. This naming convention is crucial to identify from which camera an image came, when later comparing it to the ground truth. By varying the number of cameras, location of the cameras and the pattern by which they are placed, we can evaluate the impact of camera placement and image overlap on the final reconstruction

quality.

Ground Truth Output

The final stage of the capture process involves outputting the ground truth information for later use. For this two sets of camera parameters are exported. The first are the camera intrinsic parameters. These parameters are output to the cameras.txt file in the ground_truth folder. This file is formatted to support use with COLMAP as specified in the COLMAP documentation [18]. This specification defines that each set of camera intrinsic parameters should populate a single line following the format CAMERA_ID, MODEL, WIDTH, HEIGHT, PARAMS[]. To achieve this, all the camera intrinsic parameters are extracted from the Blender camera object, and then converted into the correct format for output. Below is an example of the camera intrinsics for an image captured at a 3840x2160px resolution:

```
1 SIMPLE_RADIAL 3840 2160 5333.333333 \
1920.000000 1080.000000 0.000000
```

This single line defines the camera ID, the camera model, the capture resolution, the focal length in pixels and the principle point. In our case the cameras.txt file will only contain one entry, as the camera intrinsic parameters are shared across all cameras per capture session.

The second ground truth output file contains the camera extrinsic parameters. COLMAP expects these parameters in its own view space coordinate system. This differs from the coordinate system used in Blender and as such the Blender coordinates must be converted before being output. In the COLMAP view space coordinate system, the positive x axis points to the right, positive y axis points down and positive z axis points forward. In the Blender view space coordinate system, the positive x axis points right, positive y axis points up and positive z axis points backward. To convert from the Blender coordinate system to the COLMAP coordinate system, a rotation matrix was defined. This rotation matrix inverts the direction of both the y and z axes respectively. Multiplying by this rotation matrix performs the desired coordinate system conversion. This operation is performed in the function get_colmap_view_space_details_json() (Appendix A1.3).

The now transformed camera position and pose is output to the images.txt file in the ground_truth folder. The COLMAP specification for this file defines that each set of extrinsic parameters should populate a single line following the format IMAGE_ID, QW, QX, QY, QZ, TX, TY, TZ, CAMERA_ID, NAME. IMAGE_ID is a unique identifier for that image, with a simple integer being used in our case. QW, QX, QY and QZ are the components of the rotation quaternion, defining the camera pose. TX, TY and TZ are the camera translation vector, locating the camera within the coordinate system. CAMERA_ID

is the identifier of the camera which captured this image and should align with the identifier in the cameras.txt file. For these experiments CAMERA_ID will always be 1 as all cameras have the same intrinsic configuration for a particular capture session. Finally, NAME is the filename of the image captured from this camera. To ensure compatibility, these entries must be ordered following the same ordering convention used by COLMAP's internal database and image importer. After investigating the COLMAP source code, it was found that ascending alphabetical ordering was used and as such the camera entries are sorted using image name as a key before being output. Below is a sample entry from a completed images.txt file.

```
1 0.5183604955673218 0.48093920946121216 0.48093920946121216 \
-0.5183603763580322 3.73739226233738e-06 1.4957995414733887 \
19.943988800048828 1 progcam0.png
```

For further implementation details, the reader is directed to Appendices A1.2 and A1.3.

4.7 Reconstruction

With scene creation and capture complete, each scene now has a corresponding workspace directory containing populated images and ground_truth folders. Using this gathered information, the reconstruction process can begin. This process is defined in the reconstruct.sh bash script (Appendix A1.4). This script iterates through the pending workspace directories and executes the necessary COLMAP commands to perform a complete reconstruction. These commands follow the COLMAP wiki specification for their recommended reconstruction configuration [18] and are executed in the reconstructModel() function (Appendix A1.4). While this recommended reconstruction is performed via the command line, the equivalent process can be completed via the GUI using the automatic reconstruction option. In this section, each of the reconstruction commands and their purpose will be discussed.

4.7.1 Feature Extraction

The first stage of the reconstruction process is feature extraction and can be run using the following command:

```
colmap.bat feature_extractor \
    --database_path "${inputDir}\database.db" \
    --image_path "${inputDir}\images"
```

This command takes the path to the COLMAP project database and the image path as

parameters. If no prior database exists, a new one is created. COLMAP then iterates through all images in the supplied images path and performs feature extraction using the SIFT feature detector. This results with each image having a number of identified keypoints. Figure 4.13 shows an example of such keypoints.

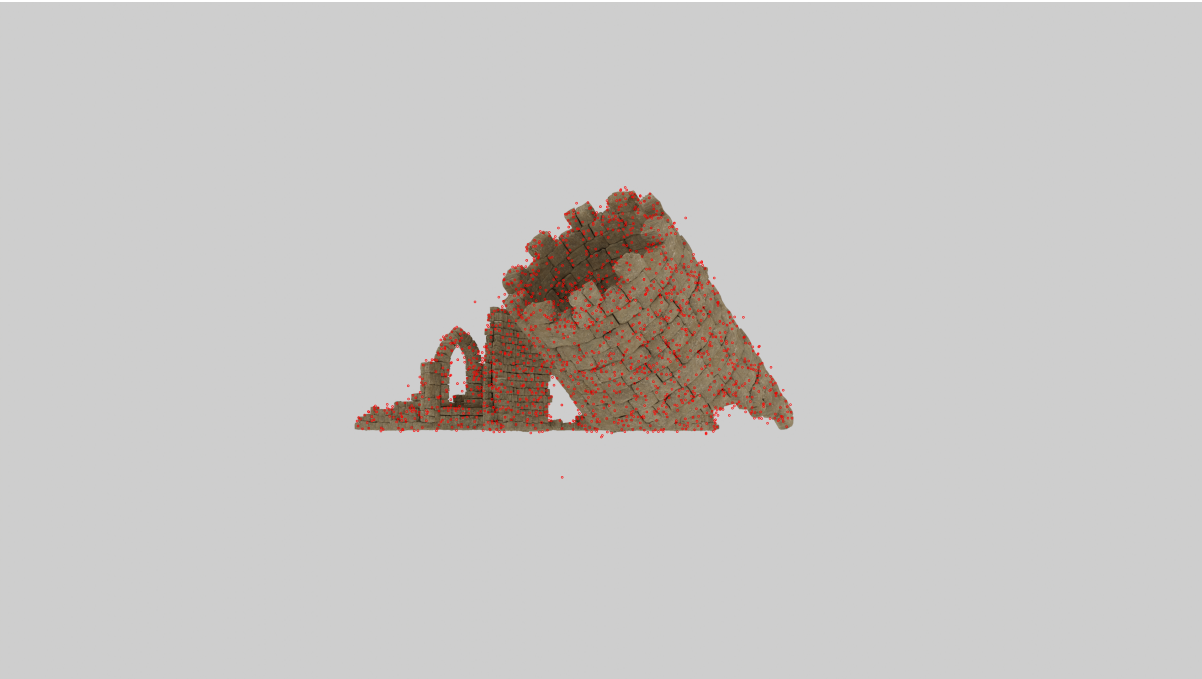

Figure 4.13: Medieval wall model with identified keypoints highlighted in red.

4.7.2 Feature Matching

The second stage of the reconstruction process is feature matching and can be run using the following command :

```
colmap.bat exhaustive_matcher \
    --database_path "${inputDir}\database.db"
```

This command takes the path to the COLMAP project database as a parameter. In this step all the features identified and stored in the database are matched together. The exhaustive matcher is recommended for use here over other options, as while it does take the longest time to run, it finds the best matches amongst the input images. Figure 4.14 shows an example of two side by side images, captured of the medieval wall scene from different viewpoints. Figure 4.15 shows the same two views, but with the common feature points identified and matched between them.

Figure 4.14: Two images of the medieval wall scene captured from different views.

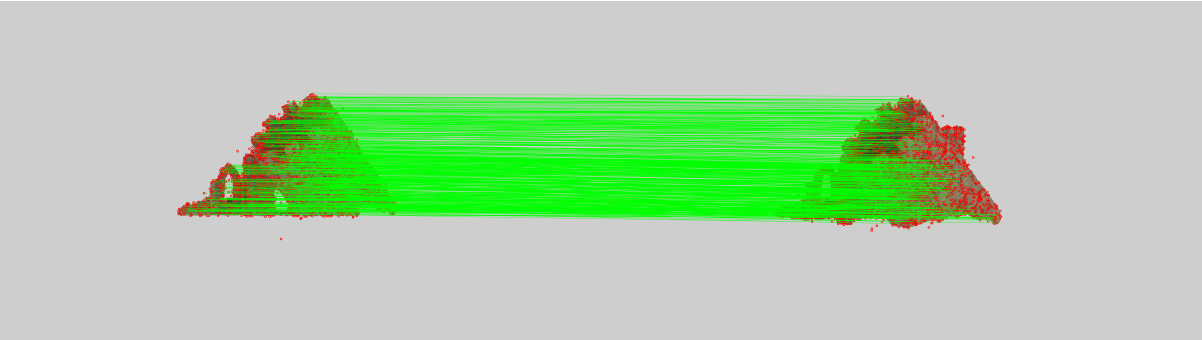

Figure 4.15: Two images of the medieval wall scene captured from different views, with common matched features highlighted in red and their tracked path highlighted in green.

4.7.3 Structure from Motion

The third stage of the reconstruction process is triangulation, where the 2D image keypoints are reprojected out into 3D space using the correspondence information identified during feature matching. At this point we have two options. Option one is to allow the SfM process to run in full and estimate camera parameters from the images alone, using an incremental SfM approach with bundle adjustment. Option two is to manually define the camera parameters using the ground truth information and then allow this to be refined using the SfM process and bundle adjustment. Having performed early testing, model alignment became an issue when conducting final quality assessment. This was the case as the coordinate system used during reconstruction is defined based on the root pair of matched images. This initial pair of images varies from reconstruction to reconstruction and as such, so does the coordinate system. This arbitrary definition of the coordinate system means that the reconstructed model cannot be directly compared to the ground truth without additional alignment and manual intervention. To remedy this issue, it was proposed to supply the ground truth coordinates for reconstruction experiments. This ensures the reconstructed model alignment perfectly matches the ground truth, simplifying quality assessment. Depending on the selected approach, one of two different commands is used. Reconstruction using fully estimated camera parameters can be performed using the following command :

```
colmap.bat mapper \
    --database_path "${inputDir}\database.db" \
    --image_path "${inputDir}\images" \
    --output_path "${inputDir}\sparse"
```

This command takes the path to the COLMAP project database, image-set and desired output directory for the sparse model as parameters. Reconstruction using ground truth camera parameters can be performed using the following command :

```
colmap.bat point_triangulator \
    --database_path "${inputDir}\database.db" \
    --image_path "${inputDir}\images" \
    --input_path "${inputDir}\ground_truth" \
    --output_path "${inputDir}\sparse"
```

This command takes the path to the COLMAP project database, image-set, ground truth information and desired output directory for the sparse model as parameters.

Regardless of the option taken, this command will follow the iterative SfM approach, registering images one by one with a round of bundle adjustment after each registration. The execution time of this process scales quadratically with the number of images, resulting in this phase of the reconstruction taking a significant amount of time. Supplying the ground truth data does expedite this process, as little to no bundle adjustment is performed, thus having the desired effect of orienting the model correctly. Figure 4.3 shows an example of a completed sparse reconstruction from this command, viewed in COLMAP's GUI.

4.7.4 Image Undistortion

The fourth stage of the reconstruction process is the beginning of dense reconstruction. At this point camera parameters have been estimated and using these parameters, a sparse model has been created. However, this sparse model is not sufficient to produce a 3D mesh by itself in most cases. Thus, densification of the sparse model is desirable. The sparse model can be used as initialisation and guidance for the dense model, with this entire process known as MVS. The first step of MVS is to undistort the input images using the camera intrinsic parameters estimated in previous stages. This can be achieved using the following command :

```
colmap.bat image_undistorter \
    --image_path "${inputDir}\images" \
    --input_path "${inputDir}\sparse" \
    --output_path "${inputDir}\dense" \
    --max_image_size 2000
```

This command takes the path to the image-set, the sparse reconstruction, the desired output directory for the dense model and a max image size as parameters. The max_image_size parameter is used to limit the resolution of the images when they are being undistorted. This 2000 pixel limit was used as recommended by the COLMAP documentation to give optimal results and provide greater runtime stability. During testing it was found that omitting this limit resulted in frequent crashes and failed reconstructions.

4.7.5 Multi-View Stereovision

The fifth stage of the reconstruction encapsulates the MVS process. This is completed using the two commands listed below. These commands take the undistorted image-set and calculate depth and normal information for the scene. During this process, geometric consistency is verified and finally, a fused dense point cloud is produced.

```
colmap.bat patch_match_stereo \
    --workspace_path "${inputDir}\dense" \
    --PatchMatchStereo.geom_consistency true

colmap stereo_fusion \
    --workspace_path "%WORKSPACE_PATH%\dense" \
    --input_type geometric \
    --output_path "%WORKSPACE_PATH%\dense\fused.ply"
```

4.7.6 Meshing

The sixth and final stage of the reconstruction process is meshing. COLMAP does provide two meshing options, poisson and delaunay. Poisson meshing is recommended by COLMAP for the best results. Through testing it was confirmed that poisson meshing indeed produced a mesh more faithful and accurate to the ground truth and thus it was selected for use. Poisson meshing can be run using the following command :

```
colmap.bat poisson_mesher \
    --input_path "${inputDir}\dense\fused.ply" \
    --output_path "${inputDir}\dense\meshed-poisson.ply" \
    --PoissonMeshing.trim 8
```

This produces the final mesh for use in 3D applications and in our case, comparison to the ground truth. Figure 4.16 shows a completed reconstruction of the medieval wall model being previewed in MeshLab.

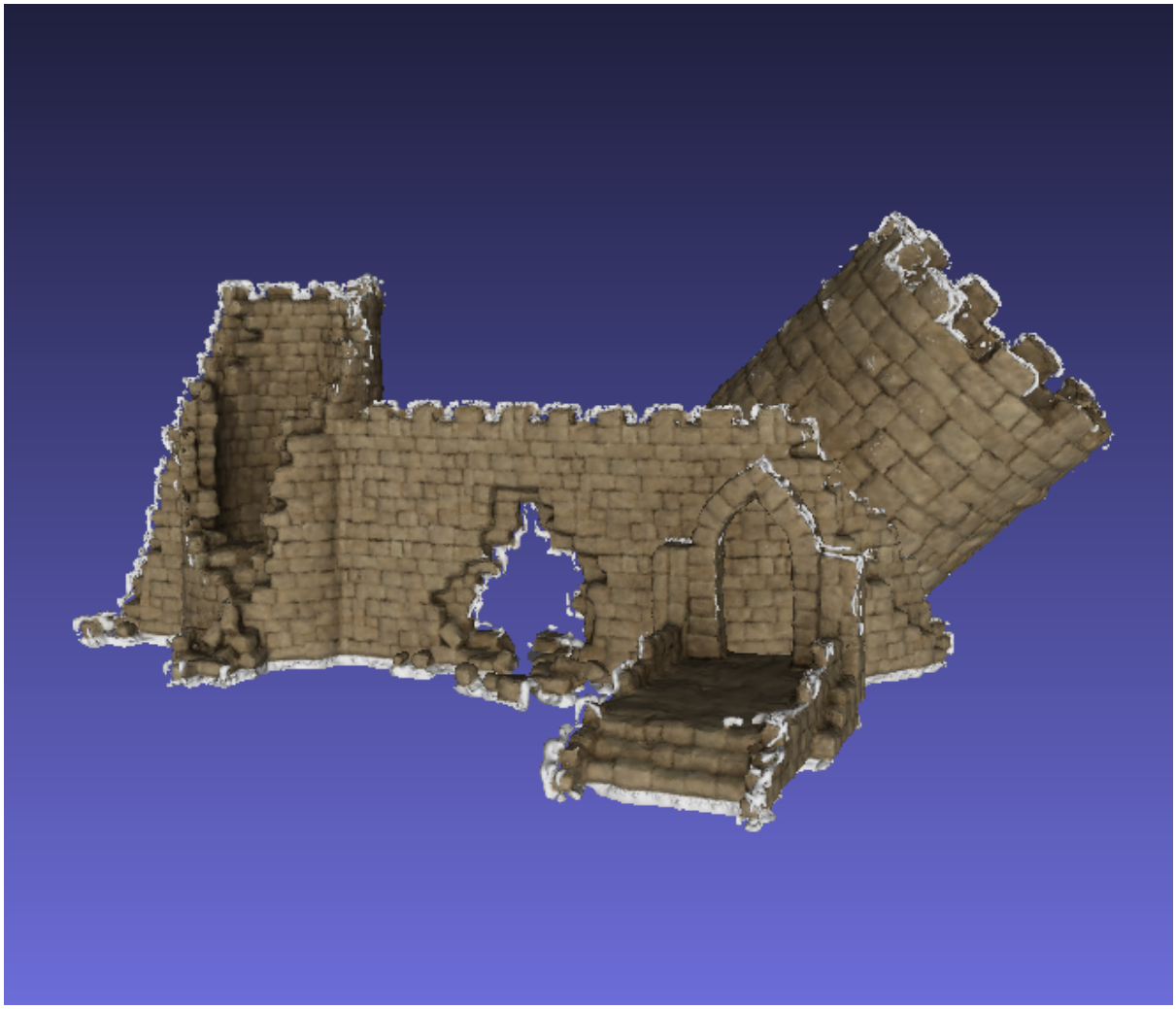

Figure 4.16: An example of a completed reconstruction of the medieval wall model.

4.8 Mesh Quality Evaluation

With the final reconstructed mesh complete, it is now possible to compare this mesh to the original. To perform this comparison, both the ground truth and reconstructed models are loaded into MeshLab. Using MeshLab's included Hausdorff distance function, the vertices of the reconstruction can be sampled and compared to the nearest vertex of the ground truth. The Hausdorff distance is calculated for these two vertices and the resulting value is stored as a quality attribute, attached to the vertex belonging to the reconstruction. By performing this calculation we can get an overall similarity measure between the two models. This resulting quality attribute is a signed and scaled distance measure and thus can be used to show how far above or below the reconstructed vertex is in relation to the ground truth. These quality attributes can then be visualised using a heatmap, which indicates areas of similarity using a colour spectrum. This comparison process can be followed visually in figures 4.17, 4.18 and 4.19. Figure 4.17 shows the ground truth teapot mesh, which is being compared to the reconstructed mesh in figure 4.18. Having computed the Hausdorff

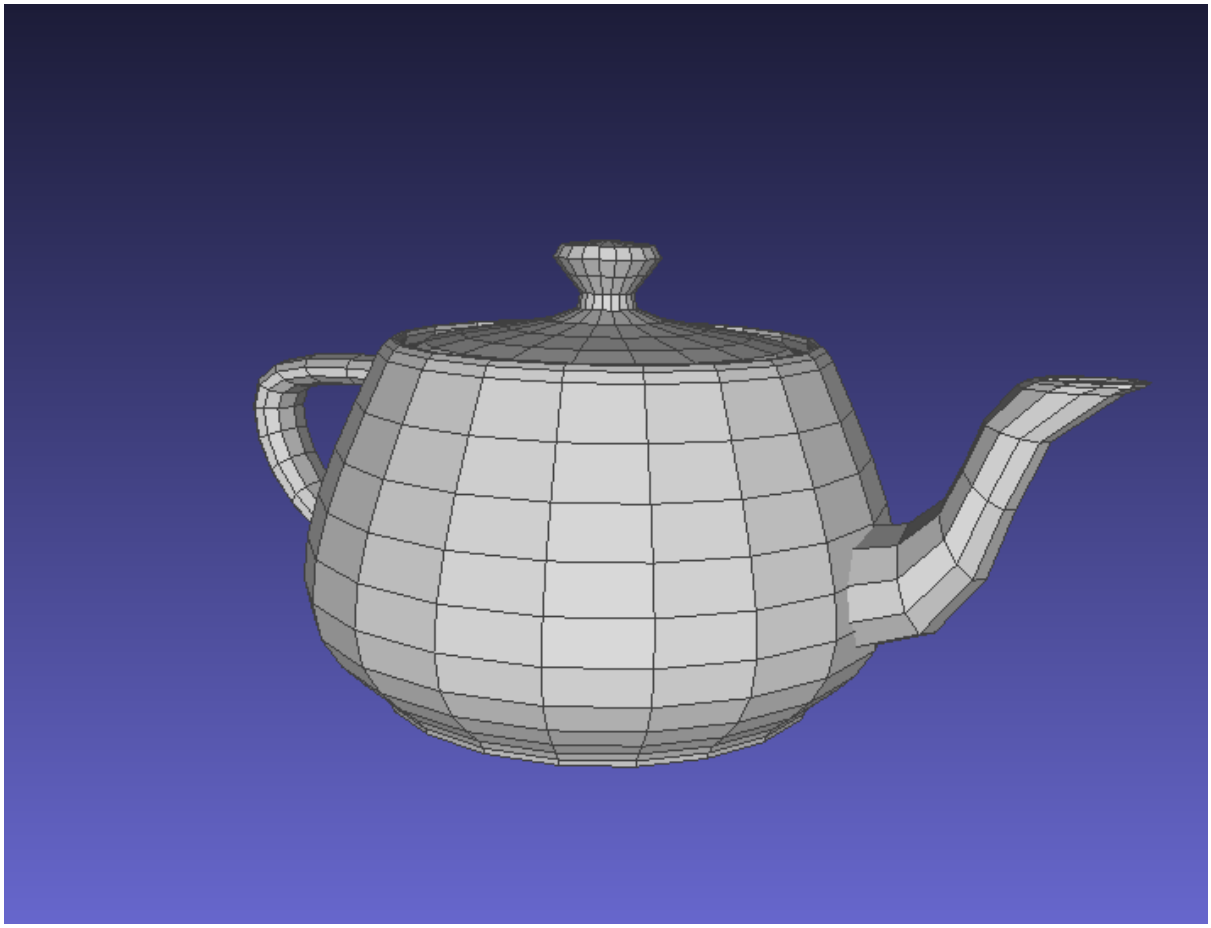

Figure 4.17: The ground truth teapot model opened in MeshLab.

distance, the heatmap in figure 4.19 can be produced to visualise the differences. The colour spectrum for this heatmap is setup starting from blue and ending with red. Visually this leads to points which are below the surface of the ground truth being represented as blue, and points above the surface of the ground truth being represented as red. Figure 4.20 shows the configuration of this spectrum in MeshLab. However, as apparent from figure 4.19, this is not always as useful as one would hope. This measure does suffer when there are outliers in the reconstruction, as the distance of the outliers from the ground truth can skew the scale used when visualising all points. The effect of this can be noted in figure 4.19, where little useful information is displayed on the heatmap due to the presence of outliers situated above the teapot, which can be seen in red. It is possible to manually adjust the scaling of the spectrum to account for outliers, using the included equaliser in figure 4.20. By manipulating the three available sliders in GUI, it is possible to present finer visual details about the mesh surface. This can be achieved by centering the three sliders around the distribution presented in the equaliser preview window. The results of this process can be seen in figure 4.21, where it is now possible to determine small surface details about the reconstructed mesh.

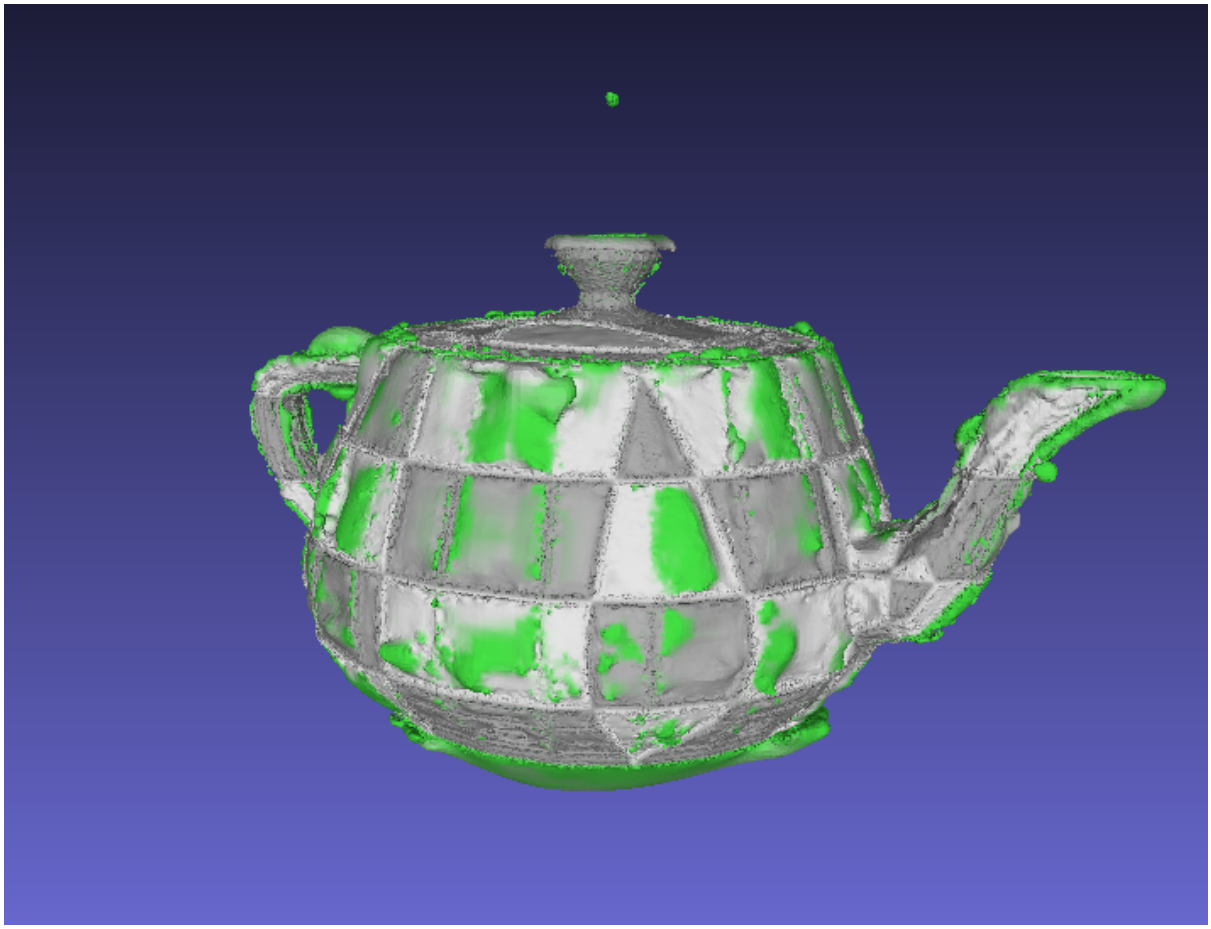

Figure 4.18: A reconstructed teapot model opened in MeshLab.

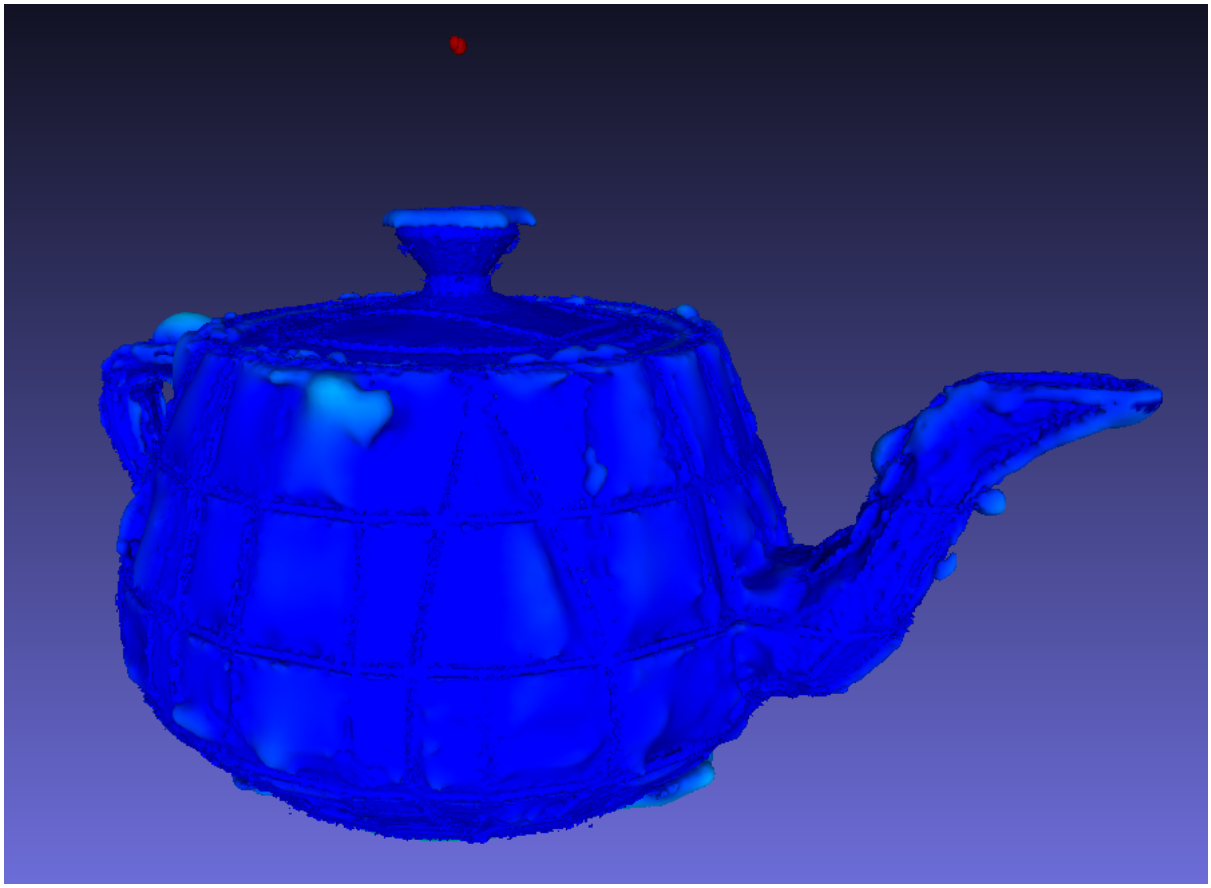

Figure 4.19: An example quality heatmap generated in the presence of outliers.

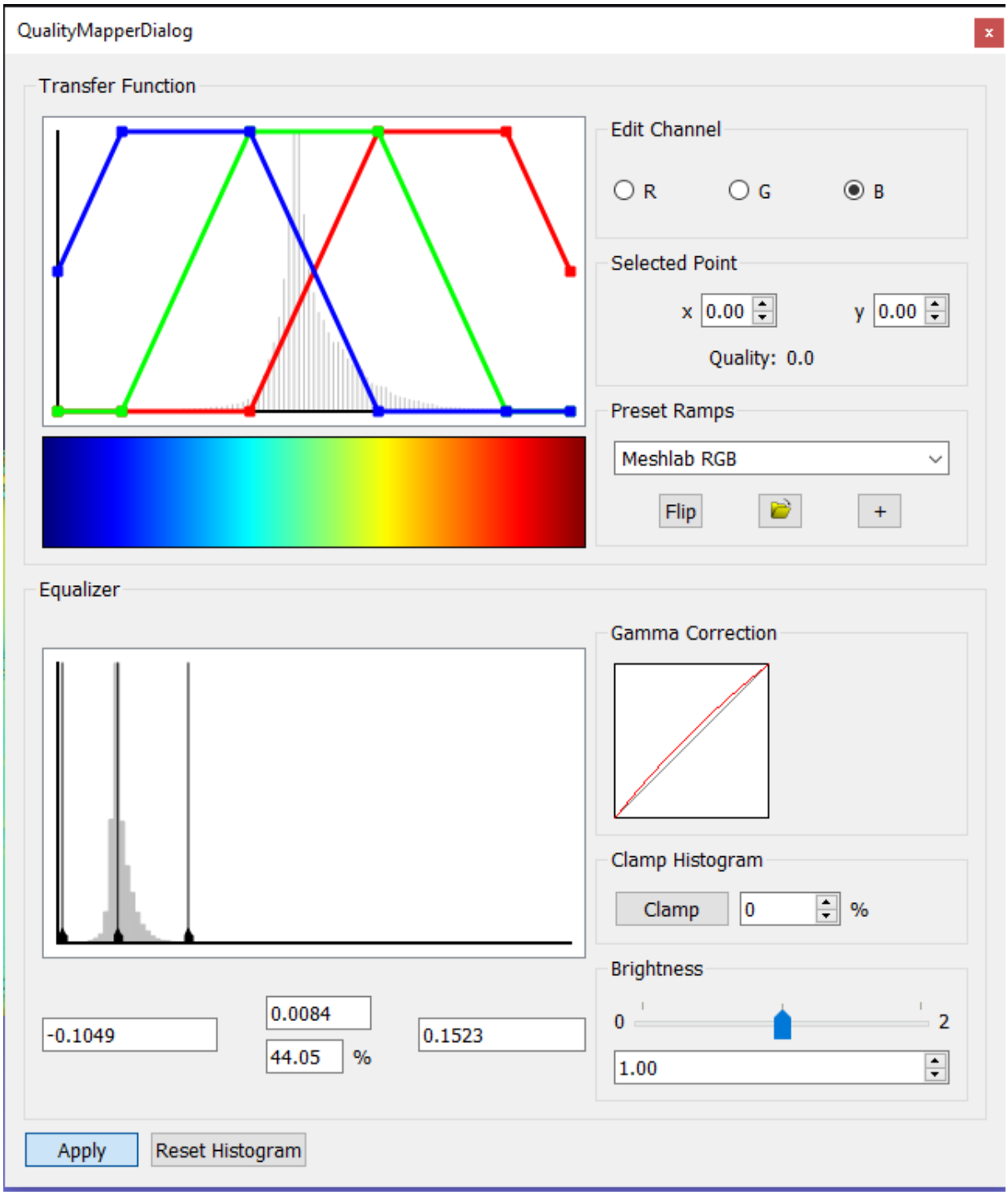

Figure 4.20: The MeshLab quality visualisation control window.

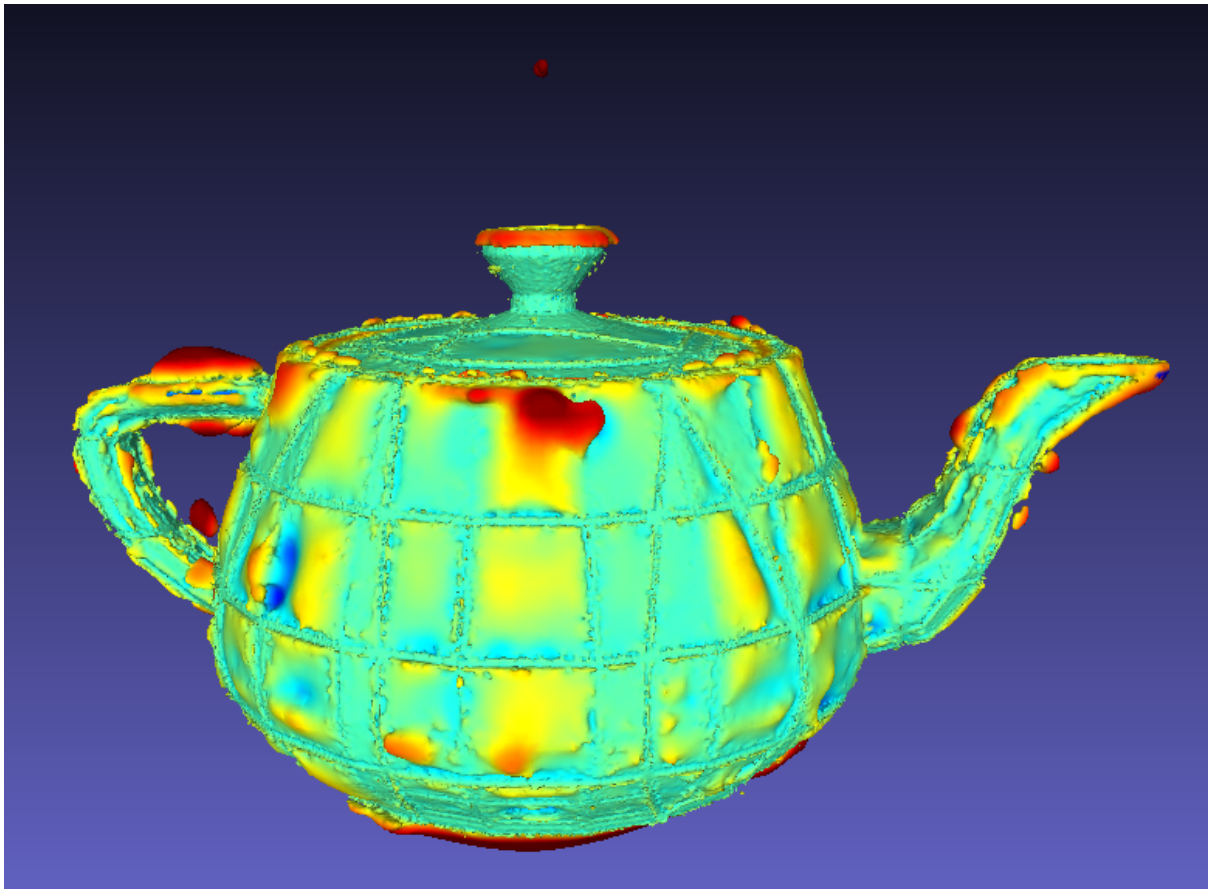

Figure 4.21: An example quality heatmap which has been corrected to account for outliers.

5 Evaluation

In this section the experiments performed as part of this research will be discussed and their results evaluated. The challenges and limitations of the experimental methods used throughout the course of research will be also be discussed.

Results

In total, over 137 experiments were ran over the course of four months,, with varying degrees of success. These are split into four phases of experiments. Each of these four phases target a varying selection of impacting factors for testing, with the overall goal to reach the best results possible and learn about the characteristics of each impacting factor. All data collected during this testing is available in section A1.1. The following sections will discuss the factors tested and the results obtained during each phase of experiments.

5.1 Phase 1

5.1.1 Model and Scene Configuration

To begin, basic models were used to create the initial iterations of the experiment framework. These scenes were created with minimal environmental details beyond the inclusion of the model of interest. Pre-made models included in Blender were used to get started, including a cube and teapot. Models were located on the origin of the 3D editor and rendered without any texture or surface reflectance. A homogeneous grey skybox was used along with global ambient lighting. Figure 5.1 shows a preview of these initial experiments using the teapot model.

5.1.2 Impact of Texture and Geometry

After initial testing, it was found that the lack of detail in the scene resulted in failed reconstruction attempts. The low level of detail on the object of interest, the symmetry present in its geometry and poor segmentation between the model and the background,

prevented sufficient keypoints being detected at the feature extraction phase of COLMAP. This propagated into the feature matching stage, where no matches could be made. Ultimately, this resulted in a failed reconstruction attempt with not even a sparse model being generated.

In the next round of testing, the impact of object texture was evaluated. Once again the set of models were captured, but this time with a varying suite of textures, including matte black, checkerboard, gradient and completely unique. A green skybox was used to improve segmentation between the object and the background. Taking the same battery of tests, the sparse and later dense reconstructions were then performed. The black texturing performed poorly as we again had the issue of a low level of detail on the object of interest. The Checkerboard texture improved upon this as more keypoints were identified, however the repeating nature of the texture resulted in incorrect matches being made during the feature matching stage of the reconstruction pipeline. Figure 5.2 shows an example of two images of the checkerboard teapot model being matched incorrectly, one images shows a clear view of the teapot's handle and the other its spout. Applying these additional textures to the teapot, indeed helped performance, as less failures were occurring despite image mismatching. However, it should be noted that these reconstructions were still unrecognisable as a teapot.

Having observed the impact of texture on the reconstruction, an additional test was performed using the cube model. This model is completely symmetrical in both horizontal and vertical directions, making it difficult to reconstruct. This test aimed to see if favourable texturing could overcome the difficulties associated with reconstructing a symmetrical object. For this a completely unique texture was created by hand using image editing software. This was applied to the cube giving it a unique appearance on every face. This test is named the calibration cube test and from table A1.1.2 we can see that this texture did indeed result in a high number of keypoints detected, matching that of the previous best tests when using the teapot model with more complex geometry. Figures 4.6 and 4.10 show examples of this scene configuration.

5.1.3 Observations and Conclusions

By the end of this phase of testing, both the positive and negative impacts of texture were identified, along with the impact of object geometry. From this we can conclude that symmetric geometry is difficult to reconstruct as mismatches are common, as we saw with the teapot spout and handle. Texture can be used to improve these results, with unique surface texturing being optimal for this. Texture also has a pivotal role in emphasizing geometry on the model of interest, increasing the chances of detection. If a single homogeneous texture is used on the model, little to no keypoints are detected and feature

matching fails in many cases. These results verify those that were given in the literature discussed in section 2.

At this point, challenges comparing the reconstructed mesh to the ground truth were identified. As discussed in section 4.7.3, the orientation of the final model required manual intervention to align with the ground truth. This is the case due to the reconstructed coordinate system used varying from reconstruction to reconstruction. As we have just discussed, during testing it was found that the symmetric nature of the models being reconstructed resulted in failed and incorrect image registration. These outcomes had the impact of preventing final quality assessment against the ground truth. Figure 5.3 shows a completed reconstruction of the checkerboard teapot, which suffered a number of incorrect image registrations, leaving the final mesh unrecognisable. Furthermore, it became desirable to improve image registration and orient the model correctly for the next phase of experiments.

Figure 5.1: Phase 1 grey teapot example

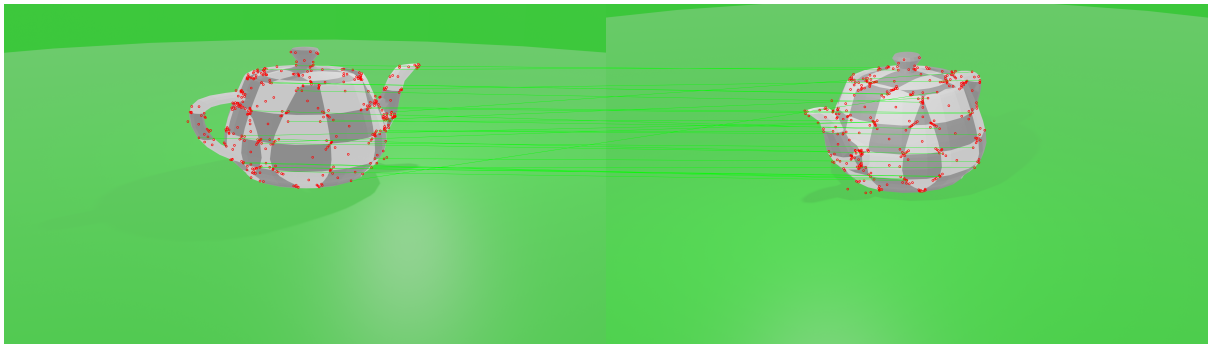

Figure 5.2: Phase 1 checkerboard teapot with mismatched keypoints

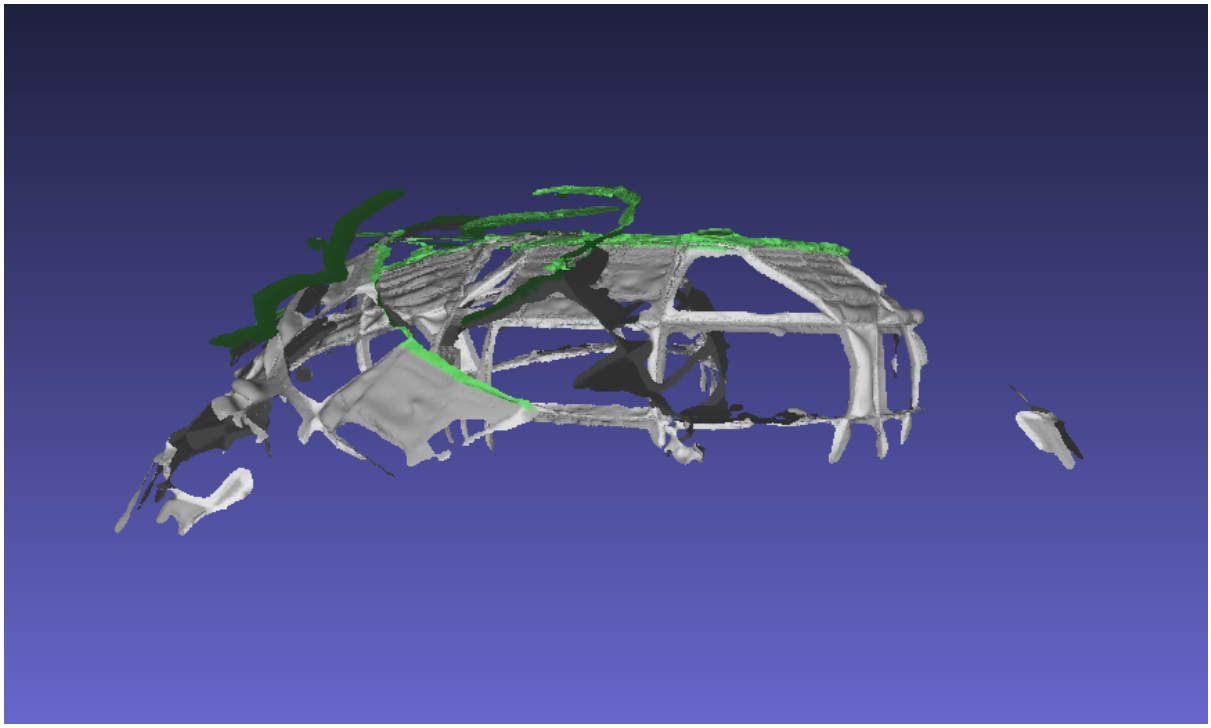

Figure 5.3: Phase 1 poor teapot reconstruction due to symmetry challenges.

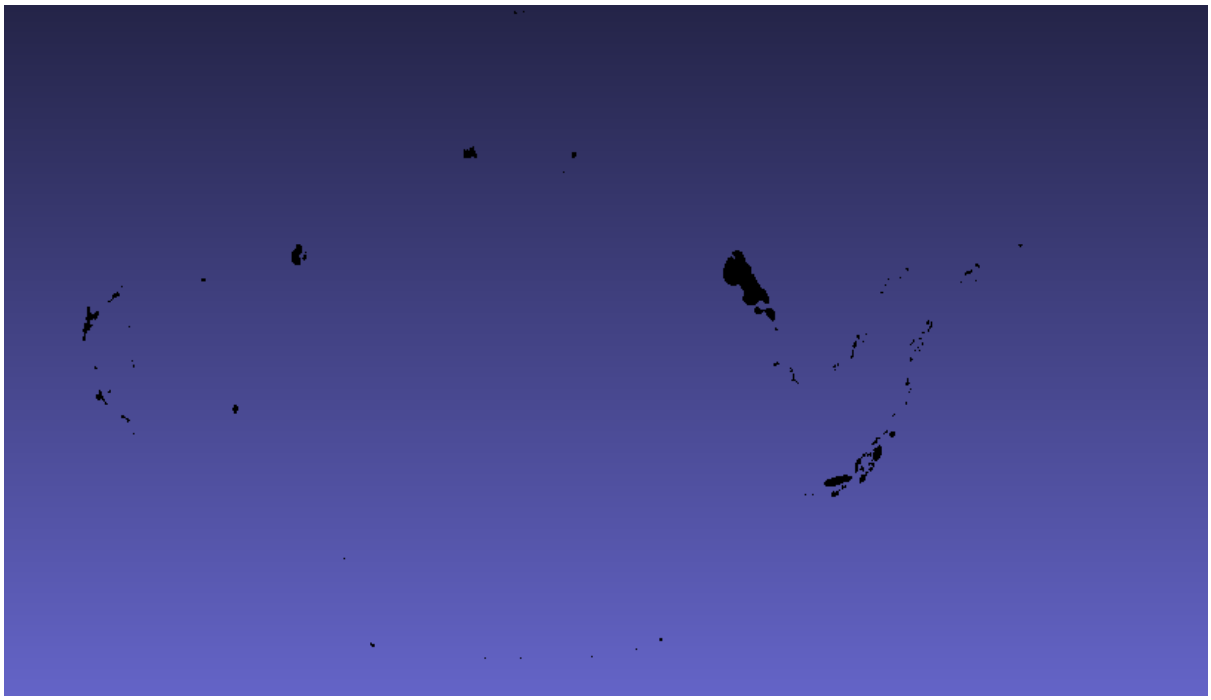

Figure 5.4: Poor teapot reconstruction result barely showing an outline of the teapot.

5.2 Phase 2

5.2.1 Updates to Testing Framework

The first key change implemented during this phase of experiments, was orienting the model to align with the ground truth model. As discussed throughout section 4, this involved the output of ground truth camera information from the capture phase and its later use during the reconstruction process. By utilising COLMAP's *point_triangulator* command, the formatted ground truth information from Blender can be used as a partial result, in which the cameras are already located within the scene. This has the effect of fixing the coordinate system in place. This does however have the additional impact of removing much of the camera estimation, which is performed during SfM. When the ground truth cameras are supplied in this way, only a single round of bundle adjustment is performed and the camera extrinsics remain unchanged. This was a necessary step to ensure accurate final comparison to the ground truth. While this change does impact the amount of the reconstruction pipeline being used, it does not prevent the investigation of the impacting factors.

5.2.2 Model and Scene Configuration

With the updated testing framework, this phase of experiments were designed to further measure the impact of texture on the final reconstruction. While this is the primary factor under investigation, it remains closely coupled to factors such as image distribution, number of images and image resolution. With these considerations in mind, this phase of testing investigates the impact of texture, using distribution, resolution and number of images as devices to instigate change. To investigate the impact of texture, the teapot model was once again used with varying textures applied to it. The selection of textures include black, gradient and checkerboard textures, with black representing a worst case texture and gradient and checkerboard acting as potential improvements we wish to examine.

Beyond the model of interest, much of the scene remained the same as in phase 1. A green skybox was used in the hopes of improving background segmentation. The scene was lit using an ambient light source to minimise the impact of shadows. Updates were made to the camera resolution and render engine configurations for these experiments. Camera resolution was varied between 1920x1080px and 3840x2160px, respectively. These resolutions are representative of the capabilities present in most modern smartphones, giving us an appreciation for how much detail is captured by using a lower or higher amount of quantisation. While only ambient lighting was used, the raytracing implemented by the Cycles render engine produced more realistic images, particularly when capturing the finer details of these models. The Evee render engine acted a simplified scene, from which we can reason about how much detail is necessary for reconstruction to be performed

successfully.

5.2.3 Impact of Texture

Homogeneous Texture

As a baseline, it was decided to setup a scene to measure how much of an improvement the updates to the testing framework made to reconstruction quality. Using the Evee render engine, a known difficult reconstruction scenario could be tested, that of a homogeneous black textured teapot. Rather than outright failing like phase 1 testing, this time the reconstruction process completed in full and a dense model was produced. However, this model was largely incomplete beyond capturing a small amount of the teapot outline. Figure 5.4 shows the final reconstructed mesh from this test.

Turning attention towards table A1.1.2, we can see that even for this poor reconstruction, all the images supplied were registered as part of the reconstruction. This is a characteristic which was not observed across all tests in phase 1 and comes as a result of fixing the cameras in place using the ground truth. Thus, we note the number of images successfully registered becomes a less useful metric. The average number of keypoints for this experiment is a more telling metric. Due to the now fixed nature of the cameras, the number of 3D points successfully triangulated ultimately depends on the success of keypoint detection. The poor reconstruction results of this experiment further shows how difficult it is to capture objects with homogeneous textures, as feature detection struggles to identify unique keypoints. Further tests were performed using this homogeneous texture, this time varying the render engine and lighting conditions by adding a point light source. These results did not uncover any additional information as they performed similarly to those already discussed in this section. As such I will not discuss these results further, however they have been included as appendix A1.1.3 for completeness.

Gradient Texture

An area of possible improvement upon the homogeneous texture results, was through the use of a gradient texture. The texture selected was a simple black to white gradient moving from the handle of the teapot to the spout. Figure 4.9 shows an example of this texturing applied to the teapot. It was hoped that the use of this texture along with the flat rendering approach taken, would emphasise the geometry of the model. The quality of the final mesh and the number of keypoints identified were key metrics of interest with these experiments.

To evaluate the impact of this texture under varying conditions, the capture distribution, resolution and number of images were varied. These tests can be grouped by capture resolution and then by placement pattern. Figure 5.5 gives an illustration of how these

experiments were organised. Primary testing was performed using 4k images, with our first area of interest being placement pattern and the second being the number of images. Finally, a set of images were captured at 1080p to measure the impact of the lower resolution.

Placement Pattern

Examining the results obtained from these tests, it becomes possible to ascertain if placement pattern does indeed impact reconstruction results. Graphs 5.6 and 5.7 were plotted to examine the placement pattern relationship using the mean keypoints, number of 3D points and number of images registered as indicators. Looking at graph 5.7 we can see that the square placement pattern consistently identifies a higher number of keypoints when compared to the circle placement pattern. However, when viewing graph 5.6 we see this does not necessarily translate to a significant increase in the number of 3D points being triangulated. For the tests where we have an equal number of images taken with both square and circle distributions, we see the square placement pattern consistently identifies more keypoints and 3D points, but only marginally.

A visual inspection of the final mesh produced from each placement pattern tells a different story to the SfM results. From this inspection we see that the circular placement pattern produces a model with a much higher number of vertices and faces when compared to the square placement pattern. Figures 5.8 and 5.9 show both of these results respectively. Note how the circular placement pattern results in better overall model coverage when compared with the square placement pattern. We also note the large empty portions of the model, these correspond to homogeneous regions of texture and thus, little to no detail was gathered during keypoint detection.

Number of Images

Regardless of the placement pattern all tests exhibit the same increasing linear trend. Across the board we saw increased keypoint detection and 3D point triangulation with each increase to the image-set size. Visual inspection of the final meshes confirms that this did indeed improve mesh quality and coverage with each increase. Due to timing constraints it was not possible to test larger image-sets. Therefore, it is uncertain if this trend continues with larger image-sets beyond what was tested here.

Image Resolution

The final gradient tests performed were at a lower 1080p resolution. This decrease in resolution did not result in a significant drop off to the number of keypoints identified, when compared to the 4k tests with the same number of images. However, visual inspection showed the final meshed results to be of much lower quality. They feature much less

complete geometry when compared to their 4k counterparts and include a greater amount of noise around the edges of the mesh. Furthermore, we see the final texturing and geometry in general to be more blotchy and less refined. Figure 5.10 shows an example of the final reconstructed model using a 1080p image-set. Note the large amount of noise and green fringing in the model.

Checkerboard

Performing the same suite of experiments using the checkerboard texture, yielded similar trends to those observed with the gradient texture. Looking at the results obtained from these experiments in table A1.1.5, we see that the checkerboard texture yielded increases across the board in both mean number of keypoints and the number of 3D points reprojected, when compared to the gradient texture results. Visual inspection of the final models produced shows that the bottom of the teapot is better covered compared to the gradient. This can be attributed to the extra corner keypoints detected due to the nature of the checkerboard texture. We can also see that the homogeneous regions of the checkerboard texture are unfilled in these reconstructions, showing how they were not useful in the collection of data. Figure 5.15 shows a heatmap representation of the final reconstruction results (Figure 5.14 when compared to the ground truth. Here we see that the majority of the reconstructed geometry is close to that of the ground truth, indicated in yellow and green. Note that there still remains a small amount of background fringing around the edges of the reconstruction, indicated in red.

5.2.4 Observations and Conclusions

From this phase of experimentation, it was found that the use of ground truth camera information could be used to great effect in orienting the final reconstruction. This ground truth camera information also helped in minimising the negative effects of object symmetry. While we supplied exact ground truth information in these experiments, similar improvements could be hoped through the use of GPS coordinates as discussed in section 2. However, even with the help of ground truth data, homogeneous texturing performed quite poorly. In general, areas of homogeneous texture are missing from reconstructions due to a lack of detail.

These experiments found that using texture with additional details has the effect of improving reconstruction quality. The use of both gradient and checkerboard textures improved reconstruction results, with checkerboard giving the overall best coverage of the model of interest. During these experiments it was determined that increasing the number of images captured improved the performance of the reconstruction linearly regardless of the capture pattern undertaken. Finally, we saw that a circular placement pattern outperformed a square placement pattern in the same conditions, making it preferable for scene capture.

Following this, I recommend end users capture their scene of interest using a circular pattern around their point of focus, capturing as many evenly distributed images as possible.

```
Phase-2-Gradient-Tests
├── 3840x2160
│   ├── Square Placement Pattern
│   │   ├── 12 Images
│   │   ├── 24 Images
│   │   ├── 36 Images
│   │   └── 48 Images
│   └── Circle Placement Pattern
│       ├── 12 Images
│       ├── 24 Images
│       ├── 36 Images
│       ├── 48 Images
│       ├── 54 Images
│       ├── 108 Images
│       └── 216 Images
└── 1920x1080
    └── Circle Placement Pattern
        ├── 18 Images
        ├── 36 Images
        └── 72 Images
```

Figure 5.5: Structure of phase 2 teapot gradient texture experiments.

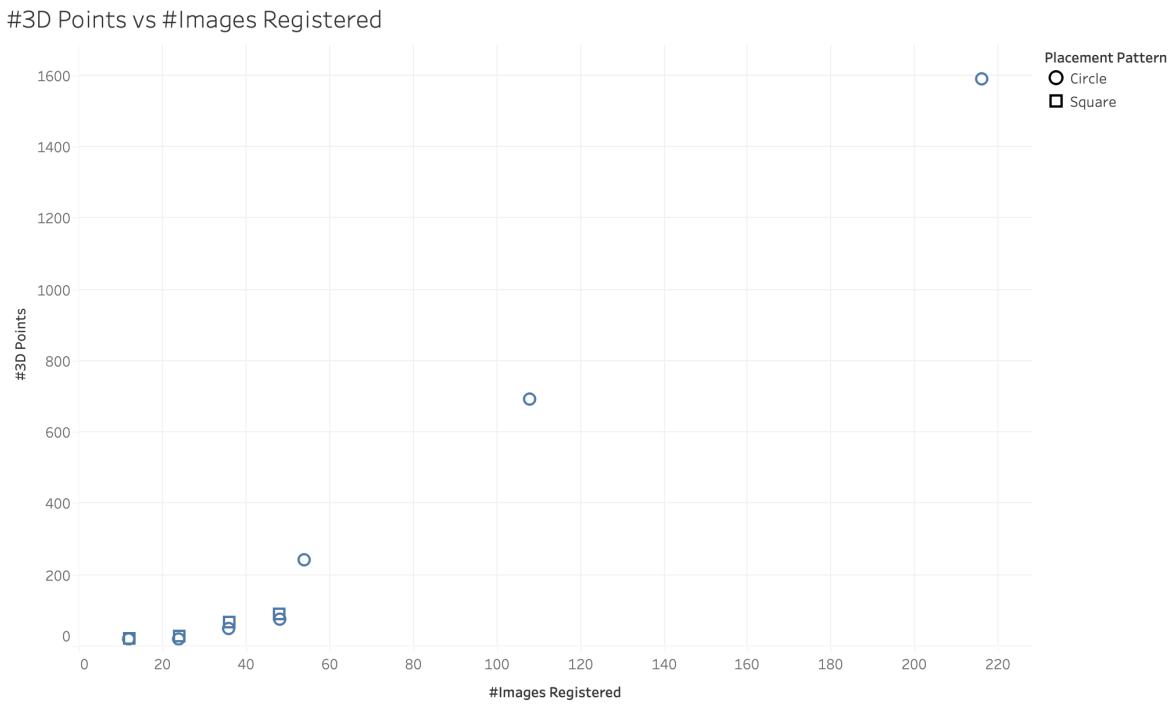

Figure 5.6: Graph plotting number of 3D points reprojected vs the number of images registered for the phase 2 gradient teapot experiments.

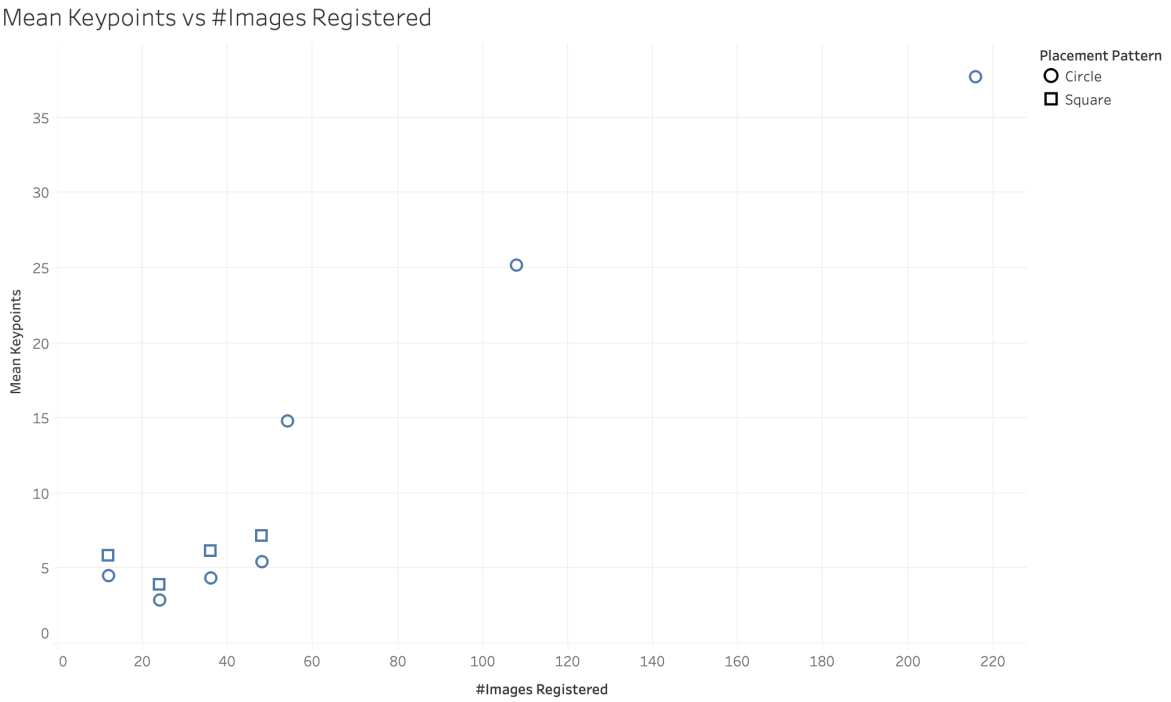

Figure 5.7: Graph plotting mean number of keypoints detected vs the number of images registered for the phase 2 gradient teapot experiments.

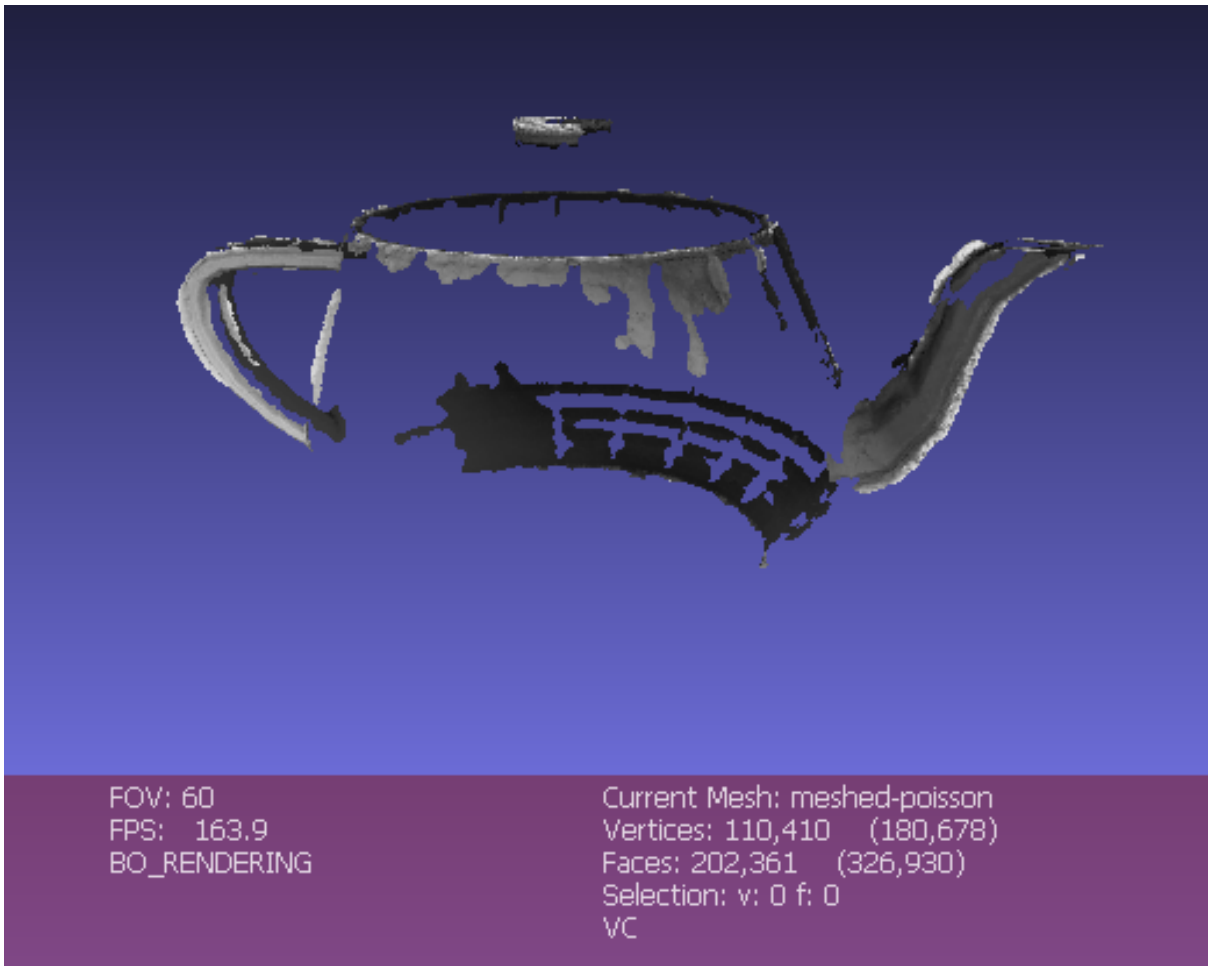

Figure 5.8: The reconstructed gradient teapot model captured with a circular placement pattern at 4k resolution with 216 images.

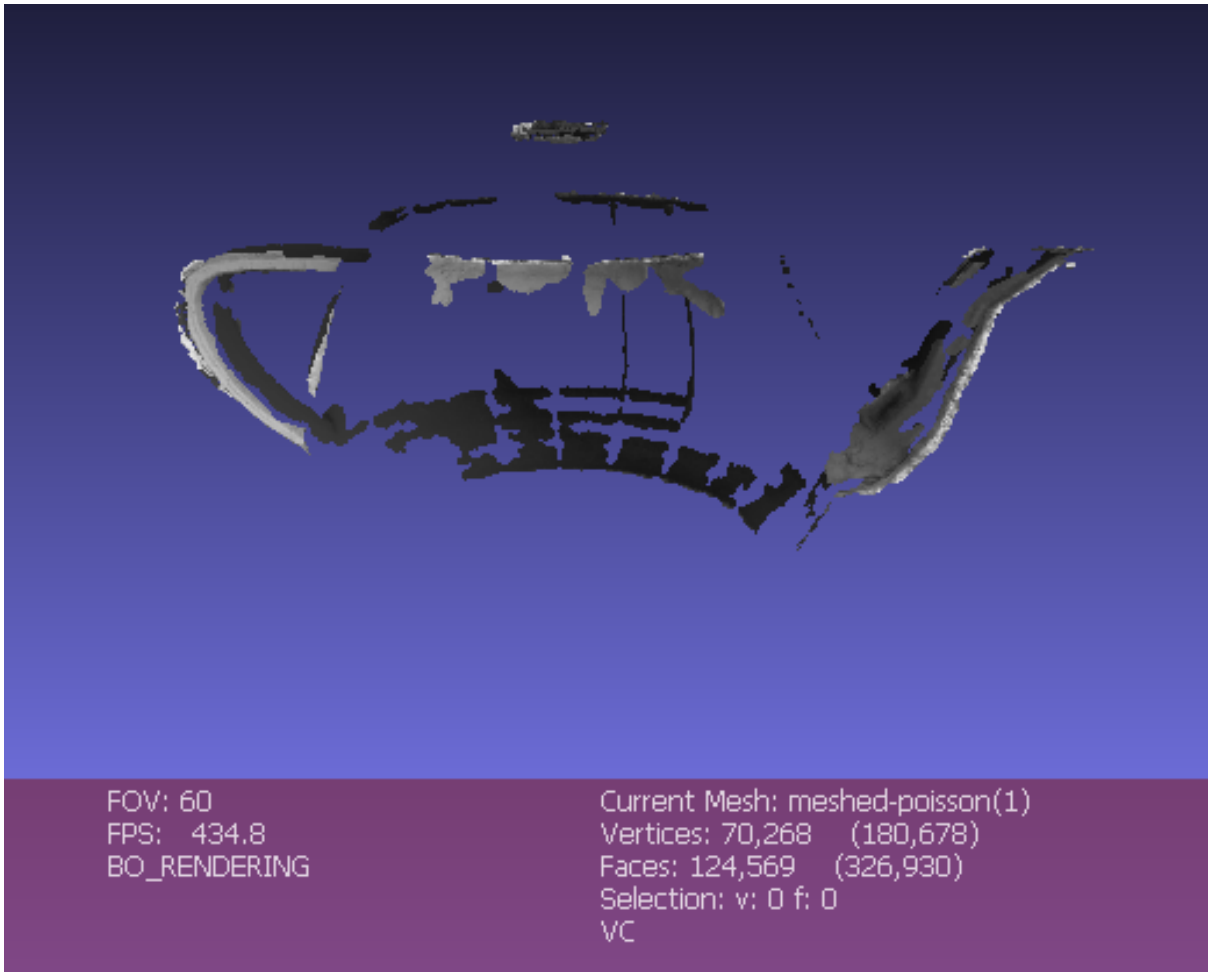

Figure 5.9: The reconstructed gradient teapot model captured with a square placement pattern at 4k resolution with 216 images.

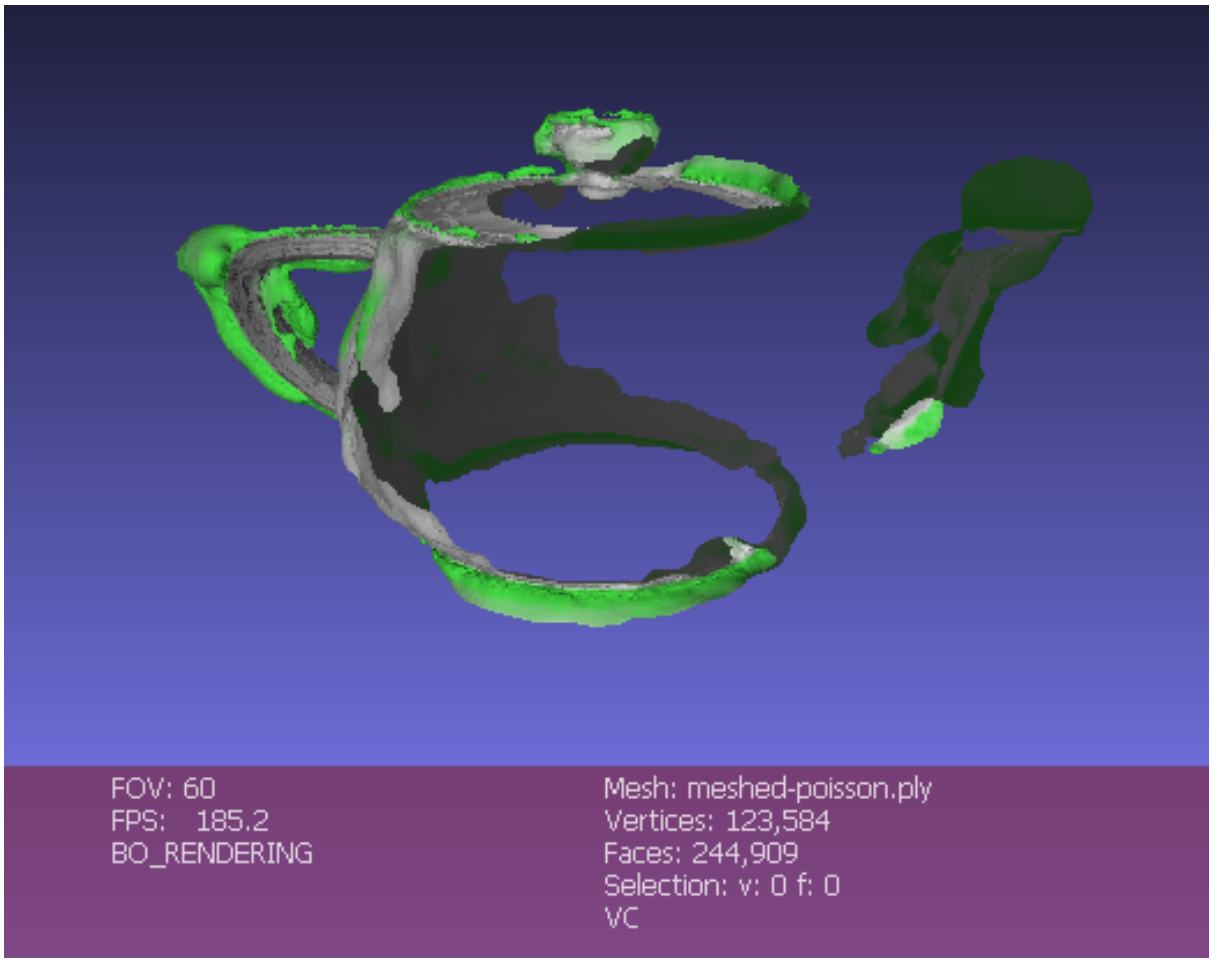

Figure 5.10: The reconstructed gradient teapot model captured with a circular placement pattern at 1080p resolution with 216 images.

```
Phase-2-Checkerboard-Tests
├── 3840x2160
│   ├── Square Placement Pattern
│   │   ├── 12 Images
│   │   ├── 24 Images
│   │   ├── 36 Images
│   │   └── 48 Images
│   └── Circle Placement Pattern
│       ├── 12 Images
│       ├── 24 Images
│       ├── 36 Images
│       ├── 48 Images
│       ├── 108 Images
│       └── 216 Images
└── 1920x1080
    └── Circle Placement Pattern
        ├── 54 Images
        ├── 108 Images
        └── 216 Images
```

Figure 5.11: Structure of phase 2 teapot checkerboard texture experiments.

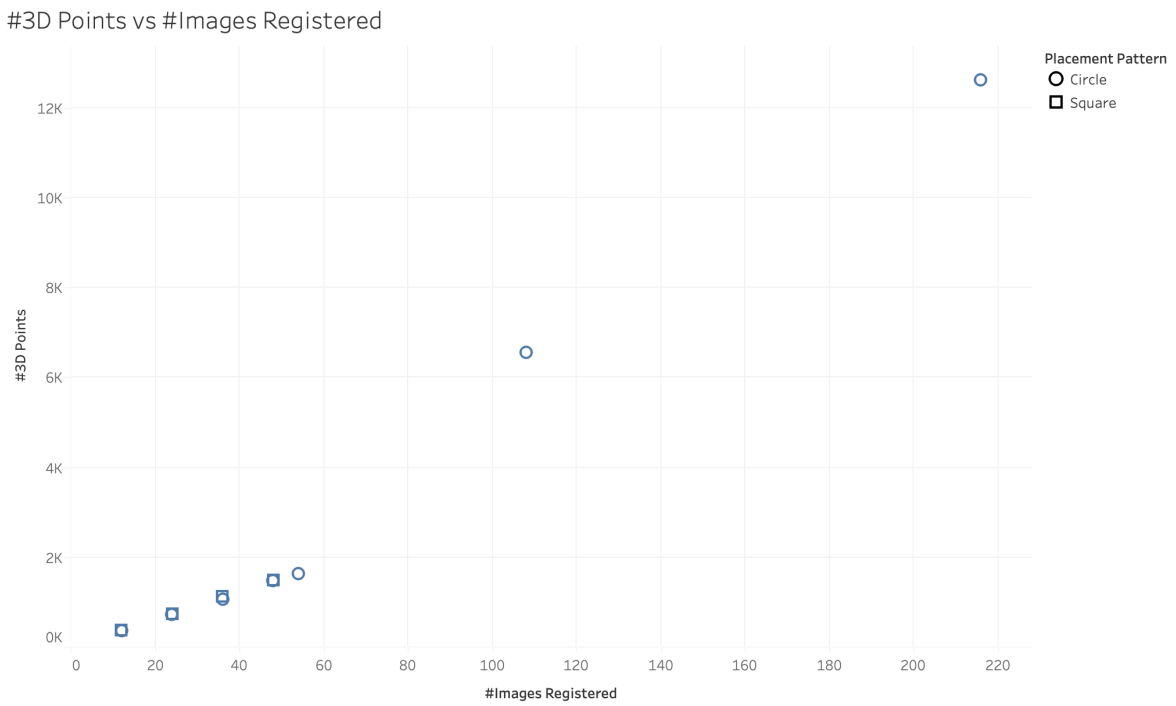

Figure 5.12: Graph plotting number of 3D points reprojected vs the number of images registered for the phase 2 checkerboard teapot experiments.

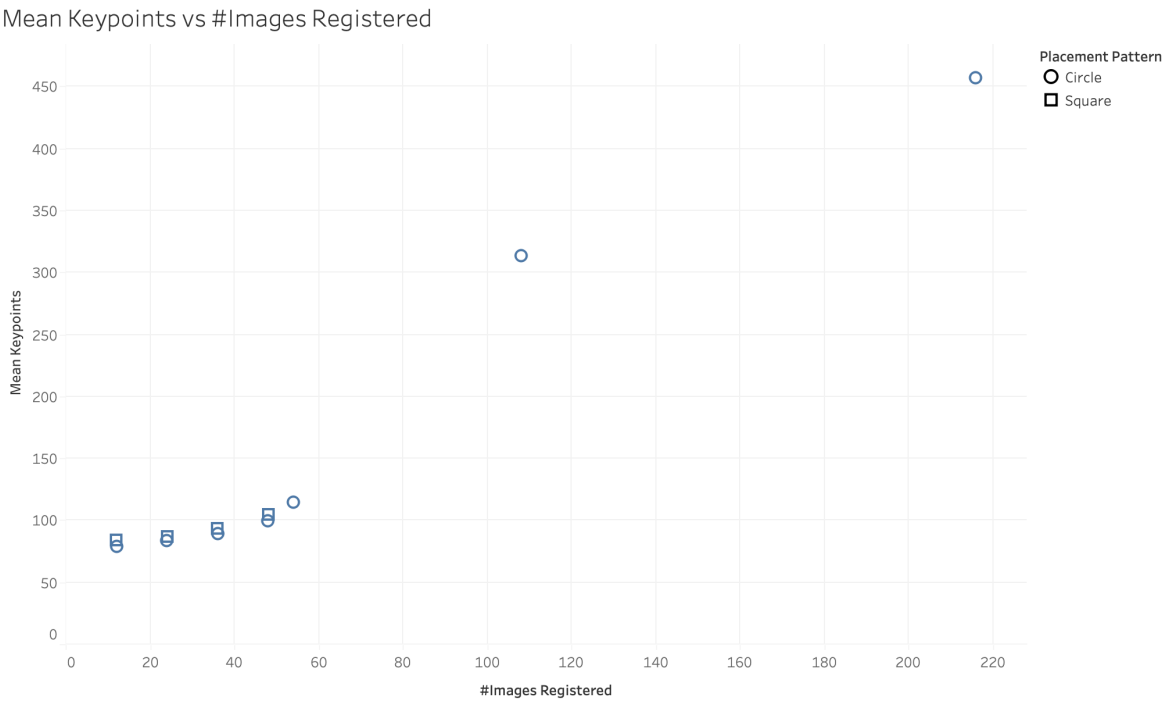

Figure 5.13: Graph plotting mean number of keypoints detected vs the number of images registered for the phase 2 checkerboard teapot experiments.

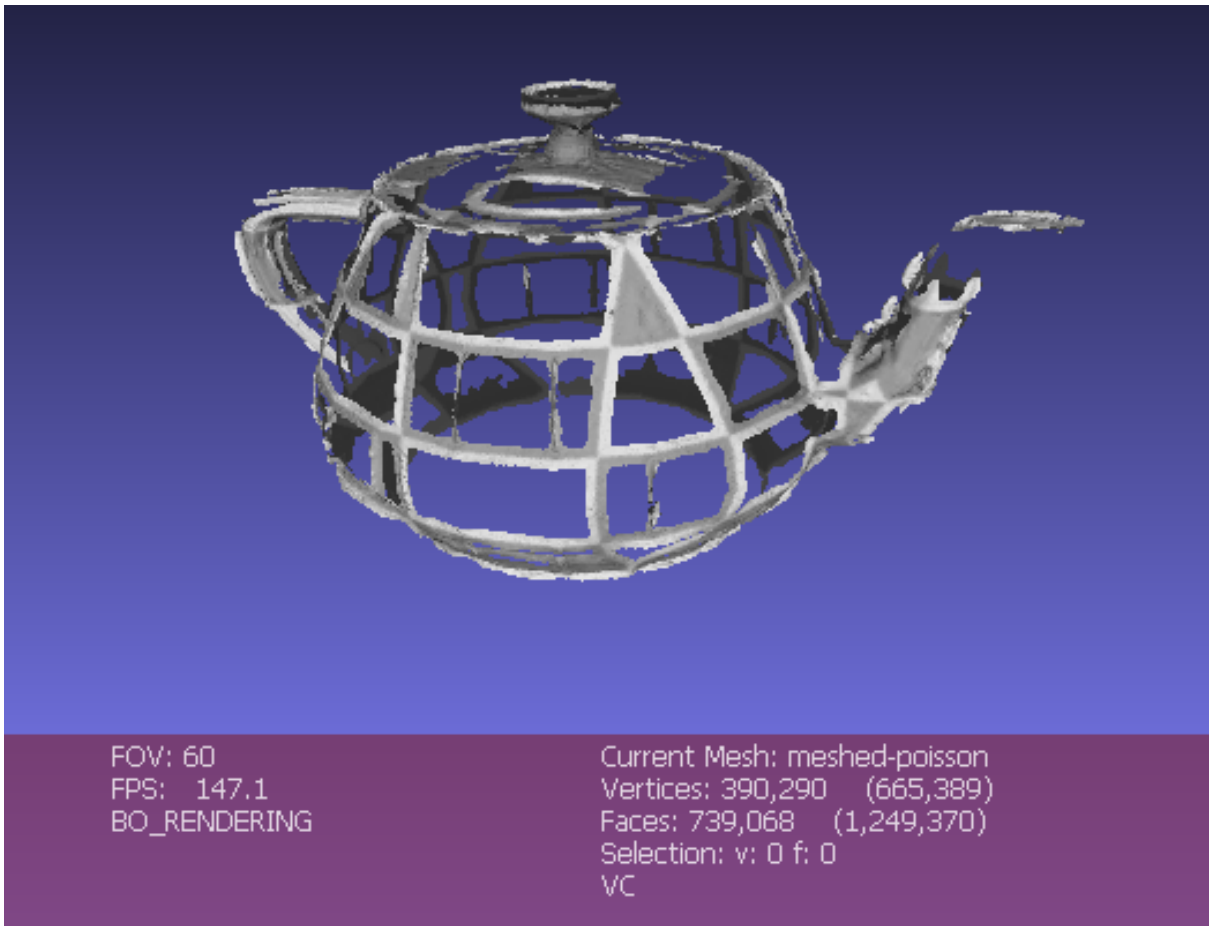

Figure 5.14: The reconstructed checkerboard teapot model captured with a circular placement pattern at 4k resolution with 216 images.

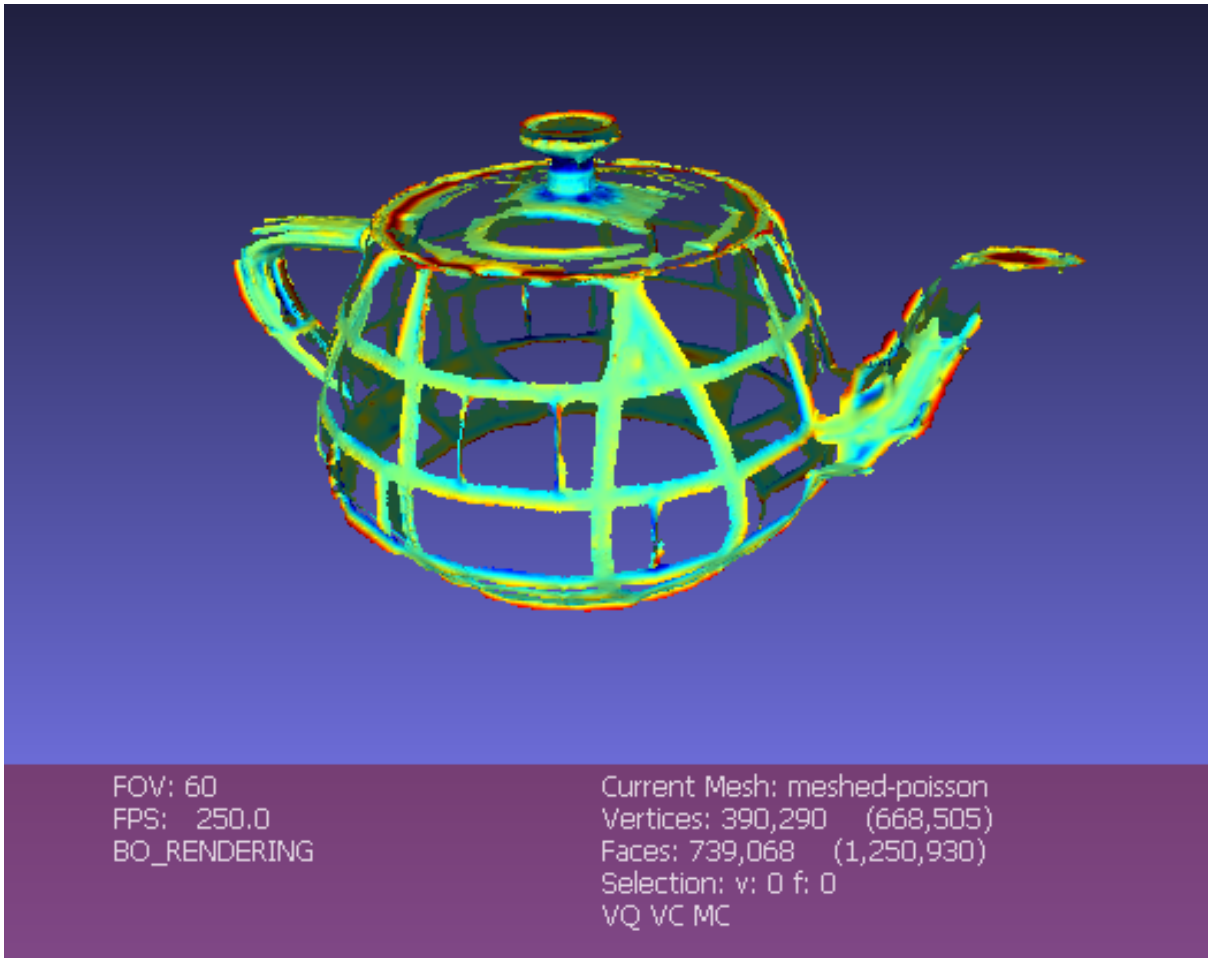

Figure 5.15: A heatmap comparing the reconstructed checkerboard teapot model captured with a circular placement pattern at 4k resolution with 216 images to the ground truth.

5.3 Phase 3

5.3.1 Model and Scene Configuration

In this phase of experiments, the impact of object geometry and lighting was investigated. Moving from the previous phase of testing, the scene configuration was refined for these experiments. Up until this point, the majority of reconstructions were mostly incomplete, with large amounts of geometry missing from the final mesh. From this analysis, I have established that both object symmetry and texturing play roles in this outcome. At this point, it was thought that increasing the amount of realism within the scene and the complexity of the model could benefit reconstruction results. To achieve this, the green skybox was replaced with a white one in order to better model the sky, which would be present on an overcast day in real life. This makes the ambient lighting conditions used in these tests, mimic real life conditions. To alter the geometry, the teapot model was replaced with a medieval wall model. The medieval wall model adds more complexity to the scene due to its small geometric surface details and texturing. Having changed these factors about the scene, it becomes of interest to determine if previous results about the reconstruction behaviour hold true.

Both Evee and Cycles were used to alter lighting conditions within the scene for these experiments. Coming from the previous results, the expectation is that Cycles will outperform Evee due to more surface geometry being visible thanks to raytracing. Figure 4.8 shows an example of the medieval wall scene configuration rendered using Cycles. The scene once again will be captured with varying camera placement patterns and resolutions. Figures 5.16 and 5.17 show the experiments undertaken at 4k and 1080p resolutions respectively. The SfM results obtained from this testing is included as appendices A1.1.6 and A1.1.7.

5.3.2 Impact of Geometry and Lighting

SfM Results Analysis

The increased amount of texture and geometric detail present in the medieval wall model resulted in increased keypoint detection across all tests. We can see in graph 5.18 that on average these figures increase by a factor of 4 when compared with previous tests. Further querying these results, we also find significant improvements in the number of 3D keypoints identified as seen in graph 5.19. Once again we see a linear relationship between the mean keypoints detected and the number of images captured, regardless of placement pattern. With the same holding true for the relationship between the number of 3D points triangulated and the number of images captured.

In these tests, we see a larger separation between the square and circle placement patterns, especially as the number of images captured increases. This behaviour is also exhibited when the image resolution drops from 4k to 1080p, as can be seen in graphs 5.21 and 5.22. The results observed from this phase of testing further solidifies the performance improvement possible through the use of a circular camera placement pattern. Figure 5.24 shows a heatmap of the differences between square and circle placement patterns, with areas in red indicating surfaces missing from the final mesh when a square placement pattern is used.

Drilling more into the data, I wanted to see what conclusions could be drawn from the mean track length and its relationship to the number of 3D points reprojected. The track length defines through how many images a keypoint can be successfully matched and thus tracked. From the literature discussed in section 2, it is indicated that a longer track length suggests better matches within the dataset and can be viewed as an additional indicator of image-set quality. Looking at graph 5.20, we can see a linear relationship between the number of 3D points reprojected and the average track length. This relationship holds true for the 1080p tests performed, albeit with higher variance as seen in graph 5.23. This relationship confirms that track length can be useful as a quality measure for this capture scenario. Much like phase 2 testing, we see that the number of images captured has only a positive impact on overall SfM results. From these factors combined, it can be concluded that for this capture scenario, a higher number of evenly distributed images captured around an object of interest in a circular pattern gives the best possible reconstruction results.

Render Engine

To evaluate the impact of the render engine, we turn our attention towards visual inspection and the Hausdorff distance results. Compared to previous phases, this phase of testing produced much more complete models across the board. Figures 5.25, 5.26 and 5.27 show an examples of a final mesh produced by an image-set rendered using the Evee engine. This final reconstruction successfully captures most of the geometric detail of the model, with all surfaces being meshed to a satisfactory level. However, beyond the basic structure, much of the complexities of the model were not reconstructed. The finer surface geometry, such as the concavities around each brick are missing. The surface texture is of a lesser quality when compared to the original, with the reconstruction appearing quite smooth. Heatmap 5.28 shows these differing areas when compared to the ground truth. I also note the continued presence of fringing around the edge of the model coming from the background.

When compared with the results obtained using the Cycles render engine, we can immediately see improved performance. Figure 5.29 shows the same capture environment using the Cycles render engine. Immediately, we can observe much more of the surface texture being retained in the reconstruction. Additionally, closer inspection reveals much of

the surface geometry missing from the Evee reconstruction to now be present. To investigate how well the reconstructions from the Cycles render engine perform, additional tests were conducted with a larger image-set of 192 images. This increase in image-set size had the impact of visually improving mesh coverage in some areas, such as the tower as seen in figures 5.30 and 5.31.

Calculating the Hausdorff distance between the final Cycles reconstruction and ground truth, brought some interesting observations to light. Captured from the front of the model, heatmap 5.32 shows the majority of the surface geometry to be reconstructed accurately, indicated by the yellow and green areas. On the negative side, we see quite a significant amount of red regions on the edges of the model. This again is due to poor segmentation with the background, further establishing this factor as an area of difficulty for this type of reconstruction.

Decreasing the capture resolution from 4k to 1080p resulted in a reduced number of vertices and faces reconstructed as observed in previous tests. However, these 1080p reconstructions still exhibit the same characteristics as the 4k reconstructions. The observed quality delta between these resolutions has closed considerably compared with reconstructions in previous phases. The differences between figures 5.33 and 5.29 demonstrate this observation.

5.3.3 Observations and Conclusions

From this phase of experimentation, it was found that the use of more complex geometry did indeed improve upon previous reconstruction results. This increase in complexity had the direct impact of improving keypoint detection and tracking. Overall, this improved the amount of scene geometry successfully reconstructed. This enabled the quality delta between capture resolutions to be closed significantly, with 1080p capture yielding impressive results. Once again, increasing the image-set was shown to be beneficial, without any downside to final reconstruction quality. Using photo-realistic rendering improved upon the fidelity of the final reconstruction results, improving texturing and the appearance of geometry. However, despite these improvements it was found that background segmentation remains a challenge, with lower resolutions suffering more.

```
Phase-3-Medieval-Wall-Tests
└── 3840x2160
    ├── Cycles
    │   ├── Square Placement Pattern
    │   │   ├── 12 Images
    │   │   ├── 24 Images
    │   │   ├── 36 Images
    │   │   ├── 48 Images
    │   │   ├── 96 Images
    │   │   └── 192 Images
    │   └── Circle Placement Pattern
    │       ├── 12 Images
    │       ├── 24 Images
    │       ├── 36 Images
    │       ├── 48 Images
    │       ├── 96 Images
    │       └── 192 Images
    └── Evee
        ├── Square Placement Pattern
        │   ├── 12 Images
        │   ├── 24 Images
        │   ├── 36 Images
        │   └── 48 Images
        └── Circle Placement Pattern
            ├── 12 Images
            ├── 24 Images
            ├── 36 Images
            └── 48 Images
```

Figure 5.16: Structure of phase 3 4k medieval wall experiments.

```
Phase-3-Medieval-Wall-Tests
└── 1920x1080
    ├── Cycles
    │   ├── Square Placement Pattern
    │   │   ├── 12 Images
    │   │   ├── 24 Images
    │   │   ├── 36 Images
    │   │   └── 48 Images
    │   └── Circle Placement Pattern
    │       ├── 12 Images
    │       ├── 24 Images
    │       ├── 36 Images
    │       └── 48 Images
    └── Evee
        ├── Square Placement Pattern
        │   ├── 12 Images
        │   ├── 24 Images
        │   ├── 36 Images
        │   └── 48 Images
        └── Circle Placement Pattern
            ├── 12 Images
            ├── 24 Images
            ├── 36 Images
            └── 48 Images
```

Figure 5.17: Structure of phase 3 1080p medieval wall experiments.

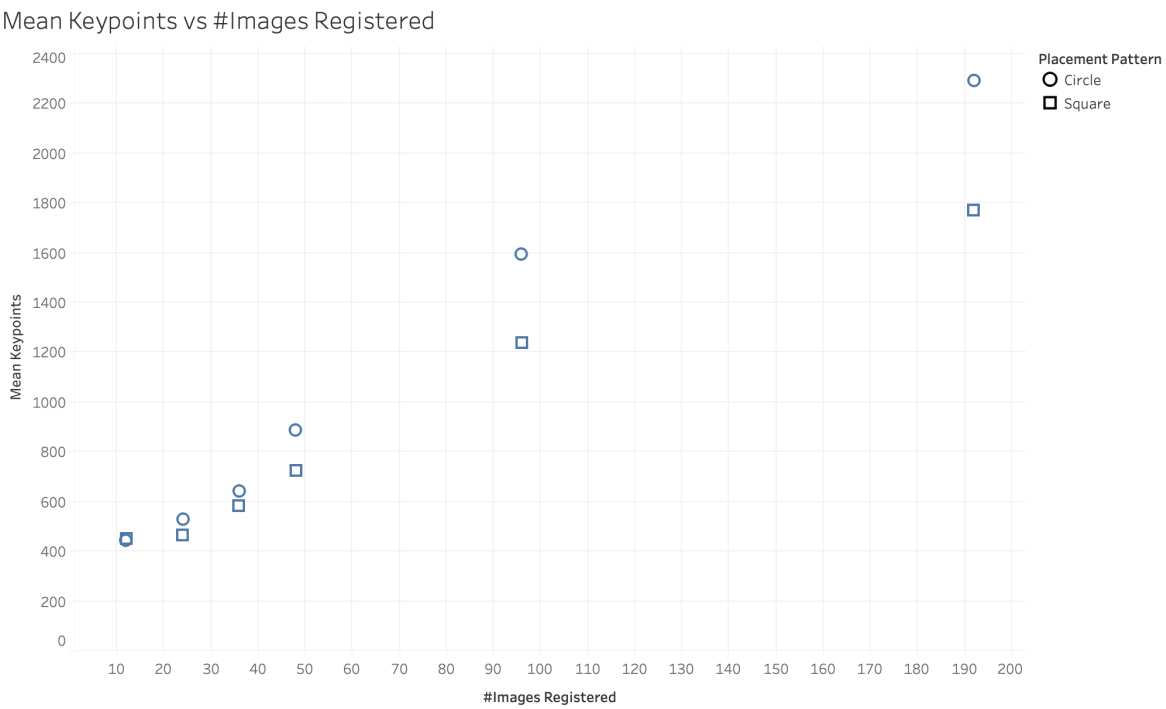

Figure 5.18: Graph plotting mean number of keypoints detected vs the number of images registered for the phase 3 4k medieval wall experiments.

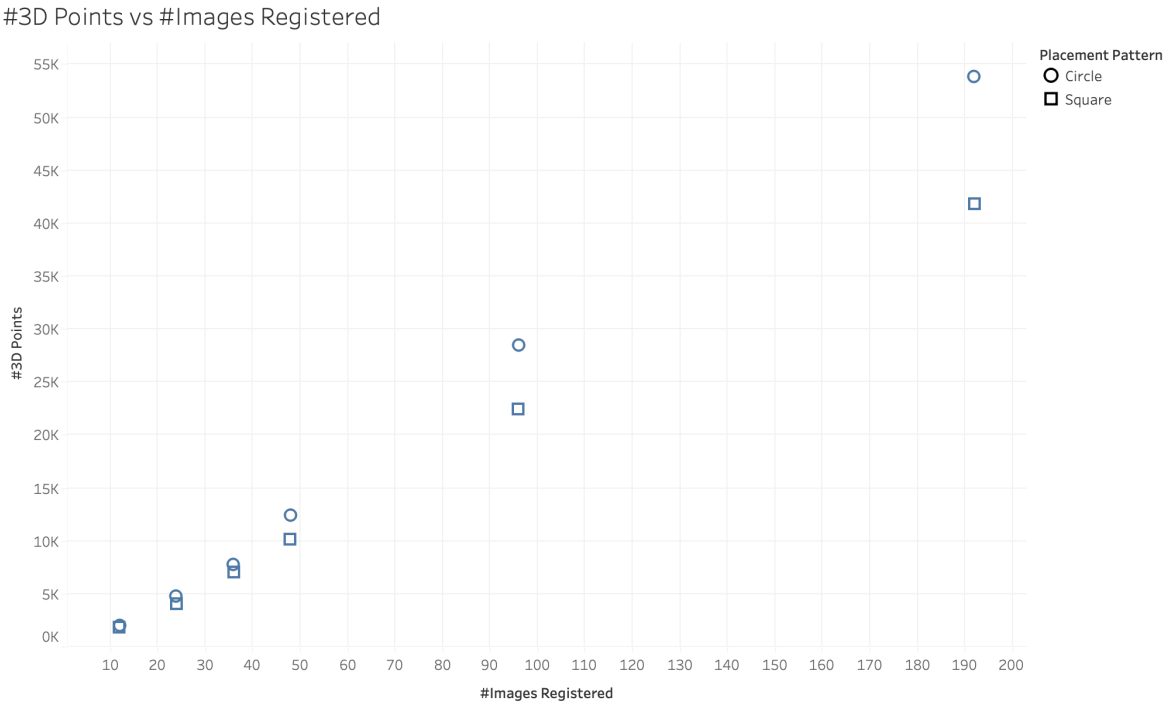

Figure 5.19: Graph plotting number of 3D points reprojected vs the number of images registered for the phase 3 4k medieval wall experiments.

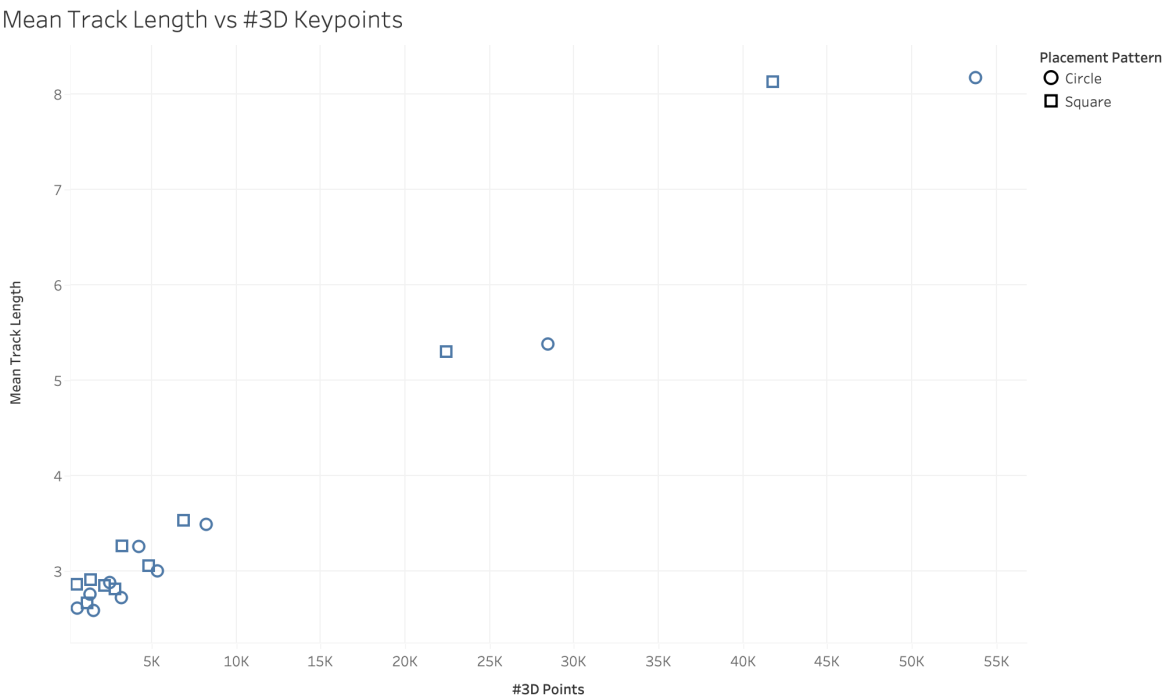

Figure 5.20: Graph plotting mean track length vs the number of 3D points reprojected for the phase 3 4k medieval wall experiments.

Figure 5.21: Graph plotting mean number of keypoints detected vs the number of images registered for the phase 3 1080p medieval wall experiments.

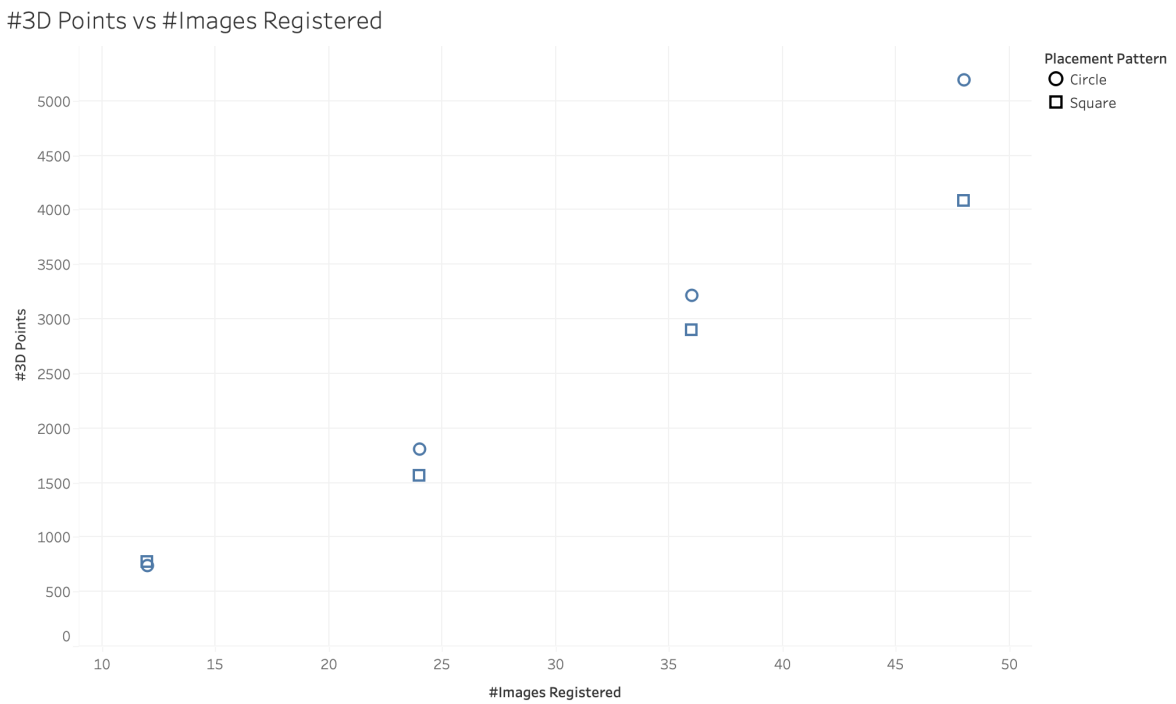

Figure 5.22: Graph plotting number of 3D points reprojected vs the number of images registered for the phase 3 1080p medieval wall experiments.

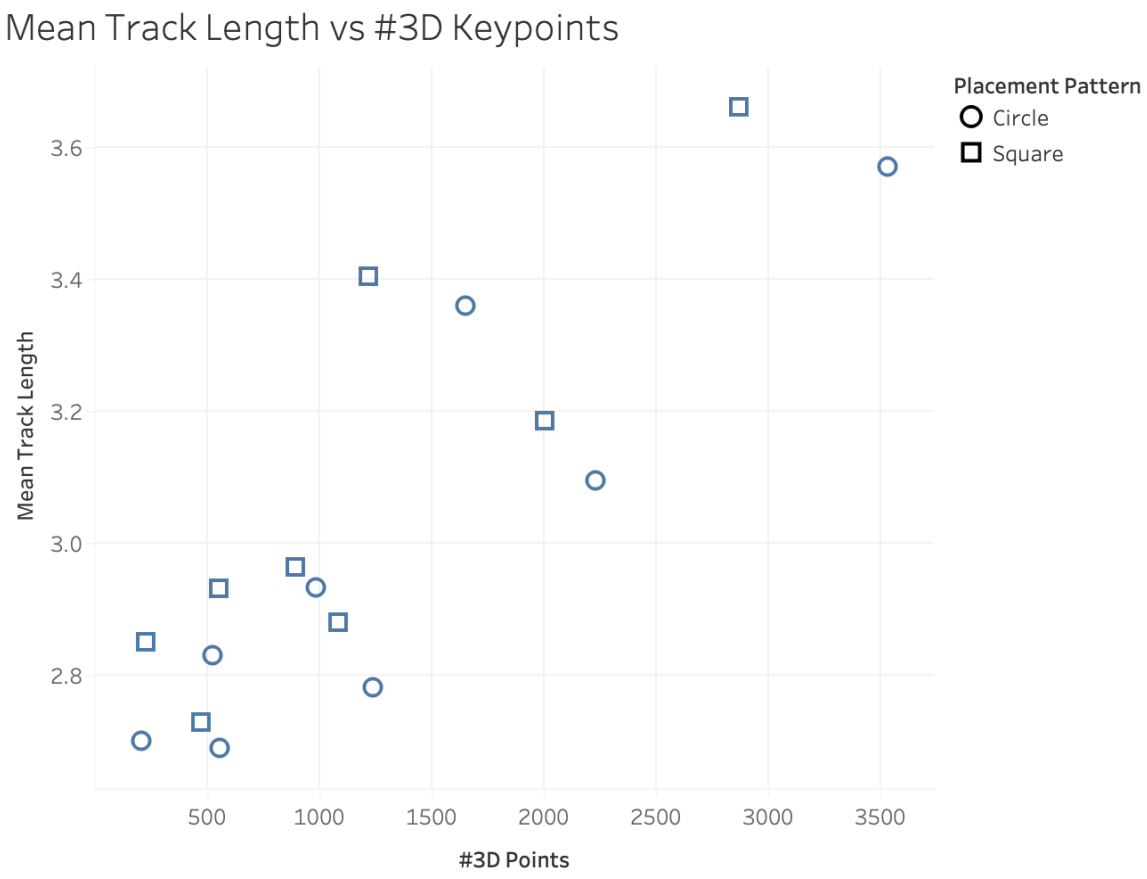

Figure 5.23: Graph plotting mean track length vs the number of 3D points reprojected for the phase 3 1080p medieval wall experiments.

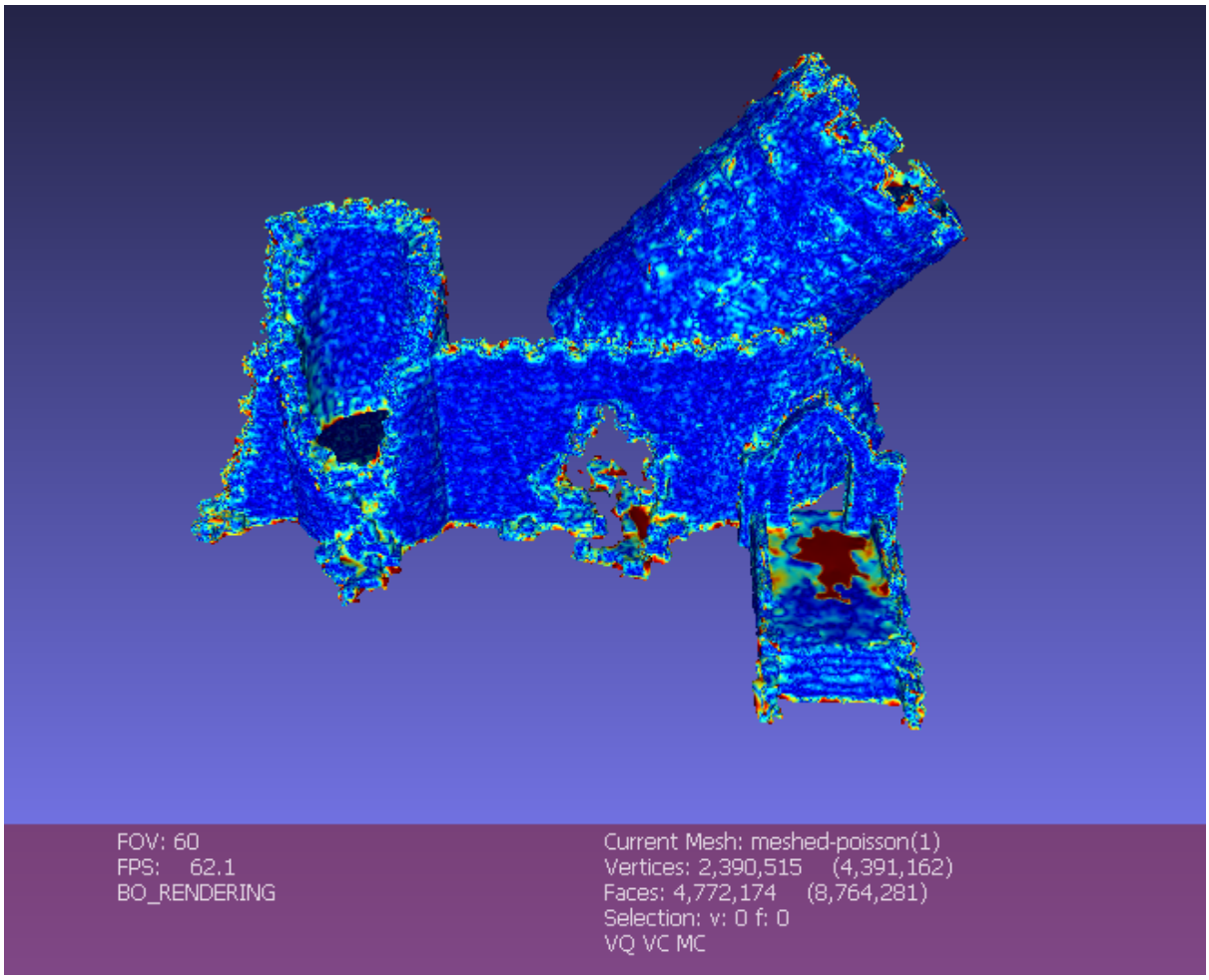

Figure 5.24: Heatmap generated using the Hausdorff distance measure to compare circle and square placement patterns for a Cycles 192 4k image reconstruction of the medieval wall model.

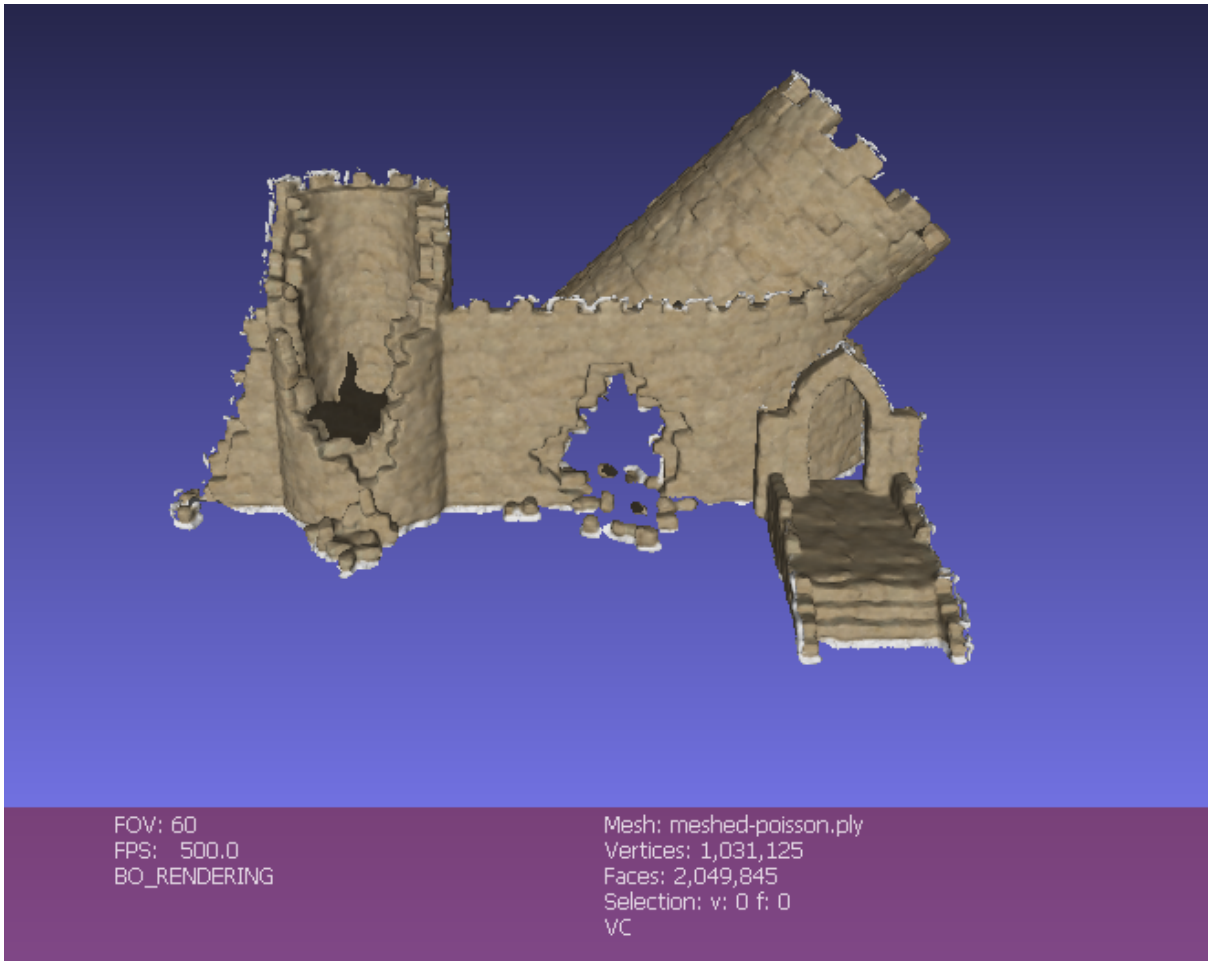

Figure 5.25: Front on view of an Evee 48 4k image reconstruction of the medieval wall model.

Figure 5.26: Top down view of an Evee 48 4k image reconstruction of the medieval wall model.

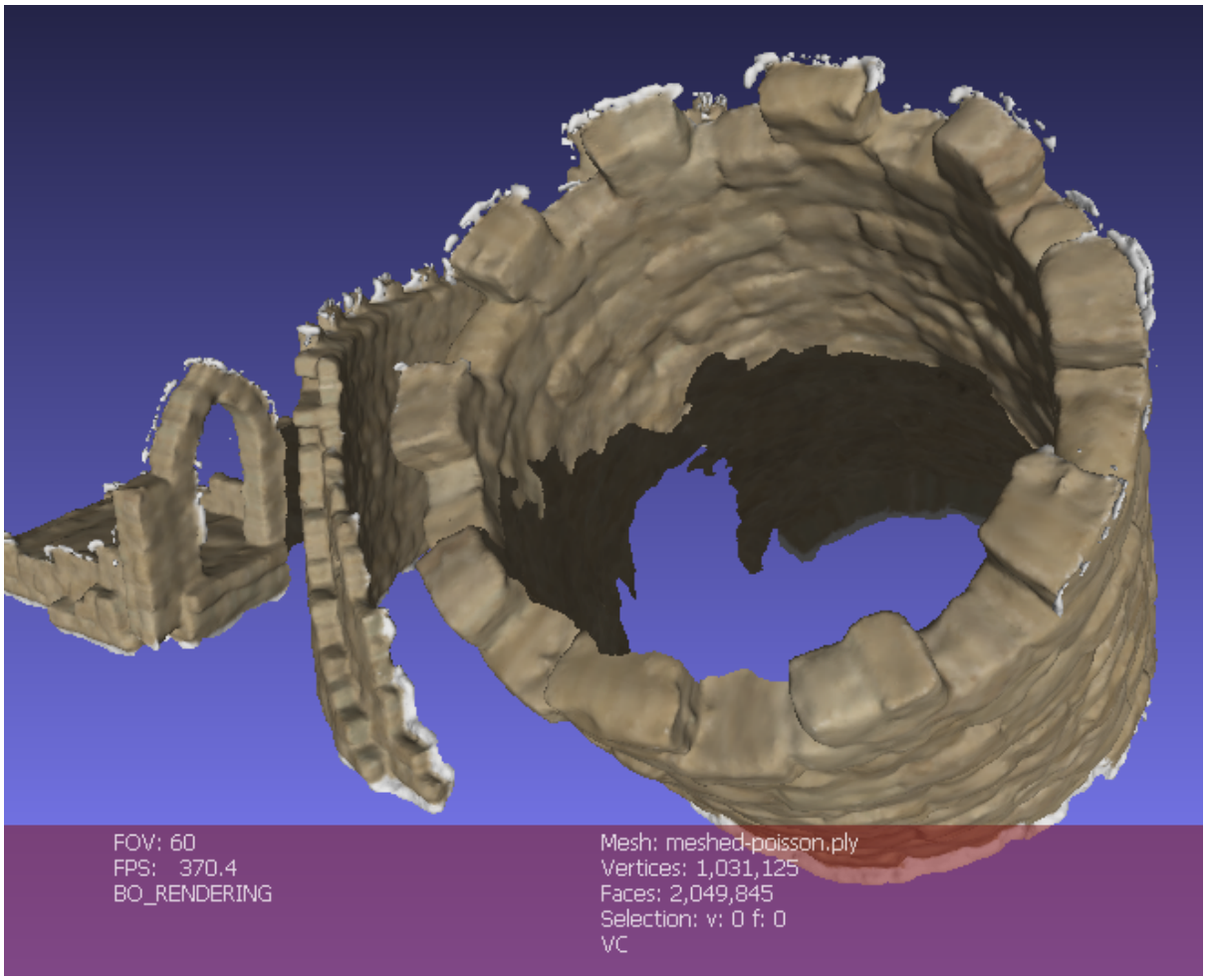

Figure 5.27: A view of an Evee 48 4k image reconstruction of the medieval wall model focusing on the meshing in the sideways tower.

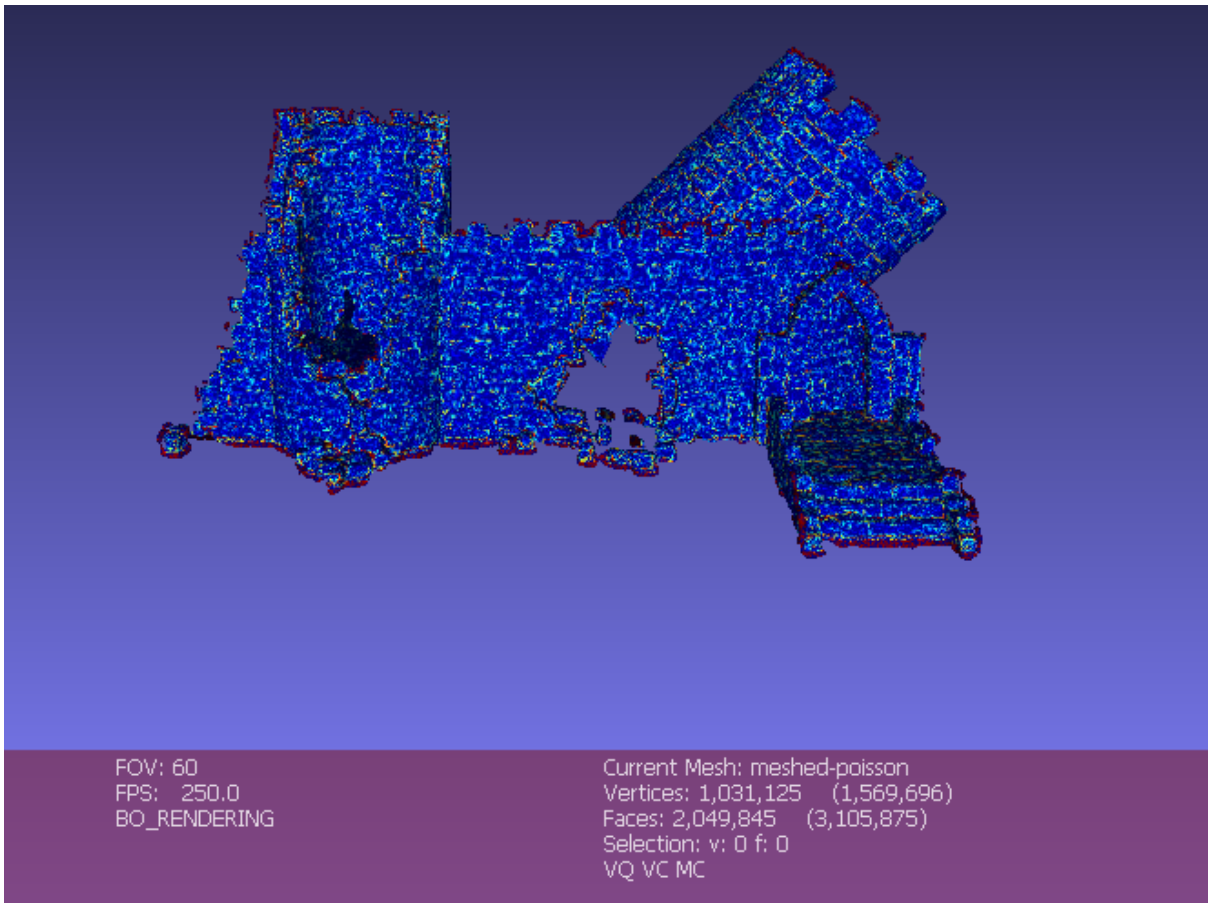

Figure 5.28: Heatmap generated using the Hausdorff distance measure to compare an Evee 48 4k image reconstruction of the medieval wall model to the ground truth model.

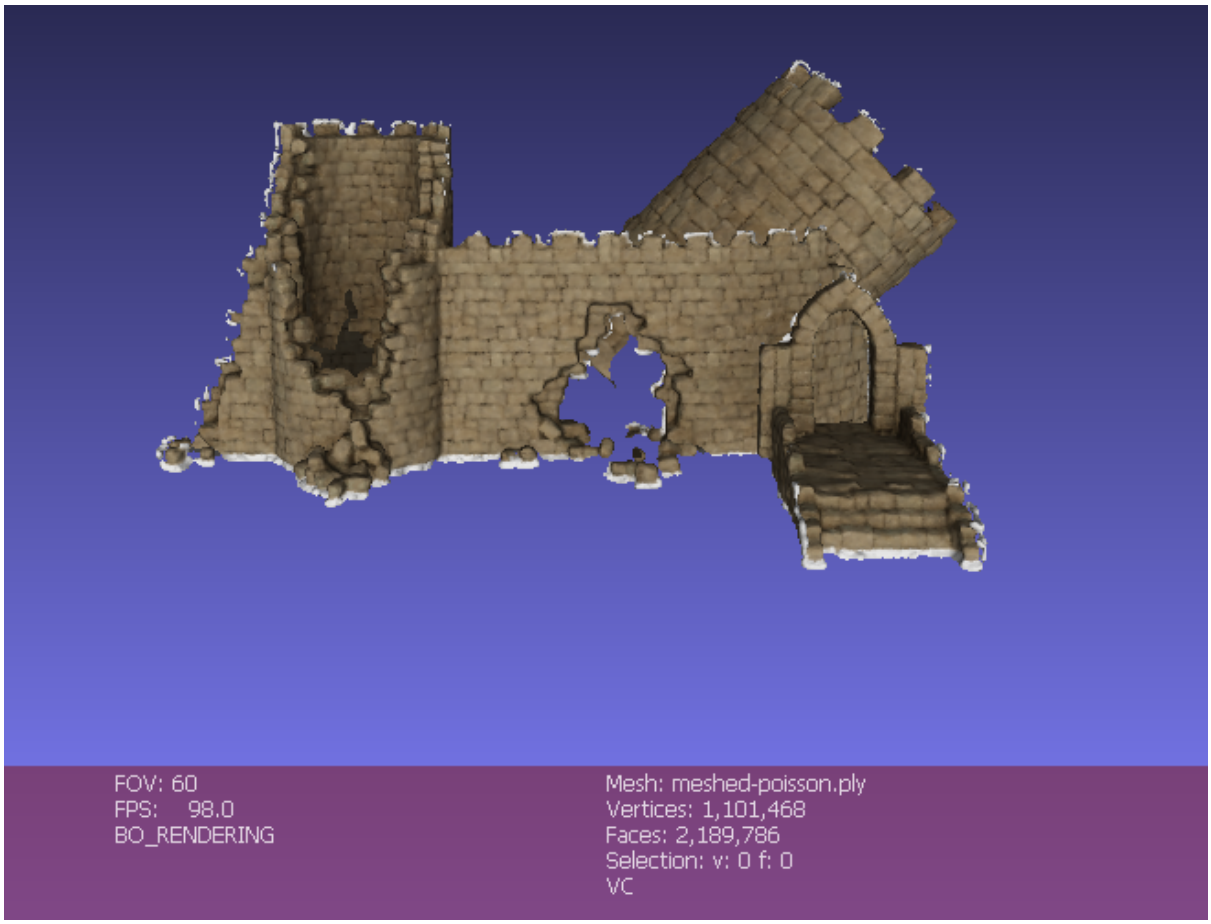

Figure 5.29: Front on view of a Cycles 48 4k image reconstruction of the medieval wall model.

Figure 5.30: Front on view of a Cycles 192 4k image reconstruction of the medieval wall model.

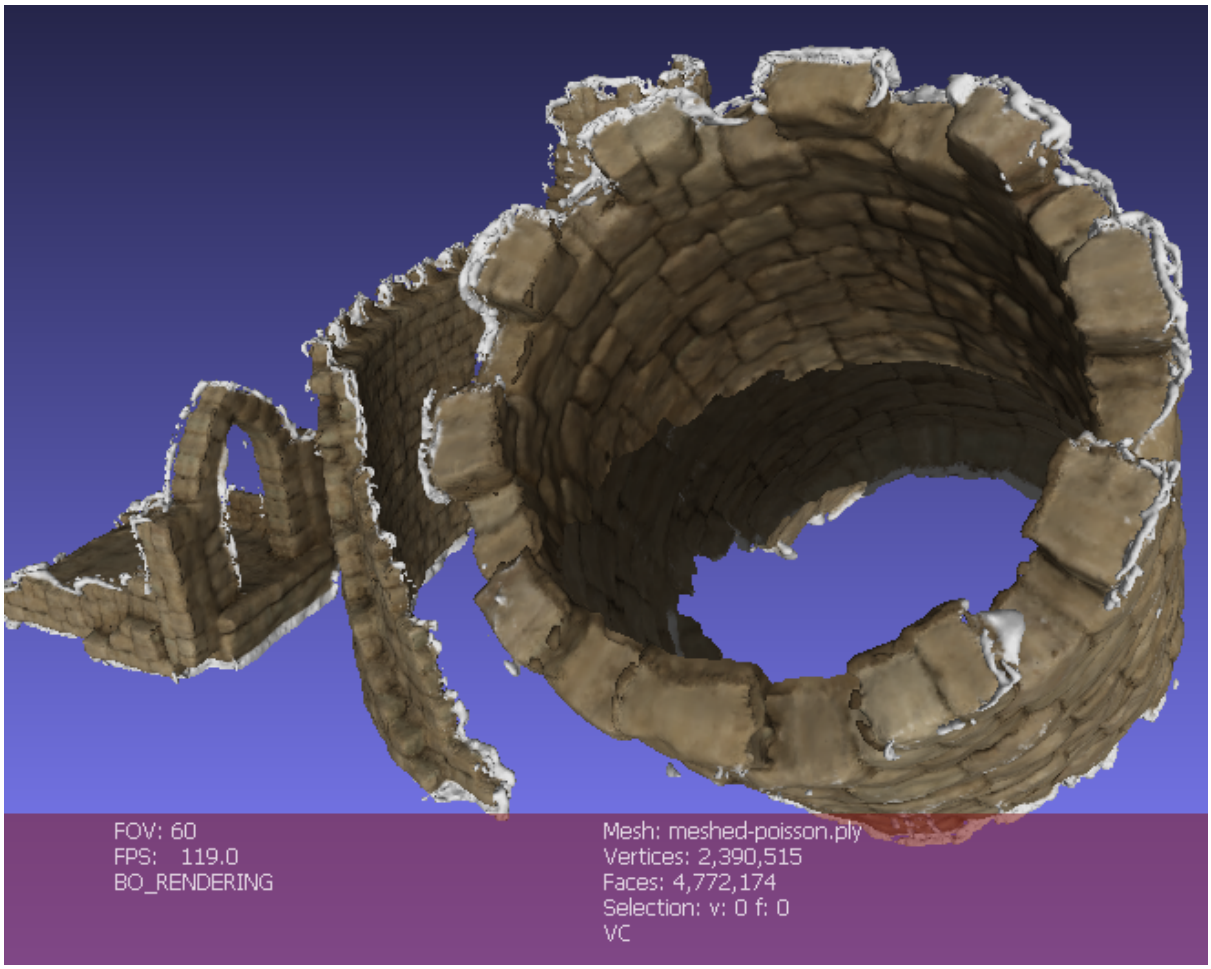

Figure 5.31: A view of a Cycles 192 4k image reconstruction of the medieval wall model focusing on the meshing in the sideways tower.

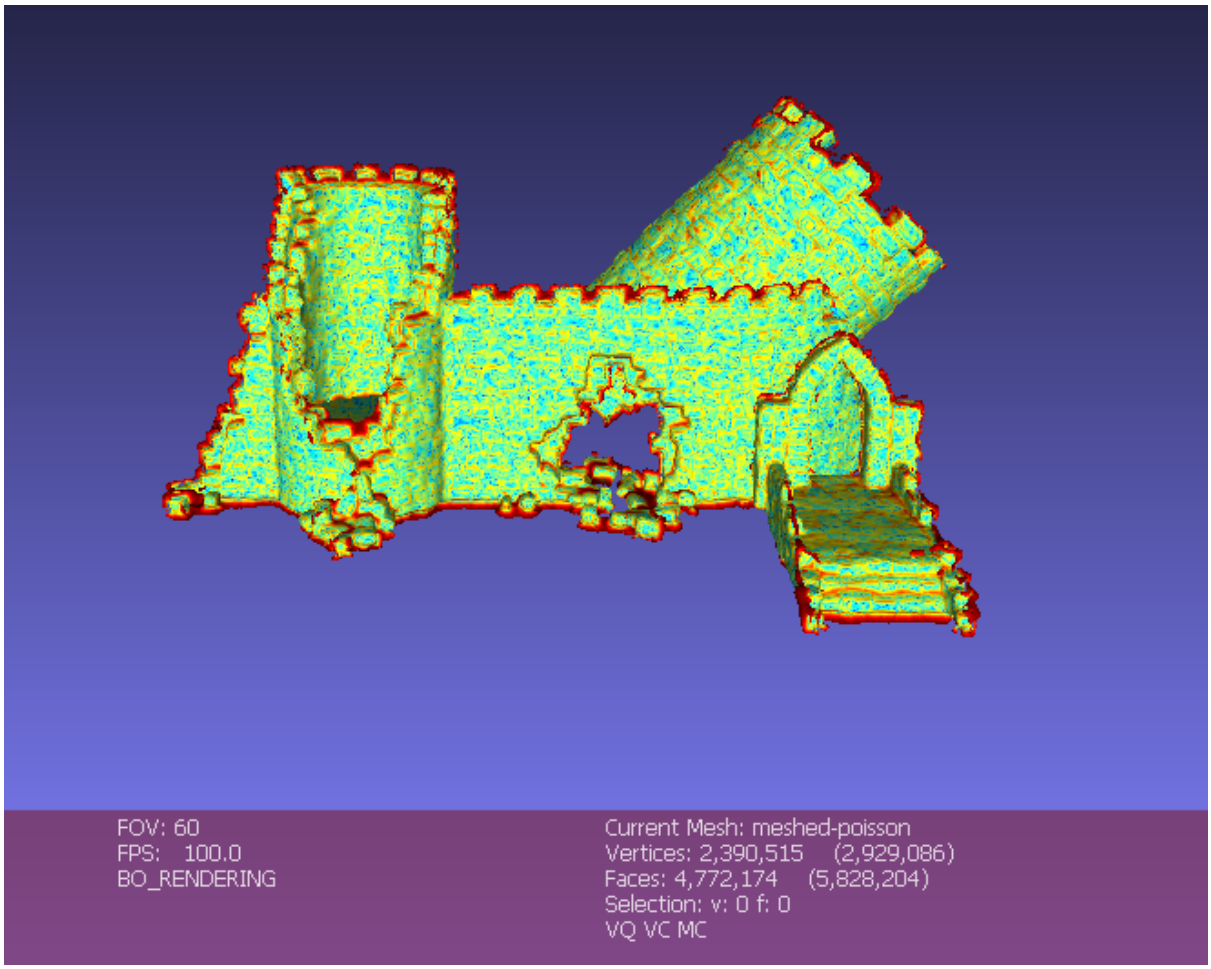

Figure 5.32: Heatmap generated using the Hausdorff distance measure to compare a Cycles 192 4k image reconstruction of the medieval wall model to the ground truth model.

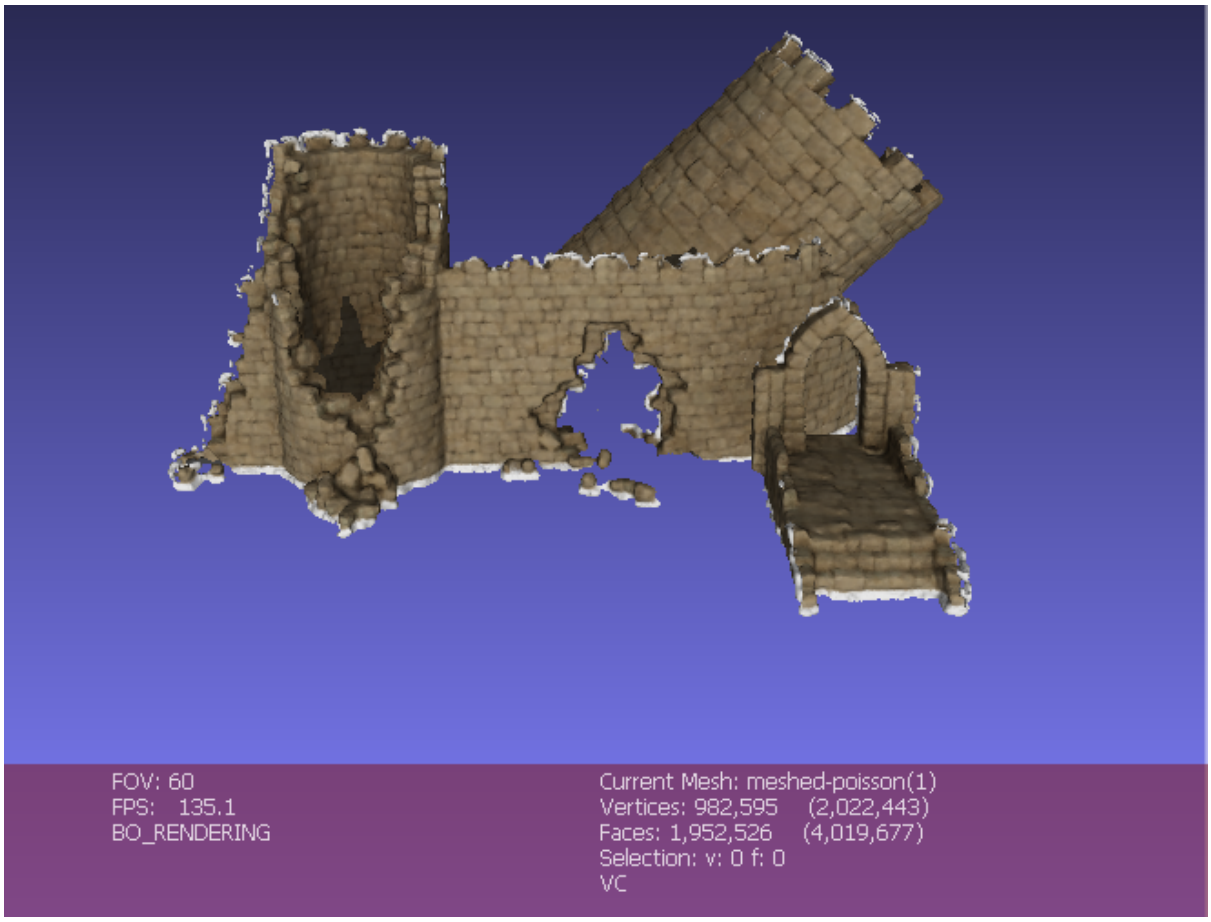

Figure 5.33: Front on view of a Cycles 48 1080p image reconstruction of the medieval wall model.

5.4 Phase 4

Through each phase of experimentation, I have refined and improved upon the reconstruction results by utilising each point of new knowledge obtained. From this testing, I have shown that increasing the complexity of the geometry in the scene, improving the texturing and ensuring good lighting, are all measurable quantities that impact scene quality. However, while utilising these findings has yielded improved reconstruction results, they do not entirely mimic the behaviour discussed in other literature. As has been shown with the use of more complex geometry, this may be due to the level of detail in the scene. Most reconstructions performed in the literature are of urban environments. These environments have no shortage of background details, from lamp posts, to cars, to buildings. As such, it was thought the final phase of testing could be performed with a more realistic scene setup, moving outside of the controlled capture studio environment and closer to an urban setting.

5.4.1 Model and Scene Configuration

In phase 2, it was established that the gradient texture applied to the teapot model was unable to produce a complete mesh when captured and reconstructed. For the phase 4 tests, the gradient teapot model was placed in a cityscape scene, once again as the focus point. Rather than just a homogeneous white background, this time images of the teapot feature extensive background details, as can be seen in figure 5.37. It was hoped that this increase in background detail should help feature detection, feature tracking and ultimately meshing. Even if the keypoints are not detected on the teapot directly, it may be possible to better reconstruct it when other geometry is present.

In addition to the cityscape model, the lighting conditions were altered to include a simulated sun light source. This cast shadows around the scene when using the Cycles render engine, as seen in figure 5.38. The skybox was updated to a blue colour and included white clouds, again increasing the level of realism within the scene. A preview of the final scene configuration can be viewed in figure 5.36.

5.4.2 Impact of Scene Background Detail

SfM Results Analysis

Upon analysis of the cityscape SfM results found in tables A1.1.10 and A1.1.11, we find a similar amount of keypoints as previously observed with the medieval wall reconstructions. The data shows the same trends as phase 2 testing, with the square placement pattern outperforming the circle placement pattern. We consistently see a significant delta increase in the number of keypoints identified and the number of 3D points triangulated using the square placement pattern for these tests, as can be seen in graphs 5.39 and 5.40. A visual

inspection of the results obtained show a more complete mesh when using a square placement pattern. Figures 5.41 and 5.42 show the meshes produced by the square and circle placement patterns, respectively. These tests once again form a linear relationship between the number of keypoints detected and the number of images, and the number of 3D points triangulated and the number of images. However, these trends are not as strong as in previous tests, with a weaker correlation between mean track length and 3D keypoints as seen in graph 5.43. We even see a tapering off to the increase in track length as the number of 3D keypoints increases. This is an attribute which was not observed in previous experiments.

Render Engine

As I had hoped, these tests improved the meshing of the teapot, which now features a fully complete mesh. However, the geometric accuracy of this mesh is poor in many areas, with much of the geometry having exaggerated and slightly rounded appearances. While the mesh is full closed, it is less refined than in previous experiments. It seems that much of the reconstruction computation is being devoted to reconstructing the background detail, even though the teapot is in the foreground of the images. The impact of this is that neither the teapot nor the background details are particularly accurate.

When performing a further visual inspection, we see the Evee render engine produced a more accurate mesh when compared to Cycles. This can be attributed to the lack of shadows produced when rendering using Evee. These shadows present in the Cycles renders, decreased mesh accuracy as seen in figure 5.48. This creates large areas of additional meshing underneath the model, corresponding to the presence of a shadow. This effect is not seen on the Evee model in figure 5.50. It should also be noted that large portions of homogeneous texture such as the ground are missing from these reconstructions. Once again confirming the difficulty reconstructing textures of this nature. While the teapot model is more complete than in previous tests, general reconstruction geometry and texture is poor with this configuration. The background geometry is unrefined, with texturing being blurred but distinguishable. This makes sense as the teapot was taking up the majority of the frame in captured images, thus making it difficult to detect and track background details accurately. However, it is noted that the quality of the background structures is of less interest to us in this test as we are measuring the change to the teapot model primarily.

Dropping the resolution to 1080p resulted in the same visual characteristics as the 4k image-sets. This time there were very few noticeable differences to geometry or texturing detail, as can be seen in figure 5.51. It seems that the increase amount of background detail could still be captured at the lower quantisation level to great effect. A final point to note is the minimal fringing around the teapot model. It seems that having increased background

detail improved segmentation between the background and the model of interest. Further inspection of this behaviour brought me to the conclusion that having additional keypoints in the background helped in separating them from keypoints on the object of interest, thus reducing fringing.

5.4.3 Observations and Conclusions

From this phase of testing, it was found that an increase in background scene detail did indeed have an impact on the final reconstruction quality. We saw an increase in the number of keypoints identified, which ultimately had the effect of improved tracking and reconstruction of an otherwise difficult model. These tests did exhibit different behaviour to the previous tests, with the square placement pattern outperforming the circular pattern in both SfM and visual results. Finally, the true impact of lighting was observed in the final mesh. Areas of shadow not only lacked major keypoints, but also added superfluous area to the final mesh, reducing its accuracy. This result aligns with findings discussed in section 2, in which overcast weather is recommended for optimal scene capture.

```
Phase-4-Gradient-Cityscape-Tests
└── 3840x2160
    ├── Cycles
    │   ├── Square Placement Pattern
    │   │   ├── 12 Images
    │   │   ├── 24 Images
    │   │   ├── 36 Images
    │   │   └── 48 Images
    │   └── Circle Placement Pattern
    │       ├── 12 Images
    │       ├── 24 Images
    │       ├── 36 Images
    │       └── 48 Images
    └── Evee
        ├── Square Placement Pattern
        │   ├── 12 Images
        │   ├── 24 Images
        │   ├── 36 Images
        │   └── 48 Images
        └── Circle Placement Pattern
            ├── 12 Images
            ├── 24 Images
            ├── 36 Images
            └── 48 Images
```

Figure 5.34: Structure of phase 4 4k teapot checkerboard texture in the cityscape experiments.

```
Phase-4-Gradient-Cityscape-Tests
└── 1920x1080
    ├── Cycles
    │   ├── Square Placement Pattern
    │   │   ├── 12 Images
    │   │   ├── 24 Images
    │   │   ├── 36 Images
    │   │   └── 48 Images
    │   └── Circle Placement Pattern
    │       ├── 12 Images
    │       ├── 24 Images
    │       ├── 36 Images
    │       └── 48 Images
    └── Evee
        ├── Square Placement Pattern
        │   ├── 12 Images
        │   ├── 24 Images
        │   ├── 36 Images
        │   └── 48 Images
        └── Circle Placement Pattern
            ├── 12 Images
            ├── 24 Images
            ├── 36 Images
            └── 48 Images
```

Figure 5.35: Structure of phase 4 1080p teapot checkerboard texture in the cityscape experiments.

Figure 5.36: Preview of the final cityscape scene configuration featuring the gradient teapot model.

Figure 5.37: An image captured using the Evee render engine of the gradient teapot in a cityscape scene.

Figure 5.38: An image captured using the Cycles render engine of the gradient teapot in a cityscape scene.

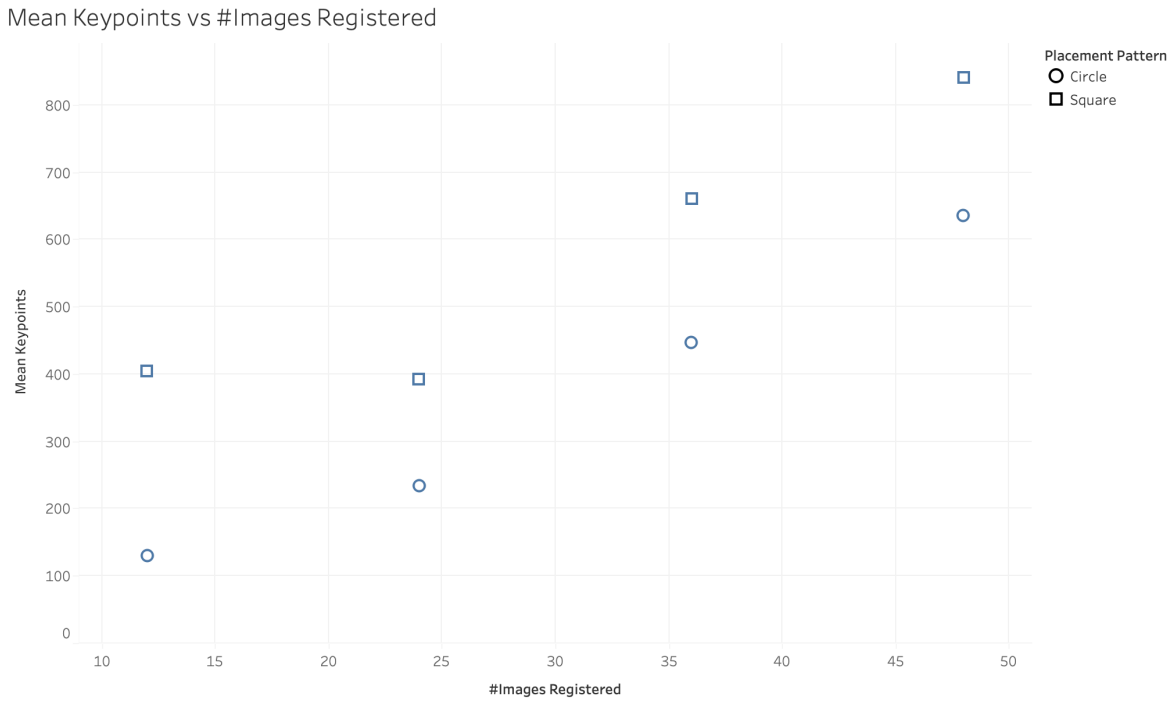

Figure 5.39: Graph plotting mean number of keypoints detected vs the number of images registered for the phase 4 4k gradient teapot cityscape experiments.

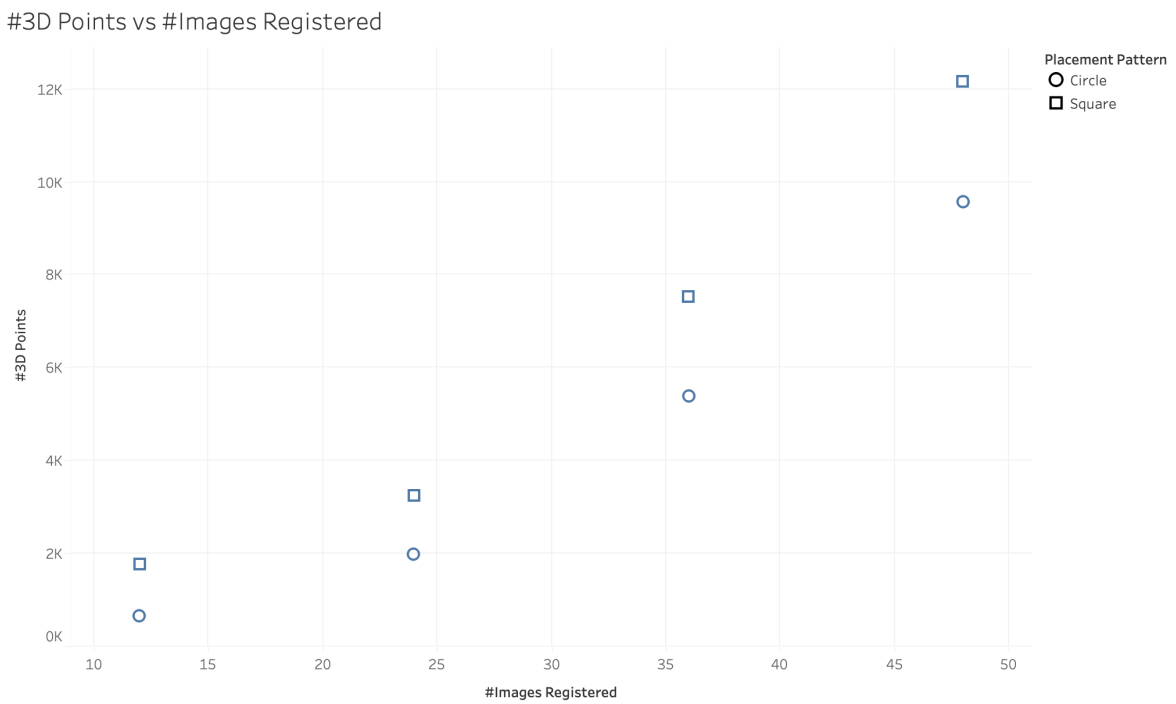

Figure 5.40: Graph plotting number of 3D points reprojected vs the number of images registered for the phase 4 4k gradient teapot cityscape experiments.

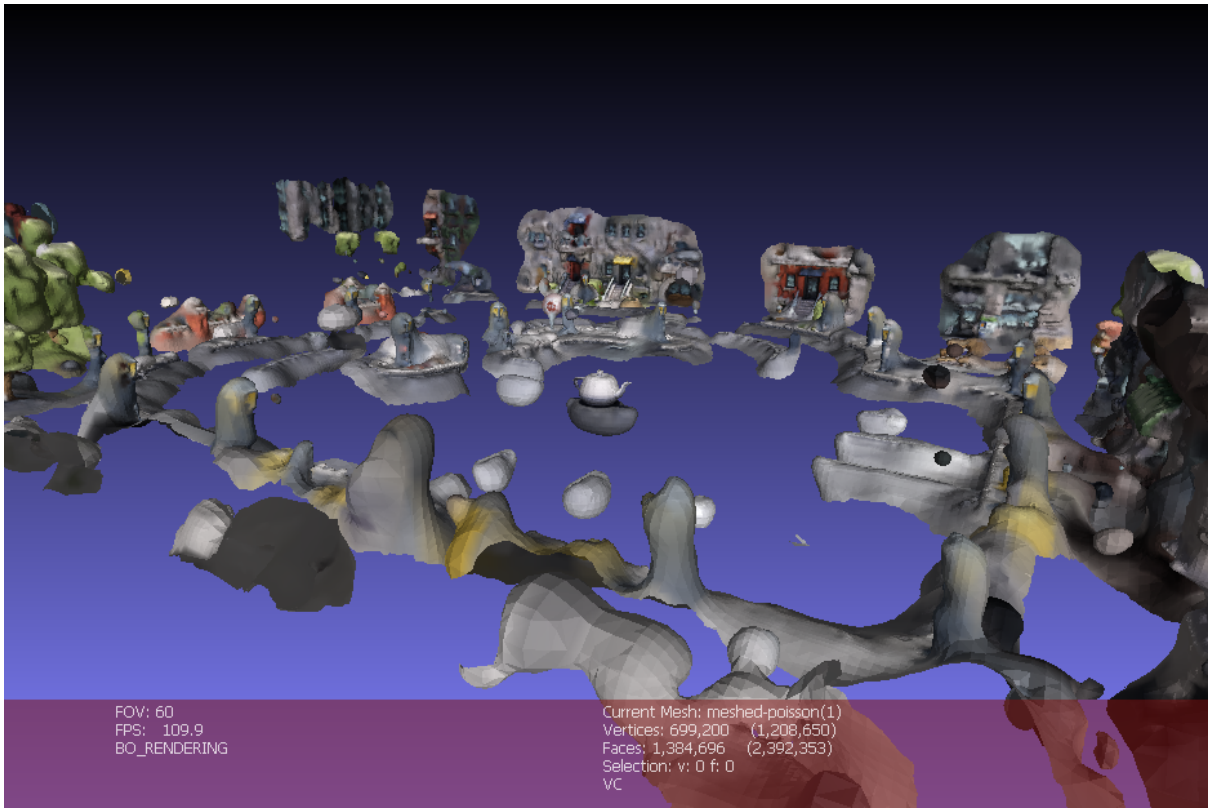

Figure 5.41: A view of a Cycles 48 4k image reconstruction of the teapot model in the cityscape scene when captured using a square camera placement pattern.

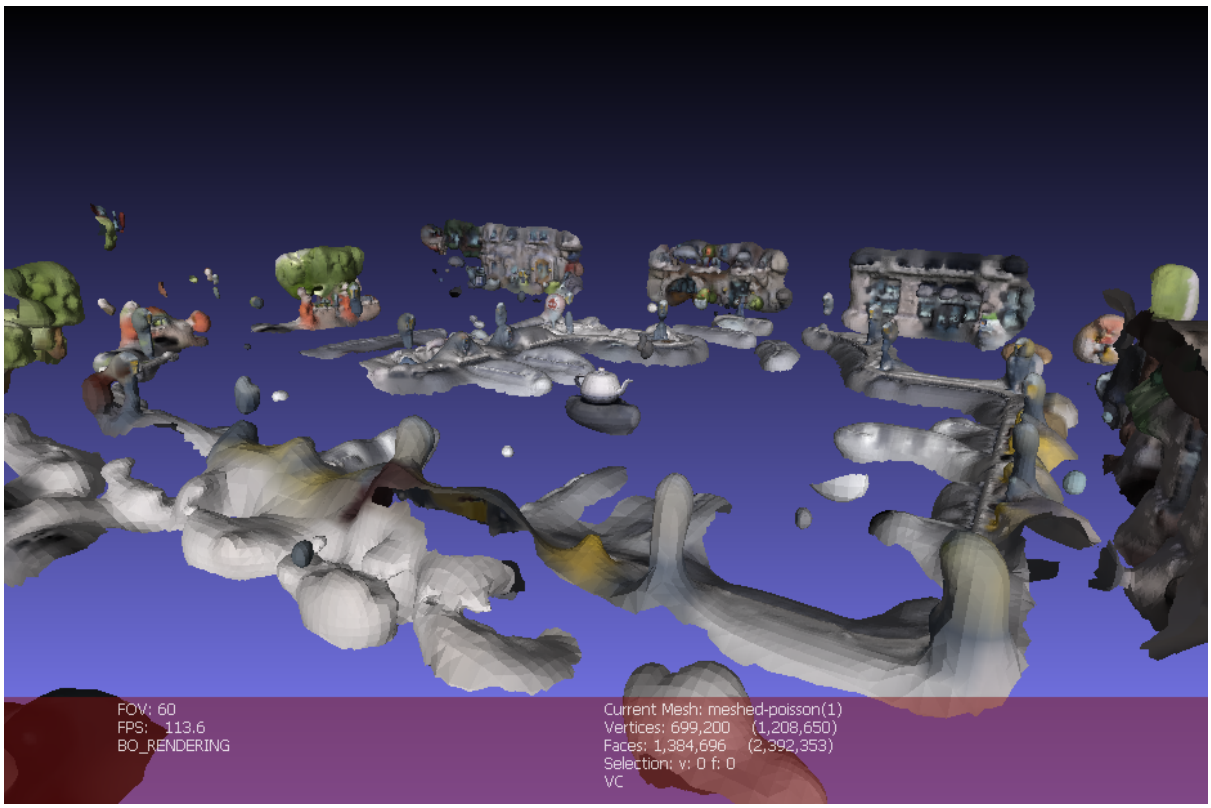

Figure 5.42: A view of a Cycles 48 4k image reconstruction of the teapot model in the cityscape scene when captured using a circle camera placement pattern.

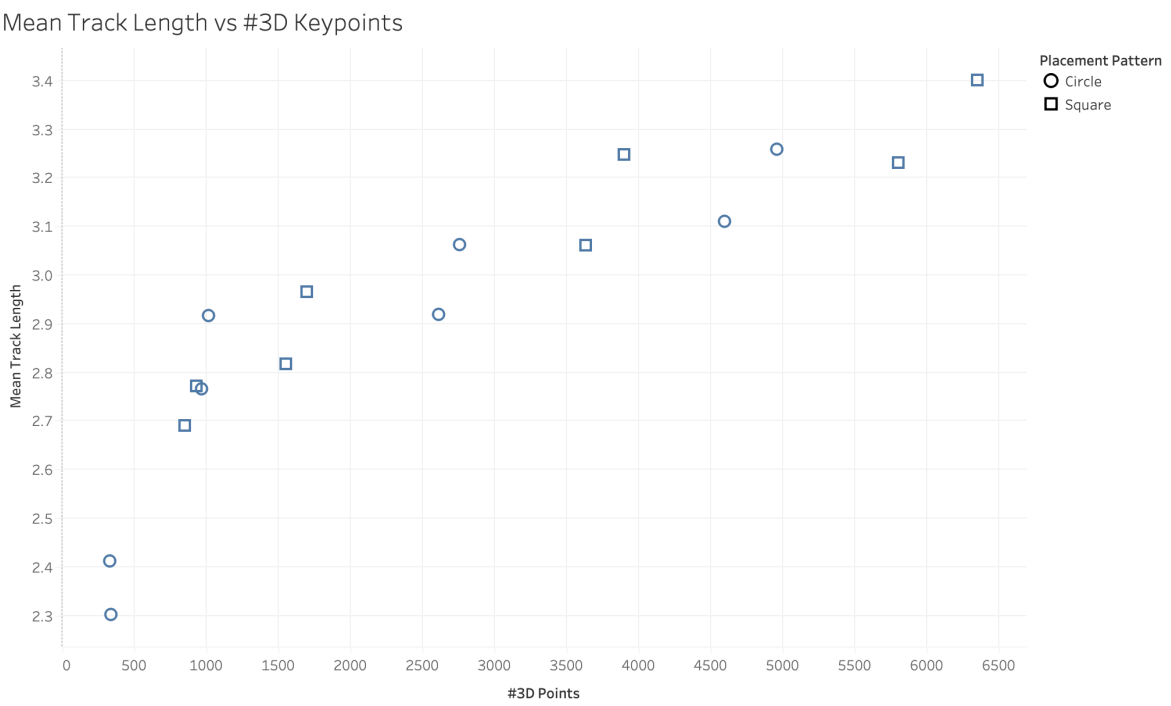

Figure 5.43: Graph plotting mean track length vs the number of 3D points reprojected for the phase 4 4k gradient teapot cityscape experiments.

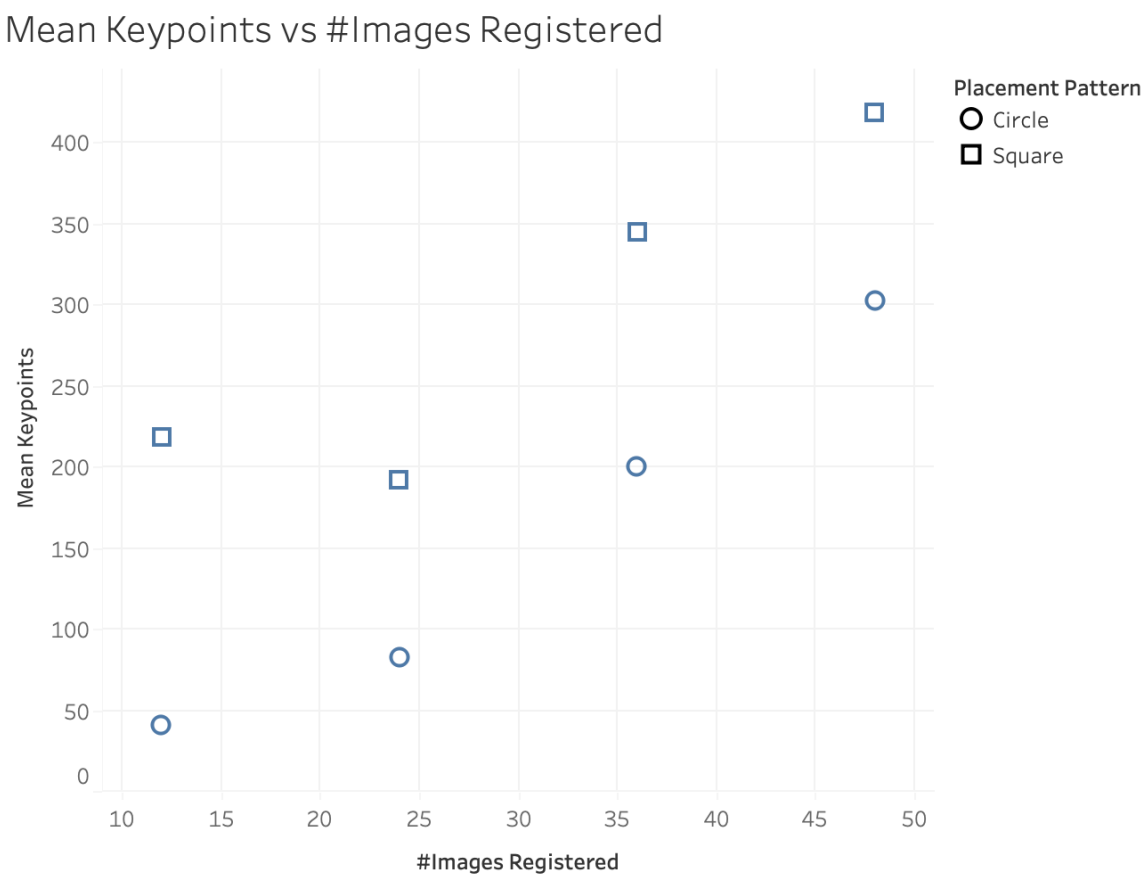

Figure 5.44: Graph plotting mean number of keypoints detected vs the number of images registered for the phase 4 1080p gradient teapot cityscape experiments.

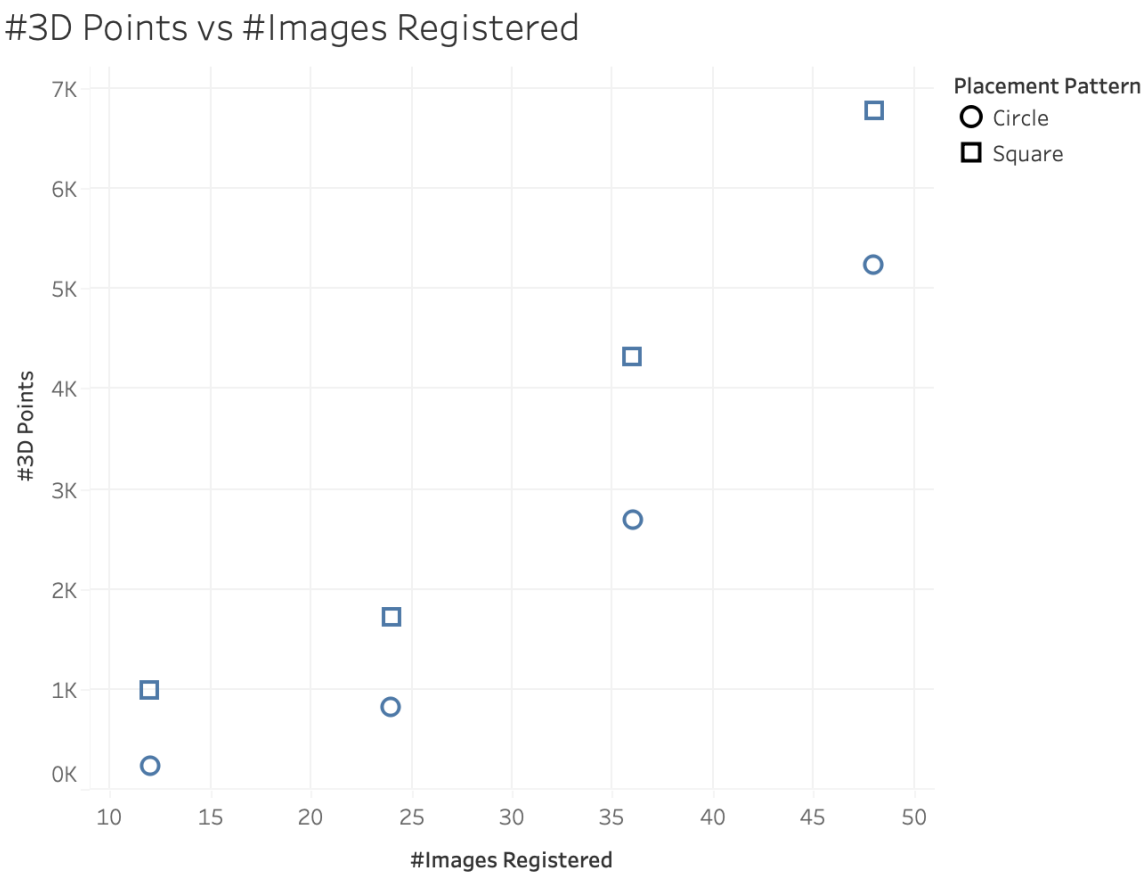

Figure 5.45: Graph plotting number of 3D points reprojected vs the number of images registered for the phase 4 1080p gradient teapot cityscape experiments.

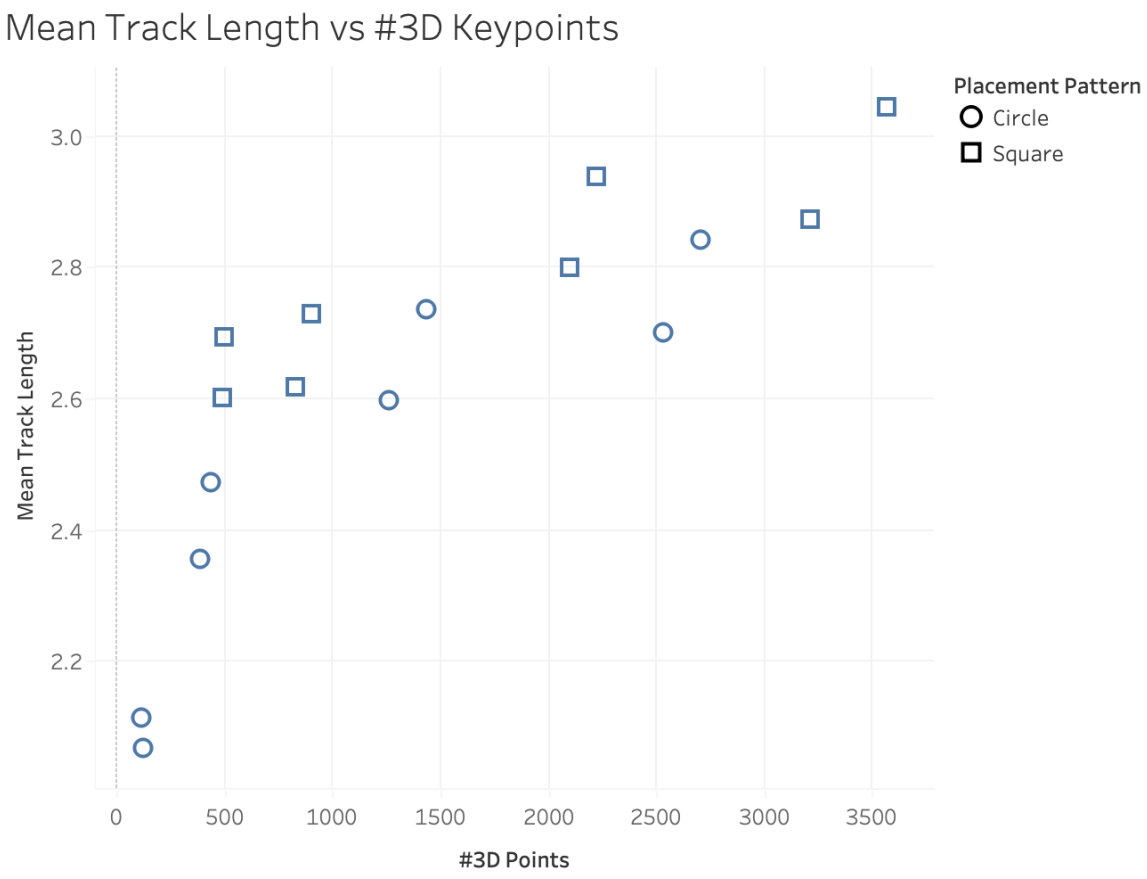

Figure 5.46: Graph plotting mean track length vs the number of 3D points reprojected for the phase 4 1080p gradient teapot cityscape experiments.

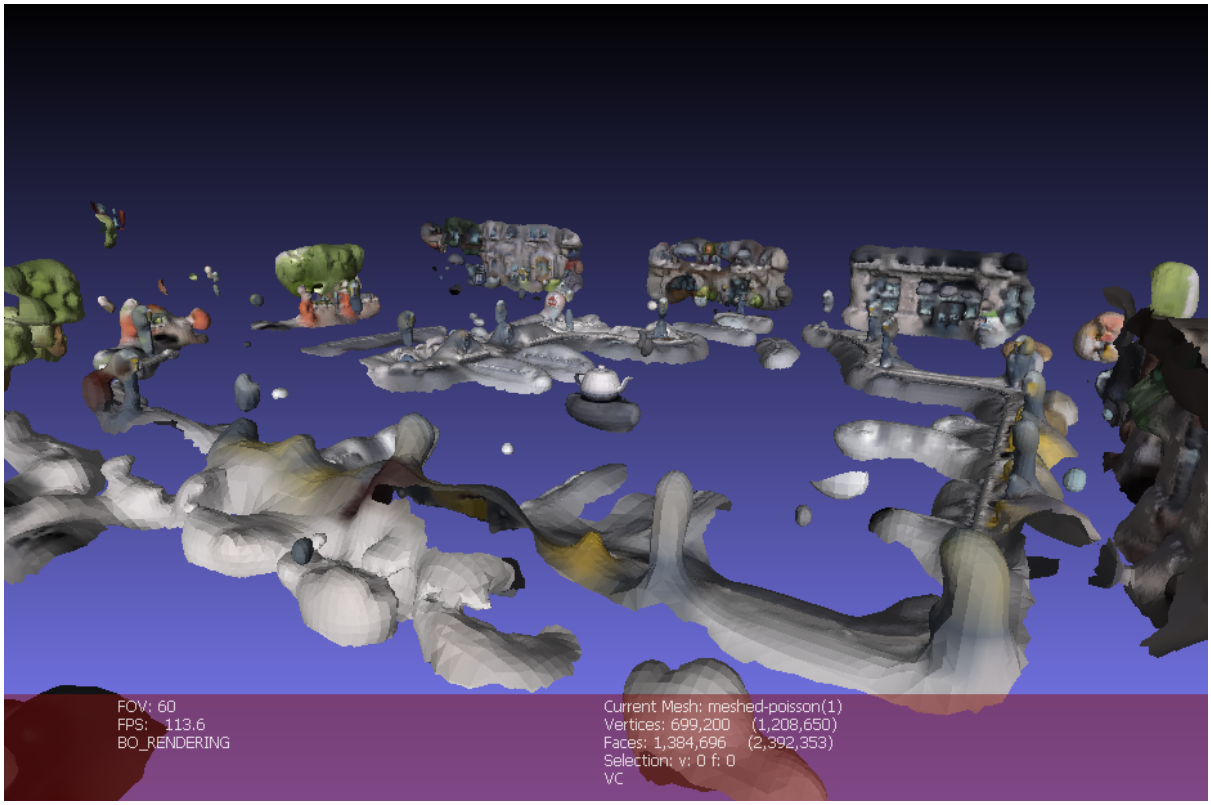

Figure 5.47: Side on view of a Cycles 48 4k image reconstruction of the gradient teapot model located in a cityscape captured in a circular pattern.

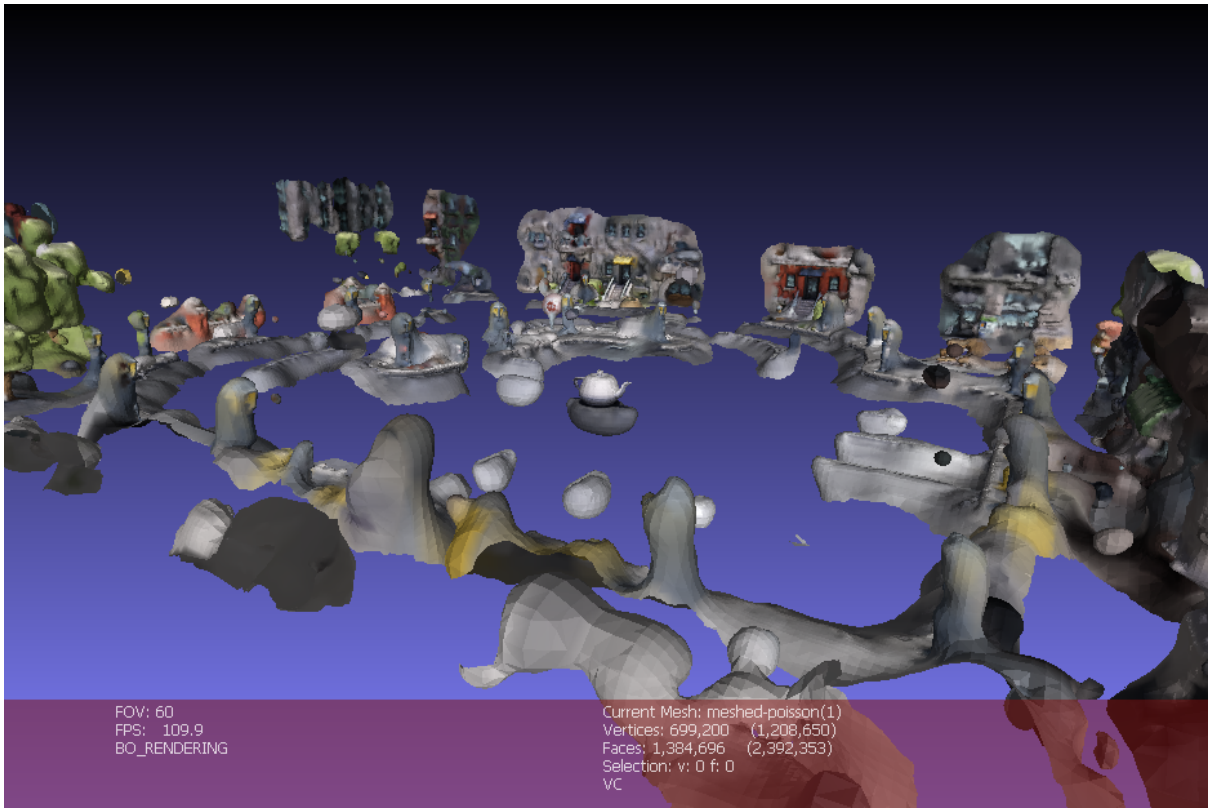

Figure 5.48: Side on view of a Cycles 48 4k image reconstruction of the gradient teapot model located in a cityscape captured in a square pattern.

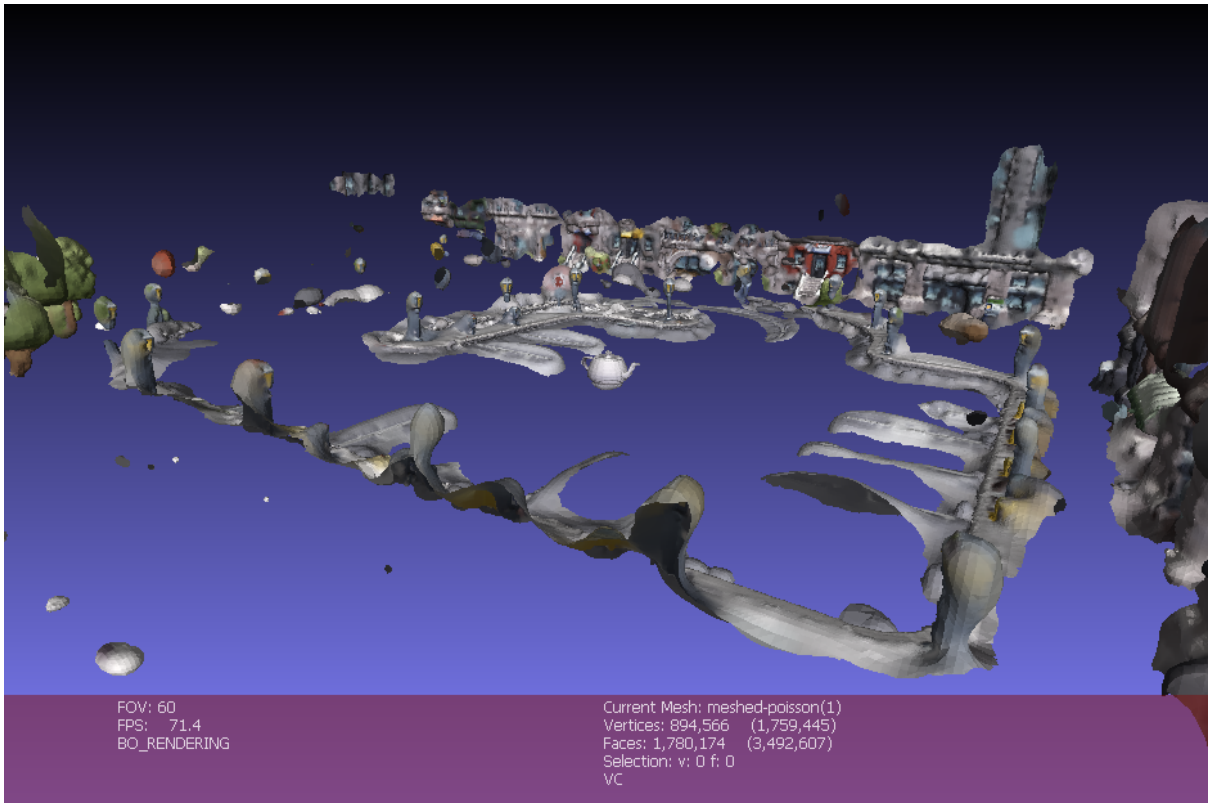

Figure 5.49: Side on view of a Evee 48 4k image reconstruction of the gradient teapot model located in a cityscape captured in a circular pattern.

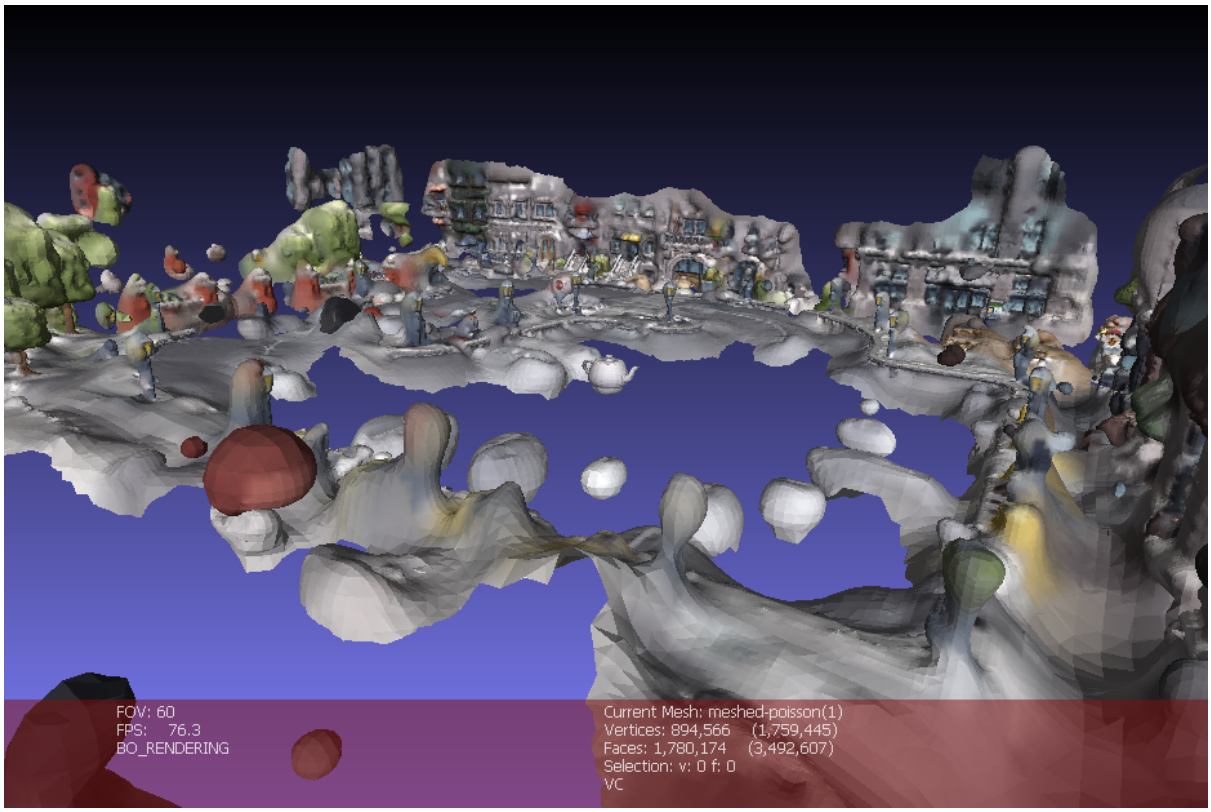

Figure 5.50: Side on view of a Evee 48 4k image reconstruction of the gradient teapot model located in a cityscape captured in a square pattern.

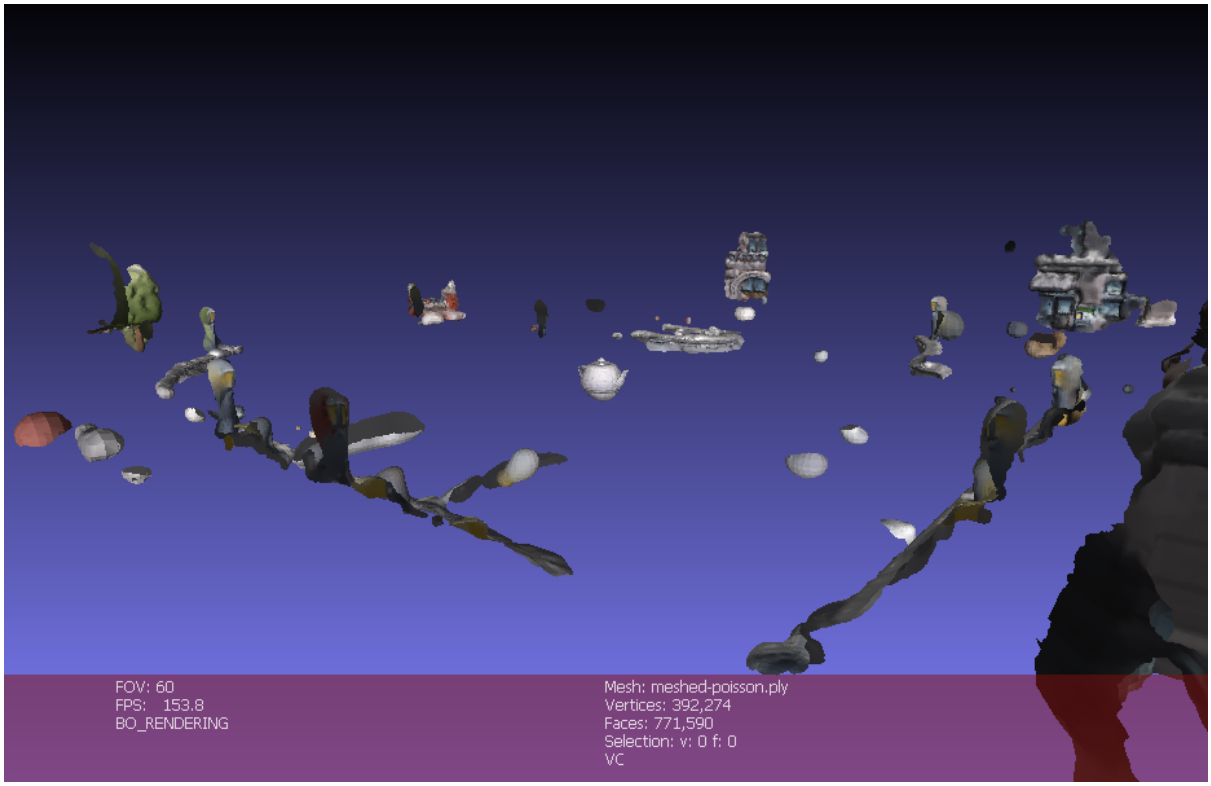

Figure 5.51: Side on view of a Evee 48 1080p image reconstruction of the gradient teapot model located in a cityscape captured in a square pattern.

5.5 Limitations

5.5.1 Supplying Ground Truth Data

By fixing the camera pose and positions in place, we remove a key part of the reconstruction pipeline from testing. This suppressed component is the SfM process, which includes bundle adjustment, where both the camera intrinsic and extrinsic parameters are refined to reduce the reprojection error. In doing this, the implementation has become closely tied to COLMAP, as the scene capture output is tailored to meet the COLMAP input requirements. In this way the final testing framework implementation is less modular than originally hoped. In its current form, the testing framework would require additional implementation effort to test with other reconstruction pipelines.

5.5.2 COLMAP Stability

During implementation, a number of challenges were faced when using COLMAP, primarily stability concerns. Throughout testing a number of bugs, random crashes and instances of unexpected behaviour were experienced. This proved tedious when automating reconstructions and hampered progress. In some cases, avoiding these issues required specific tuning of parameters, which hurt experiment repeatability. These factors combined make it a less accessible and less reliable choice for fully controlled testing.

5.5.3 Hardware Limitations

Due to the amount of load the reconstruction process puts on the GPU, if that GPU is the primary display adapter of the system the desktop environment of the machine would hang and become unresponsive for several seconds at a time. This is undesirable behaviour from an OS perspective and should be avoided at all costs. As such, there are built-in measures to detect and mitigate the issue baked into the OS. This would result in the operating system killing the reconstruction process to regain stability. While a workaround was implemented using Nvidia's nsight control panel, this behaviour is undesirable for general purpose reconstruction. To avoid this issue, the use of a machine with a secondary display adapter is required.

5.5.4 Reconstruction Timing

As discussed in section 3, it was decided to omit timing information from testing to enable a larger volume of tests to be performed. Without reconstruction timing information, the full scope of factors such as image-set size could not be assessed.

5.5.5 Mesh Quality Assessment

The Meshlab quality assessment component of the reconstruction testing framework was not fully automated during this research. Due to the varying number of applications, it became difficult to produce a singular quality measure upon which automation could be performed. As such the mesh quality assessment phase remains manual and requires time and expertise. While this does give greater control over the finer analysis of results, it detracts from the level of automation attainable.

5.6 Future Work

5.6.1 Evaluate Different Reconstruction Pipelines

COLMAP was used to perform reconstruction throughout this research. Hence, the results obtained only represent a small portion of the available image based reconstruction pipelines available. Using the testing principles discussed, the final implementation could be extended to test other reconstruction pipelines such as Meshroom or NeRF. This type of benchmark could enable us to better understand the characteristics of various reconstruction methods and identify areas of improvement within the field.

5.6.2 Timing Information and Additional Metrics

Gathering timing information during the reconstruction process has the potential to greatly enhance the quality of the results obtained. This additional information could better inform the conclusions drawn and help explain various reconstruction behaviours. It is also desirable to collect additional metrics about the reconstruction process. Such metrics of interest could include, landscape vs portrait images or random camera placement patterns. Finally, one key area of exploration that was omitted from testing is the use of known objects within a scene. This research could determine if the use of known objects within an arbitrary scene can enhance reconstruction quality, much in the same way adding the teapot model to the cityscape improved its reconstruction quality.

5.6.3 Model Alignment

During development, it was decided to manually supply camera ground truth information to align the final reconstruction. However, other solutions do exist but were not tested due to time limitations. COLMAP for example has an automatic Manhattan alignment method, which estimates the direction of gravity and uses this knowledge to transform the model accordingly. This line of enquiry could prove useful in further testing the final reconstruction quality using fully estimated camera parameters refined using the complete SfM process.

6 Conclusion

In this paper, a comprehensive investigation of the factors impacting 2D - 3D reconstruction was conducted. An understanding was gained of the research area via rigorous review of the existing literature. From this review, the state of the art in Structure-From-Motion based reconstruction was identified. This review uncovered that image resolution, number of images, distribution of images within the scene, object of interest texture and lighting conditions are prominent quality impacting factors for this type of reconstruction.

Using this knowledge, an automated testing framework was designed to quantify the impact of these factors on final reconstruction quality. This framework comprises of synthetic scene creation, capture, reconstruction and quality assessment. This framework was successfully implemented and used to conduct experiments to evaluate the impact of the identified factors. Stages from scene creation to reconstruction were fully automated with quality assessment still requiring manual intervention, due to the inherent application specific nature of perceived quality. Blender was used to perform synthetic scene creation and capture, COLMAP was used to perform SfM based reconstruction and MeshLab was used to perform final mesh quality assessment.

The experiments conducted were split into 4 phases, each investigating a different combination of impacting factors. The results obtained from these experiments uncovered further characteristics of the identified impacting factors and went on to validate the reconstruction findings from real life scenes in other research, but in a synthetic setting.

The first phase of testing identified that low detail geometry and homogeneous texturing pose significant challenges when reconstructing a capture studio like scene. Moreover, it was discovered that general detail sparsity within the scene yielded unrecognisable reconstruction results or even failure to reconstruct in some cases. Regardless of the scene setup, it was observed that increasing the number of images capturing the scene had a net benefit on reconstruction results. This fact holds true for all subsequent tests performed.

The second phase of testing established the benefit of using additional sensor data during reconstruction and further investigated the characteristics exhibited when reconstructing

poorly textured objects. The use of sensor data improved the performance of image registration, enabling the improved reconstruction of symmetric objects. From evaluating additional textures, it was solidified that low detail textures result in poor keypoint detection, and ultimately gives rise to incomplete areas and holes in the final mesh. Finally, this phase of testing showed signs of improved scene coverage when capturing using a circular camera placement pattern.

The third phase of testing showed how more detailed geometry and realistic scene configuration improves reconstruction quality. It confirmed the improved performance of a circular camera placement pattern when compared to a square placement pattern. By increasing the scene realism using ray-tracing, the crucial role lighting plays was established, especially to ensure intricate object details are captured. The impact of image resolution was also identified, with lower resolutions having noticeably lower final texture detail and greater mesh inaccuracies. Background segmentation posed a real issue during these capture studio like tests, as it did in previous tests. Poor segmentation gave rise to visible artificating and fringing to the edges of the final meshes produced.

The fourth and final phase of testing mimicked the characteristics of an urban environment more closely and measured the change to reconstruction quality. It was found that an increase to background scene detail had a positive impact on final reconstruction quality, enabling even the most challenging objects to be reconstructed more completely. This environment did exhibit different behaviour to previous tests, with the square camera placement pattern outperforming the circle placement pattern. This change draws a link between background scene detail and the optimal scene capture pattern.

The outcomes of these experiments has resulted in the contribution of a large body of experimental test results, which can be further analysed by others. It also ensured a refined automated testing framework was produced, which can be further extended and reused for additional testing with different scenes and reconstruction techniques. Ultimately, from these findings it can be concluded that an end-user trying to capture a scene should favour good lighting conditions with few shadows, like that of an overcast day. When capturing a scene with a central point of focus, the user should capture images following an evenly distributed circular pattern in low detail scenes and a square pattern in higher detail scenes. They should attempt to target in excess of 200 keypoints on average in their images, to achieve good reconstruction results. They should ensure the scene is not completely devoid of detail beyond the object of interest. Finally, they should capture at the highest resolution possible with a higher number of images being favourable.

Bibliography

[1] Siwei Wu, Shan Xiao, Yihua Di, and Cheng Di. 3d film animation image acquisition and feature processing based on the latest virtual reconstruction technology. *Complexity*, 2021, 2021. doi: 10.1155/2021/2331306.

[2] Wu Xiaojun, Xiao Shenghua, and Wei Jingyang. High accurate 3d reconstruction method using binocular stereo based on multiple constraints. In *2015 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pages 934–939, 2015. doi: 10.1109/ROBIO.2015.7418891.

[3] Ashutosh Singandhupe and Hung Manh La. A review of slam techniques and security in autonomous driving. In *2019 Third IEEE International Conference on Robotic Computing (IRC)*, pages 602–607, 2019. doi: 10.1109/IRC.2019.00122.

[4] Guillaume Trehard, Evangeline Pollard, Benazouz Bradai, and Fawzi Nashashibi. On line mapping and global positioning for autonomous driving in urban environment based on evidential slam. In *2015 IEEE Intelligent Vehicles Symposium (IV)*, pages 814–819, 2015. doi: 10.1109/IVS.2015.7225785.

[5] Henning Lategahn, Andreas Geiger, and Bernd Kitt. Visual slam for autonomous ground vehicles. In *2011 IEEE International Conference on Robotics and Automation*, pages 1732–1737, 2011. doi: 10.1109/ICRA.2011.5979711.

[6] Steven Mills, Lech Szymanski, and Reuben Johnson. Hierarchical structure from motion from endoscopic video. In *Proceedings of the 29th International Conference on Image and Vision Computing New Zealand*, IVCNZ '14, page 102–107, New York, NY, USA, 2014. Association for Computing Machinery. ISBN 9781450331845. doi: 10.1145/2683405.2683411. URL `https://doi-org.elib.tcd.ie/10.1145/2683405.2683411`.

[7] Angel Martell, Helge A. Lauterbach, klaus Schilling, and Andreas Nuchtcer. Benchmarking structure from motion algorithms of urban environments with applications to reconnaissance in search and rescue scenarios. In *2018 IEEE*

International Symposium on Safety, Security, and Rescue Robotics (SSRR), pages 1–7, 2018. doi: 10.1109/SSRR.2018.8468612.

[8] Dzenan Lapandic, Jasmin Velagic, and Haris Balta. Framework for automated reconstruction of 3d model from multiple 2d aerial images. In *2017 International Symposium ELMAR*, pages 173–176, 2017. doi: 10.23919/ELMAR.2017.8124461.

[9] Ke Gao, Hadi Aliakbarpour, Joshua Fraser, Koundinya Nouduri, Filiz Bunyak, Ricky Massaro, Guna Seetharaman, and Kannappan Palaniappan. Local feature performance evaluation for structure-from-motion and multi-view stereo using simulated city-scale aerial imagery. *IEEE Sensors Journal*, 21(10):11615–11627, 2021. doi: 10.1109/JSEjN.2020.3042810.

[10] Asla Sa, Karina Rodriguez, Ricardo Marroquim, Adolfo Ibanez, and Vivian Fonseca. Accessible digitisation and visualisation of open cultural heritage assets. In *GCH*, pages 59–68, 10 2019. doi: 10.2312/gch.20191349.

[11] M.J. Westoby, J. Brasington, N.F. Glasser, M.J. Hambrey, and J.M. Reynolds. 'structure-from-motion' photogrammetry: A low-cost, effective tool for geoscience applications. *Geomorphology*, 179:300–314, 2012. ISSN 0169-555X. doi: https://doi.org/10.1016/j.geomorph.2012.08.021. URL `https://www.sciencedirect.com/science/article/pii/S0169555X12004217`.

[12] Rafael Pagés, Konstantinos Amplianitis, David Monaghan, Jan Ondrej, and Aljosa Smolic. Affordable content creation for free-viewpoint video and vr/ar applications. *Journal of Visual Communication and Image Representation*, 53, 03 2018. doi: 10.1016/j.jvcir.2018.03.012.

[13] S. M. Iman Zolanvari, Susana Ruano, Aakanksha Rana, Alan Cummins, Rogerio Eduardo da Silva, Morteza Rahbar, and Aljosa Smolic. Dublincity: Annotated lidar point cloud and its applications. *arXiv preprint arXiv:1909.03613*, 09 2019. doi: 10.48550/ARXIV.1909.03613. URL `https://arxiv.org/abs/1909.03613`.

[14] Mial E Warren. Automotive lidar technology. In *2019 Symposium on VLSI Circuits*, pages C254–C255, 2019. doi: 10.23919/VLSIC.2019.8777993.

[15] John A Christian and Scott Cryan. A survey of lidar technology and its use in spacecraft relative navigation. In *AIAA Guidance, Navigation, and Control (GNC) Conference*, page 4641, 2013.

[16] Shahram Izadi, David Kim, Otmar Hilliges, David Molyneaux, Richard Newcombe, Pushmeet Kohli, Jamie Shotton, Steve Hodges, Dustin Freeman, Andrew Davison, and Andrew Fitzgibbon. Kinectfusion: Real-time 3d reconstruction and interaction using a

moving depth camera. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, UIST '11, page 559–568, New York, NY, USA, 2011. Association for Computing Machinery. ISBN 9781450307161. doi: 10.1145/2047196.2047270. URL https://doi.org/10.1145/2047196.2047270.

[17] Emin Zerman, Néill O'Dwyer, Gareth W. Young, and Aljosa Smolic. A case study on the use of volumetric video in augmented reality for cultural heritage. In *Proceedings of the 11th Nordic Conference on Human-Computer Interaction (NordiCHI '20)*, pages 1 – 5, Tallinn, Estonia, 2020. Association for Computing Machinery (ACM). ISBN 978-1-4503-7579-5. doi: https://doi.org/10.1145/3419249.3420115. URL https://v-sense.scss.tcd.ie: 443/wp-content/uploads/2020/10/3419249.3420115-1.pdf.

[18] Johannes L. Schönberger and Jan-Michael Frahm. Structure-from-motion revisited. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4104–4113, 2016. doi: 10.1109/CVPR.2016.445.

[19] Tom Davis. Homogeneous coordinates and computer graphics. *URL www.geometer.org/mathcircles*, 2001. URL www.geometer.org/mathcircles.

[20] Edward Angel and Dave Shreiner. *Interactive Computer Graphics: A Top-Down Approach with Shader-Based OpenGL, 6th Edition*. Pearson, 2012. ISBN 9780132545235.

[21] Hyukseong Kwon, Johnny Park, and Avinash C. Kak. A new approach for active stereo camera calibration. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 3180–3185, 2007. doi: 10.1109/ROBOT.2007.363963.

[22] Toni Schenk. Introduction to photogrammetry. *The Ohio State University, Columbus*, 106, 2005.

[23] John McCarthy. Multi-image photogrammetry as a practical tool for cultural heritage survey and community engagement. *Journal of Archaeological Science*, 43:175–185, 2014. ISSN 0305-4403. doi: https://doi.org/10.1016/j.jas.2014.01.010. URL https://www.sciencedirect.com/science/article/pii/S0305440314000132.

[24] Jakob Iglhaut, Carlos Cabo, Stefano Puliti, Livia Piermattei, James O'Connor, and Jacqueline Rosette. Structure from motion photogrammetry in forestry: a review. *Current Forestry Reports*, 5:155–168, 9 2019. ISSN 2198-6436. doi: 10.1007/s40725-019-00094-3. URL https://doi.org/10.1007/s40725-019-00094-3.

[25] Sketchfab. Photogrammetry 3d models, 2022. URL https://sketchfab.com/tags/photogrammetry.

[26] 3Dsourced. Photogrammetry guide 2022, 2022. URL https://www.3dsourced.com/guides/photogrammetry-guide/.

[27] H. C. Longuet-Higgins. A computer algorithm for reconstructing a scene from two projections. *Nature*, 293:133–135, 9 1981. ISSN 1476-4687. doi: 10.1038/293133a0. URL https://doi.org/10.1038/293133a0.

[28] C. Tomasi and T. Kanade. Shape and motion without depth. In *[1990] Proceedings Third International Conference on Computer Vision*, pages 91–95, 1990. doi: 10.1109/ICCV.1990.139501.

[29] Noah Snavely, Steven M. Seitz, and Richard Szeliski. Photo tourism: Exploring photo collections in 3d. *ACM Trans. Graph.*, 25(3):835–846, jul 2006. ISSN 0730-0301. doi: 10.1145/1141911.1141964. URL https://doi-org.elib.tcd.ie/10.1145/1141911.1141964.

[30] Sameer Agarwal, Noah Snavely, Steven M. Seitz, and Richard Szeliski. Bundle adjustment in the large. In Kostas Daniilidis, Petros Maragos, and Nikos Paragios, editors, *Computer Vision – ECCV 2010*, pages 29–42, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. ISBN 978-3-642-15552-9. doi: 10.1007/978-3-642-15552-9_3.

[31] Johannes L. Schönberger, Enliang Zheng, Jan-Michael Frahm, and Marc Pollefeys. Pixelwise view selection for unstructured multi-view stereo. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *Computer Vision – ECCV 2016*, pages 501–518, Cham, 2016. Springer International Publishing. ISBN 978-3-319-46487-9. doi: 10.1007/978-3-319-46487-9_31.

[32] Ricardo Martin-Brualla, Noha Radwan, Mehdi S. M. Sajjadi, Jonathan T. Barron, Alexey Dosovitskiy, and Daniel Duckworth. Nerf in the wild: Neural radiance fields for unconstrained photo collections. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7206–7215, 2021. doi: 10.1109/CVPR46437.2021.00713.

[33] Kun Huang, Jingyuan Li, Ye Liu, Liang Chang, and Jun Zhou. A survey on feature point extraction techniques. In *2021 18th International SoC Design Conference (ISOCC)*, pages 201–202, 2021. doi: 10.1109/ISOCC53507.2021.9613858.

[34] Chris Harris, Mike Stephens, et al. A combined corner and edge detector. In *Alvey vision conference*, volume 15, pages 10–5244. Citeseer, 1988.

[35] Edward Rosten and Tom Drummond. Machine learning for high-speed corner detection. In Aleš Leonardis, Horst Bischof, and Axel Pinz, editors, *Computer Vision – ECCV 2006*, pages 430–443, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg. ISBN 978-3-540-33833-8. doi: 10.1007/11744023_34.

[36] Onur Ozyesil, Vladislav Voroninski, Ronen Basri, and Amit Singer. A survey of structure from motion, 2017. URL https://arxiv.org/abs/1701.08493.

[37] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 11 2004. ISSN 1573-1405. doi: 10.1023/B:VISI.0000029664.99615.94. URL https://doi.org/10.1023/B:VISI.0000029664.99615.94.

[38] Chengcai Leng, Hai Zhang, Bo Li, Guorong Cai, Zhao Pei, and Li He. Local feature descriptor for image matching: A survey. *IEEE Access*, 7:6424–6434, 2019. doi: 10.1109/ACCESS.2018.2888856.

[39] Sameer Agarwal, Noah Snavely, Ian Simon, Steven M. Seitz, and Richard Szeliski. Building rome in a day. In *2009 IEEE 12th International Conference on Computer Vision*, pages 72–79, 2009. doi: 10.1109/ICCV.2009.5459148.

[40] Christopher Sweeney, Tobias Hollerer, and Matthew Turk. Theia: A fast and scalable structure-from-motion library. In *Proceedings of the 23rd ACM International Conference on Multimedia*, MM '15, page 693–696, New York, NY, USA, 2015. Association for Computing Machinery. ISBN 9781450334594. doi: 10.1145/2733373.2807405. URL https://doi-org.elib.tcd.ie/10.1145/2733373.2807405.

[41] Bill Triggs, Philip F. McLauchlan, Richard I. Hartley, and Andrew W. Fitzgibbon. Bundle adjustment — a modern synthesis. In Bill Triggs, Andrew Zisserman, and Richard Szeliski, editors, *Vision Algorithms: Theory and Practice*, pages 298–372, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg. ISBN 978-3-540-44480-0. doi: 10.1007/3-540-44480-7_21.

[42] Simon Fuhrmann, Fabian Langguth, Nils Moehrle, Michael Waechter, and Michael Goesele. Mve-an image-based reconstruction environment. *Comput. Graph.*, 53(PA): 44–53, dec 2015. ISSN 0097-8493. doi: 10.1016/j.cag.2015.09.003. URL https://doi.org/10.1016/j.cag.2015.09.003.

[43] Paolo Cignoni, Marco Callieri, Massimiliano Corsini, Matteo Dellepiane, Fabio Ganovelli, and Guido Ranzuglia. MeshLab: an Open-Source Mesh Processing Tool. In Vittorio Scarano, Rosario De Chiara, and Ugo Erra, editors, *Eurographics Italian Chapter Conference*. The Eurographics Association, 2008. ISBN 978-3-905673-68-5. doi: 10.2312/LocalChapterEvents/ItalChap/ItalianChapConf2008/129-136.

[44] Carsten Griwodz, Simone Gasparini, Lilian Calvet, Pierre Gurdjos, Fabien Castan, Benoit Maujean, Gregoire De Lillo, and Yann Lanthony. Alicevision Meshroom: An open-source 3D reconstruction pipeline. In *Proceedings of the 12th ACM Multimedia Systems Conference - MMSys '21*. ACM Press, 2021. doi: 10.1145/3458305.3478443.

[45] David J. Crandall, Andrew Owens, Noah Snavely, and Daniel P. Huttenlocher. Sfm with mrfs: Discrete-continuous optimization for large-scale structure from motion. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(12):2841–2853, 2013. doi: 10.1109/TPAMI.2012.218.

[46] Aleksander Holynski, David Geraghty, Jan-Michael Frahm, Chris Sweeney, and Richard Szeliski. Reducing drift in structure from motion using extended features. In *International Conference on 3D Vision (3DV)*, pages 51–60, November 2020. URL `https://homes.cs.washington.edu/~holynski/publications/reducing_drift/`.

[47] Riccardo Gherardi, Michela Farenzena, and Andrea Fusiello. Improving the efficiency of hierarchical structure-and-motion. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1594–1600, 2010. doi: 10.1109/CVPR.2010.5539782.

[48] Paolo Cignoni, Claudio Rocchini, and Roberto Scopigno. Metro: measuring error on simplified surfaces. In *Computer Graphics Forum*, volume 17, pages 167–174. Blackwell Publishers, 1998.

[49] Rafael Pagés, Konstantinos Amplianitis, Jan Ondrej, Emin Zerman, and Aljosa Smolic. Volograms & v-sense volumetric video dataset. *ISO/IEC JTC1/SC29/WG07 MPEG2021/m56767*, 2021.

[50] Thomas Läbe and Wolfgang Forstner. Geometric stability of low-cost digital consumer cameras. *International Archives of Photogrammetry and Remote Sensing*, 35, 01 2004.

[51] Blender Foundation. Blender, 2022. URL `https://www.blender.org/about/`.

[52] Sebastian Bullinger, Christoph Bodensteiner, and Michael Arens. A photogrammetry-based framework to facilitate image-based modeling and automatic camera tracking. *arXiv preprint arXiv:2012.01044*, 2021.

[53] Sebastian Bullinger, Christoph Bodensteiner, and Michael Arens. Blender-addon-photogrammetry-importer, 2022. URL `https://github.com/SBCV/Blender-Addon-Photogrammetry-Importer`.

[54] Alessandro Muntoni and Paolo Cignoni. PyMeshLab, January 2021.

[55] nathan.arditti. Medieval wall of bricks assets, 03 2022. URL `https://skfb.ly/otJBW`.

A1 Appendix

A1.1 Reconstruction Results

A1.1.1 Phase 1 Experiment Results

Table A1.1: Phase 1 experiment results.

Model	Texture	Resolution	Lighting	Render Engine	#Images	#Images Registered	Mean Keypoints	#3D Points	Mean Track Length
Teapot	Grey	1080p	Point	Evee	36	0	NA	NA	NA
Teapot	White	1080p	Point	Evee	108	0	NA	NA	NA
Teapot	Checkerboard	4k	Point	Evee	54	41	43.8048	564	3.1843
Teapot	Checkerboard	4k	Point	Evee	108	48	181.4375	1552	5.6114
Teapot	Checkerboard	4k	Point	Evee	216	216	180.6388	3707	10.5254
Teapot	Checkerboard	4k	Ambient	Evee	54	43	38.6511	435	3.8206
Teapot	Checkerboard	4k	Ambient	Evee	108	96	81.8703	1184	7.4679
Teapot	Checkerboard	4k	Ambient	Evee	216	216	170.0833	3566	10.3022
Cube	Calibration Pattern	4k	Point	Evee	108	108	175.6728	1002	18.7594

A1.1.2 Phase 2 Homogeneous Texture Results

Table A1.2: Phase 2 homogeneous texture results.

Model	Texture	Resolution	Lighting	Render Engine	#Images	#Images Registered	Mean Keypoints	#3D Points	Mean Track Length
Teapot	Black	4k	Ambient	Evee	216	216	10.9907	409	5.8044

A1.1.3 Phase 2 Additional Homogeneous Texture Results

Table A1.3: Phase 2 additional homogeneous texture results.

Model	Texture	Resolution	Lighting	Render Engine	Placement Pattern	#Images	#Images Registered	Mean Keypoints	#3D Points	Mean Track Length
Teapot	Black	1080p	Point	Cycles	Circle	12	12	15.4166	59	3.1355
Teapot	Black	1080p	Point	Cycles	Circle	24	24	11.3333	78	3.4871
Teapot	Black	1080p	Point	Cycles	Circle	36	36	10.6111	99	3.8585
Teapot	Black	1080p	Point	Cycles	Circle	48	48	12.1250	146	3.9863
Teapot	Black	1080p	Point	Cycles	Square	12	12	12.2500	45	3.2666
Teapot	Black	1080p	Point	Cycles	Square	24	24	13.1666	95	3.3263
Teapot	Black	1080p	Point	Cycles	Square	36	36	10.3055	101	3.6732
Teapot	Black	1080p	Point	Cycles	Square	48	48	11.7291	131	4.2977
Teapot	Black	1080p	Point	Evee	Circle	12	12	16.8333	65	3.1076
Teapot	Black	1080p	Point	Evee	Circle	24	24	10.7500	72	3.5833
Teapot	Black	1080p	Point	Evee	Circle	36	36	10.1111	89	4.0898
Teapot	Black	1080p	Point	Evee	Circle	48	48	11.4791	130	4.2384
Teapot	Black	1080p	Point	Evee	Square	12	12	9.7500	33	3.5454
Teapot	Black	1080p	Point	Evee	Square	24	24	11.2500	77	3.5064
Teapot	Black	1080p	Point	Evee	Square	36	36	12.0555	107	4.0560
Teapot	Black	1080p	Point	Evee	Square	48	48	12.2291	120	4.8916
Teapot	Black	4k	Point	Cycles	Circle	12	12	8.5000	33	3.0909
Teapot	Black	4k	Point	Cycles	Circle	24	24	7.7500	64	2.9062
Teapot	Black	4k	Point	Cycles	Circle	36	36	6.1388	71	3.1126
Teapot	Black	4k	Point	Cycles	Circle	48	48	6.7291	99	3.2626
Teapot	Black	4k	Point	Cycles	Square	12	12	8.6666	35	2.9714
Teapot	Black	4k	Point	Cycles	Square	24	24	9.3750	75	3.0000
Teapot	Black	4k	Point	Cycles	Square	36	36	7.2500	79	3.3037
Teapot	Black	4k	Point	Cycles	Square	48	48	6.7291	89	3.6292
Teapot	Black	4k	Point	Evee	Circle	12	12	9.8333	37	3.1891
Teapot	Black	4k	Point	Evee	Circle	24	24	6.8333	53	3.0943
Teapot	Black	4k	Point	Evee	Circle	36	36	7.1944	85	3.0470
Teapot	Black	4k	Point	Evee	Circle	48	48	6.2916	90	3.3555
Teapot	Black	4k	Point	Evee	Square	12	12	8.0833	33	2.9393
Teapot	Black	4k	Point	Evee	Square	24	24	8.8750	68	3.1323
Teapot	Black	4k	Point	Evee	Square	36	36	5.7777	63	3.3015
Teapot	Black	4k	Point	Evee	Square	48	48	6.3958	82	3.7439

A1.1.4 Phase 2 Gradient Texture Results

Table A1.4: Phase 2 gradient texture results.

Model	Texture	Resolution	Lighting	Render Engine	Placement Pattern	#Images	#Images Registered	Mean Keypoints	#3D Points	Mean Track Length
Teapot	Gradient	4k	Ambient	Cycles	Square	12	12	5.8333	22	3.1818
Teapot	Gradient	4k	Ambient	Cycles	Square	24	24	3.9166	28	3.3571
Teapot	Gradient	4k	Ambient	Cycles	Square	36	36	6.1388	68	3.2500
Teapot	Gradient	4k	Ambient	Cycles	Square	48	48	7.1250	91	3.7582
Teapot	Gradient	4k	Ambient	Cycles	Circle	12	12	4.5000	20	2.7000
Teapot	Gradient	4k	Ambient	Cycles	Circle	24	24	2.8750	21	3.2857
Teapot	Gradient	4k	Ambient	Cycles	Circle	36	36	4.3611	50	3.1400
Teapot	Gradient	4k	Ambient	Cycles	Circle	48	48	5.3958	75	3.4533
Teapot	Gradient	4k	Ambient	Cycles	Circle	54	54	6.4629	116	3.0086
Teapot	Gradient	4k	Ambient	Cycles	Circle	108	108	12.1481	374	3.5080
Teapot	Gradient	4k	Ambient	Cycles	Circle	216	216	19.6342	935	4.5358
Teapot	Gradient	1080p	Ambient	Cycles	Circle	54	54	8.3333	125	3.6000
Teapot	Gradient	1080p	Ambient	Cycles	Circle	108	108	13.0092	319	4.4043
Teapot	Gradient	1080p	Ambient	Cycles	Circle	216	216	18.0324	653	5.9647

A1.1.5 Phase 2 Checkerboard Texture Results

Table A1.5: Phase 2 checkerboard texture results.

Model	Texture	Resolution	Lighting	Render Engine	Placement Pattern	#Images	#Images Registered	Mean Keypoints	#3D Points	Mean Track Length
Teapot	Checkerboard	4k	Ambient	Cycles	Square	12	12	83.7500	369	2.7235
Teapot	Checkerboard	4k	Ambient	Cycles	Square	24	24	87.1250	737	2.8371
Teapot	Checkerboard	4k	Ambient	Cycles	Square	36	36	93.5833	1122	3.0026
Teapot	Checkerboard	4k	Ambient	Cycles	Square	48	48	104.7083	1477	3.4028
Teapot	Checkerboard	4k	Ambient	Cycles	Circle	12	12	78.7500	354	2.6694
Teapot	Checkerboard	4k	Ambient	Cycles	Circle	24	24	83.3333	720	2.7777
Teapot	Checkerboard	4k	Ambient	Cycles	Circle	36	36	89.5277	1072	3.0065
Teapot	Checkerboard	4k	Ambient	Cycles	Circle	48	48	99.0416	1472	3.2296
Teapot	Checkerboard	4k	Ambient	Cycles	Circle	108	108	149.4907	3518	4.5892
Teapot	Checkerboard	4k	Ambient	Cycles	Circle	216	216	226.4953	7271	6.7285
Teapot	Checkerboard	1080p	Ambient	Cycles	Circle	54	54	114.5555	1641	3.7696
Teapot	Checkerboard	1080p	Ambient	Cycles	Circle	108	108	163.4259	3035	5.8154
Teapot	Checkerboard	1080p	Ambient	Cycles	Circle	216	216	230.0925	5334	9.3175

A1.1.6 Phase 3 Medieval Wall 4k Results

Table A1.6: Phase 3 medieval wall 4k experiment results

Model	Resolution	Lighting	Render Engine	Placement Pattern	#Images	#Images Registered	Mean Keypoints	#3D Points	Mean Track Length
Medieval Wall	4k	Ambient	Cycles	Square	12	12	330.5000	1362	2.9118
Medieval Wall	4k	Ambient	Cycles	Square	24	24	330.1666	2827	2.8029
Medieval Wall	4k	Ambient	Cycles	Square	36	36	410.0000	4831	3.0552
Medieval Wall	4k	Ambient	Cycles	Square	48	48	504.7500	6859	3.5322
Medieval Wall	4k	Ambient	Cycles	Square	96	96	1238.6562	22457	5.2950
Medieval Wall	4k	Ambient	Cycles	Square	192	192	1769.3697	41794	8.1284
Medieval Wall	4k	Ambient	Cycles	Circle	12	12	314.9166	1375	2.7483
Medieval Wall	4k	Ambient	Cycles	Circle	24	24	361.1666	3185	2.7215
Medieval Wall	4k	Ambient	Cycles	Circle	36	36	441.8888	5303	2.9998
Medieval Wall	4k	Ambient	Cycles	Circle	48	48	598.9791	8234	3.4917
Medieval Wall	4k	Ambient	Cycles	Circle	96	96	1593.4062	28430	5.3804
Medieval Wall	4k	Ambient	Cycles	Circle	192	192	2289.6770	53793	8.1724
Medieval Wall	4k	Ambient	Evee	Square	12	12	119.7500	503	2.8568
Medieval Wall	4k	Ambient	Evee	Square	24	24	132.7500	1197	2.6616
Medieval Wall	4k	Ambient	Evee	Square	36	36	172.8333	2184	2.8489
Medieval Wall	4k	Ambient	Evee	Square	48	48	221.0625	3255	3.2599
Medieval Wall	4k	Ambient	Evee	Circle	12	12	128.3333	590	2.6101
Medieval Wall	4k	Ambient	Evee	Circle	24	24	166.8750	1551	2.5822
Medieval Wall	4k	Ambient	Evee	Circle	36	36	199.3888	2494	2.8781
Medieval Wall	4k	Ambient	Evee	Circle	48	48	286.1875	4219	3.2559

A1.1.7 Phase 3 Medieval Wall 1080p Results

Table A1.7: Phase 3 medieval wall 1080p experiment results

Model	Resolution	Lighting	Render Engine	Placement Pattern	#Images	#Images Registered	Mean Keypoints	#3D Points	Mean Track Length
Medieval Wall	1080p	Ambient	Cycles	Square	12	12	134.5000	551	2.9292
Medieval Wall	1080p	Ambient	Cycles	Square	24	24	130.0833	1084	2.8800
Medieval Wall	1080p	Ambient	Cycles	Square	36	36	177.0277	2001	3.1849
Medieval Wall	1080p	Ambient	Cycles	Square	48	48	218.9791	2872	3.6598
Medieval Wall	1080p	Ambient	Cycles	Circle	12	12	123.5833	524	2.8301
Medieval Wall	1080p	Ambient	Cycles	Circle	24	24	143.9583	1243	2.7795
Medieval Wall	1080p	Ambient	Cycles	Circle	36	36	191.8888	2233	3.0935
Medieval Wall	1080p	Ambient	Cycles	Circle	48	48	262.7083	3531	3.5712
Medieval Wall	1080p	Ambient	Evee	Square	12	12	53.6666	226	2.8495
Medieval Wall	1080p	Ambient	Evee	Square	24	24	54.0833	476	2.7268
Medieval Wall	1080p	Ambient	Evee	Square	36	36	73.8055	897	2.9620
Medieval Wall	1080p	Ambient	Evee	Square	48	48	86.2500	1216	3.4046
Medieval Wall	1080p	Ambient	Evee	Circle	12	12	47.0000	209	2.6985
Medieval Wall	1080p	Ambient	Evee	Circle	24	24	62.8333	561	2.6880
Medieval Wall	1080p	Ambient	Evee	Circle	36	36	80.2777	986	2.9310
Medieval Wall	1080p	Ambient	Evee	Circle	48	48	115.8125	1655	3.3589

A1.1.8 Phase 3 Difference in Render Engine 4k

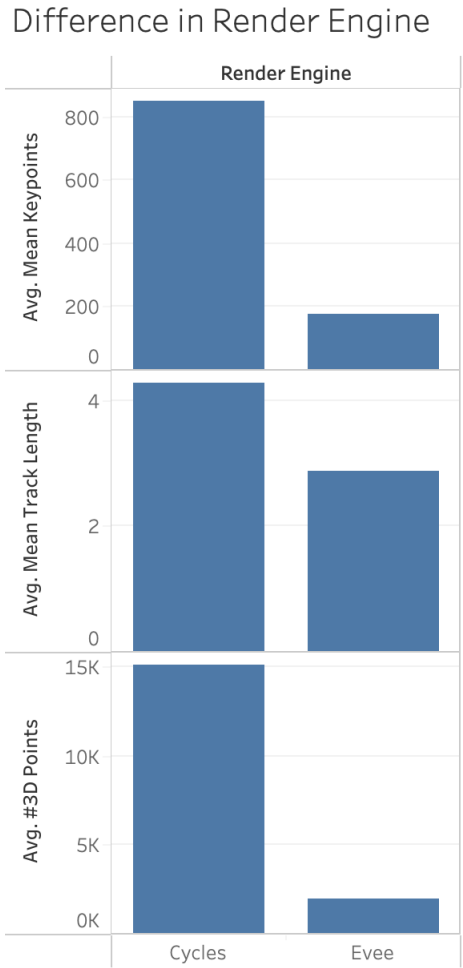

Figure A1.1: Graph summarising the results and differences between the Evee and Cycles render engines for the 4k medieval wall experiments.

A1.1.9 Phase 3 Difference in Render Engine 1080p

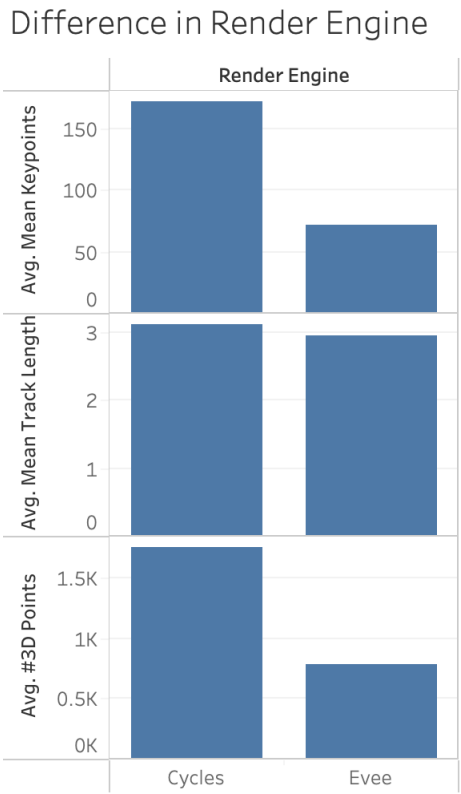

Figure A1.2: Graph summarising the results and differences between the Evee and Cycles render engines for the 1080p medieval wall experiments.

A1.1.10 Phase 4 Gradient Teapot 4k Cityscape Experiment Results

Table A1.8: Phase 4 gradient teapot 4k cityscape experiment results

Model	Resolution	Lighting	Render Engine	Placement Pattern	#Images	#Images Registered	Mean Keypoints	#3D Points	Mean Track Length
Cityscape	4k	Simulated Sunlight	Cycles	Circle	12	12	66.3333	330	2.4121
Cityscape	4k	Simulated Sunlight	Cycles	Circle	24	24	123.1666	1014	2.9151
Cityscape	4k	Simulated Sunlight	Cycles	Circle	36	36	234.6944	2760	3.0612
Cityscape	4k	Simulated Sunlight	Cycles	Circle	48	48	336.8750	4962	3.2587
Cityscape	4k	Simulated Sunlight	Cycles	Square	12	12	215.0000	931	2.7712
Cityscape	4k	Simulated Sunlight	Cycles	Square	24	24	209.5000	1696	2.9646
Cityscape	4k	Simulated Sunlight	Cycles	Square	36	36	351.9166	3900	3.2484
Cityscape	4k	Simulated Sunlight	Cycles	Square	48	48	450.0000	6354	3.3994
Cityscape	4k	Simulated Sunlight	Evee	Circle	12	12	63.6666	332	2.3012
Cityscape	4k	Simulated Sunlight	Evee	Circle	24	24	110.9583	963	2.7653
Cityscape	4k	Simulated Sunlight	Evee	Circle	36	36	211.8888	2613	2.9192
Cityscape	4k	Simulated Sunlight	Evee	Circle	48	48	297.7708	4597	3.1092
Cityscape	4k	Simulated Sunlight	Evee	Square	12	12	189.4166	845	2.6899
Cityscape	4k	Simulated Sunlight	Evee	Square	24	24	181.7500	1548	2.8178
Cityscape	4k	Simulated Sunlight	Evee	Square	36	36	309.0555	3635	3.0607
Cityscape	4k	Simulated Sunlight	Evee	Square	48	48	390.7916	5806	3.2307

A1.1.11 Phase 4 Gradient Teapot 1080p Cityscape Experiment Results

Table A1.9: Phase 4 gradient teapot 1080p cityscape experiment results

Model	Resolution	Lighting	Render Engine	Placement Pattern	#Images	#Images Registered	Mean Keypoints	#3D Points	Mean Track Length
Cityscape	1080p	Simulated Sunlight	Cycles	Circle	12	12	20.3333	118	2.0677
Cityscape	1080p	Simulated Sunlight	Cycles	Circle	24	24	45.2083	439	2.4715
Cityscape	1080p	Simulated Sunlight	Cycles	Circle	36	36	109.0555	1435	2.7358
Cityscape	1080p	Simulated Sunlight	Cycles	Circle	48	48	160.1250	2705	2.8414
Cityscape	1080p	Simulated Sunlight	Cycles	Square	12	12	112.0000	499	2.6933
Cityscape	1080p	Simulated Sunlight	Cycles	Square	24	24	102.2916	900	2.7277
Cityscape	1080p	Simulated Sunlight	Cycles	Square	36	36	181.4166	2224	2.9366
Cityscape	1080p	Simulated Sunlight	Cycles	Square	48	48	226.2916	3570	3.0425
Cityscape	1080p	Simulated Sunlight	Evee	Circle	12	12	20.0833	114	2.1140
Cityscape	1080p	Simulated Sunlight	Evee	Circle	24	24	37.7083	384	2.3567
Cityscape	1080p	Simulated Sunlight	Evee	Circle	36	36	91.1111	1263	2.5969
Cityscape	1080p	Simulated Sunlight	Evee	Circle	48	48	142.4375	2532	2.7002
Cityscape	1080p	Simulated Sunlight	Evee	Square	12	12	106.6666	492	2.6016
Cityscape	1080p	Simulated Sunlight	Evee	Square	24	24	89.8750	824	2.6177
Cityscape	1080p	Simulated Sunlight	Evee	Square	36	36	162.9444	2095	2.7999
Cityscape	1080p	Simulated Sunlight	Evee	Square	48	48	192.3541	3214	2.8727

A1.2 capture.sh

```
#!/usr/bin/env bash

# Global variables
scriptLocation="$( dirname -- "${BASH_SOURCE[0]}" )"
modelsPath=""
captureScriptPath=""
workspacePath=""

function showUsage() {
    echo "A script to conduct sfm evaluation experiments."
    echo "Usage: experiments.sh [-h] [-m modelsPath] \
        [-c captureScript] [-w workspacePath]"
    echo "Flags: "
    echo "      -h : Help, gives usage instructions."
    echo "      -m : Path to the folder containing the models to be reconstructed."
    echo "      -c : Path to the capture script to be used."
    echo "      -w : Path to the workspace location where results will be stored."
    exit 1
}

# captureModel(
    modelPath,
```

```
    captureScriptPath,
    modelOfInterestName,
    numberOfCamerasPerOrbit,
    cameraPlacementPattern,
    orbitHeights,
    outputDir)
function captureModel() {
    modelPath=$1
    captureScriptPath=$2
    modelOfInterestName=$3
    numberOfCamerasPerOrbit=$4
    cameraPlacementPattern=$5
    orbitHeights=$6
    outputDir=$7
    echo $'\n'
    echo "----------------------------------------------------------------------"
    echo "Capturing model with the following setup"
    echo "Model Path : ${modelPath}"
    echo "Capture Script Path : ${captureScriptPath}"
    echo "Model of interest name : ${modelOfInterestName}"
    echo "Number of cameras per orbit : ${numberOfCamerasPerOrbit}"
    echo "Camera placement pattern : ${cameraPlacementPattern}"
    echo "Orbit heights : ${orbitHeights}"
    echo "Output directory : ${outputDir}"
    echo "----------------------------------------------------------------------"
    echo $'\n'
    blender -b "${modelPath}" \
        --python "${captureScriptPath}" -- \
        "${modelOfInterestName}" "${numberOfCamerasPerOrbit}" \
        "${orbitHeights}" "${cameraPlacementPattern}" "${outputDir}"
}

while getopts "hm:c:w:" flag; do
  case "${flag}" in
    m) modelsPath="${OPTARG}" ;;
    c) captureScriptPath="${OPTARG}" ;;
    w) workspacePath="${OPTARG}" ;;
    h) showUsage
       exit 1 ;;
```

```
  esac
done

if [[ "${modelsPath}" == "" ]]; then
    echo "Missing models path. Please supply the path to \
        the models you wish to evaluate."
    exit 0
fi

if [[ "${captureScriptPath}" == "" ]]; then
    echo "Missing capture script path. Please supply the \
        path to the capture script you wish to use."
    exit 0
fi

if [[ "${workspacePath}" == "" ]]; then
    echo "Missing workspace path. Please supply the path \
        where you want the outputs to be stored."
    exit 0
fi

# Tidy up paths
modelsPath=${modelsPath%/}
captureScriptPath=${captureScriptPath%/}
workspacePath=${workspacePath%/}
echo $'\n'
echo "----------------------------------------------------------------------"
echo "Models Path : ${modelsPath}"
echo "Capture Script Path : ${captureScriptPath}"
echo "Workspace Path : ${workspacePath}"
echo "----------------------------------------------------------------------"
echo $'\n'

for model in "${modelsPath}"/*.blend; do
    modelName=${model##*/}
    outputPath="${workspacePath}/${modelName%.blend}"
    echo "Processing model : ${model}"
    echo "Model Name : ${modelName}"
```

```
    captureModel "${model}" "${captureScriptPath}" \
        "Focus" 4 "circle" '[0,4,8]' "${outputPath}-Circle-4"
    captureModel "${model}" "${captureScriptPath}" \
        "Focus" 8 "circle" '[0,4,8]' "${outputPath}-Circle-8"
    captureModel "${model}" "${captureScriptPath}" \
        "Focus" 12 "circle" '[0,4,8]' "${outputPath}-Circle-12"
    captureModel "${model}" "${captureScriptPath}" \
        "Focus" 16 "circle" '[0,4,8]' "${outputPath}-Circle-16"
    captureModel "${model}" "${captureScriptPath}" \
        "Focus" 4 "square" '[0,4,8]' "${outputPath}-Square-4"
    captureModel "${model}" "${captureScriptPath}" \
        "Focus" 8 "square" '[0,4,8]' "${outputPath}-Square-8"
    captureModel "${model}" "${captureScriptPath}" \
        "Focus" 12 "square" '[0,4,8]' "${outputPath}-Square-12"
    captureModel "${model}" "${captureScriptPath}" \
        "Focus" 16 "square" '[0,4,8]' "${outputPath}-Square-16"
done

echo "Execution Complete!"
```

A1.3 automation-script.py

```
import os
import sys
import argparse
import math
import time
import json
import enum
import bpy
import json
import random
import mathutils
import numpy as np

DEFAULT_MODEL_OF_INTEREST_NAME = "Focus"

# Enumeration to represent different camera placement patterns.
class Pattern(enum.Enum):
```

```
    circle = 1
    square = 2
    random = 3

# Helper function to avoid KeyErrors with dictionaries.
def is_key_in_dictionary(dictionary, search_key):
    for key in dictionary:
        if(key == search_key):
            return True
    return False

# Write a dictionary to a file.
def write_dict_to_file(file_path, file_contents):
    try:
        write_file = open(file_path,'w')
        write_file.write(json.dumps(file_contents))
        write_file.close()
    except Exception as e:
        print(e)

# Read a dictionary from a file.
def read_dict_from_file(file_path):
    try:
        read_file = open(file_path,'r')
        file_contents = json.loads(read_file.read())
        read_file.close()
        return file_contents
    except Exception as e:
        print(e)

# Negate a mathutils 3x3 matrix.
def negate_3x3_matrix(matrix):
    negated_matrix = matrix.copy()
    for i in range(0, 3):
        for j in range(0, 3):
            negated_matrix[i][j] = negated_matrix[i][j] * -1
    return negated_matrix

# Retreives the camera intrinsic parameters.
```

```
def get_camera_intrinsic_parameters_json(camera, scene):
    try:
        camera_intrinsic_parameters = {}
        camera_intrinsic_parameters['render_resolution_x'] =
            scene.render.resolution_x
        camera_intrinsic_parameters['render_resolution_y'] =
            scene.render.resolution_y
        # focal_length_pixels = focal_length_mm / sensor_width * max_size.
        camera_intrinsic_parameters['focal_length_pixels'] =
            camera.lens / camera.sensor_width * scene.render.resolution_x
        camera_intrinsic_parameters['focal_point_x'] =
            scene.render.resolution_x / 2
        camera_intrinsic_parameters['focal_point_y'] =
            scene.render.resolution_y / 2
        camera_intrinsic_parameters['angle'] = camera.angle
        camera_intrinsic_parameters['angle_x'] = camera.angle_x
        camera_intrinsic_parameters['angle_y'] = camera.angle_y
        camera_intrinsic_parameters['clip_end'] = camera.clip_end
        camera_intrinsic_parameters['clip_start'] = camera.clip_start
        camera_intrinsic_parameters['display_size'] = camera.display_size
        camera_intrinsic_parameters['dof'] = str(camera.dof)
        camera_intrinsic_parameters['lens'] = camera.lens
        camera_intrinsic_parameters['lens_unit'] = camera.lens_unit
        camera_intrinsic_parameters['ortho_scale'] = camera.ortho_scale
        camera_intrinsic_parameters['sensor_fit'] = camera.sensor_fit
        camera_intrinsic_parameters['sensor_height'] = camera.sensor_height
        camera_intrinsic_parameters['sensor_width'] = camera.sensor_width
        camera_intrinsic_parameters['shift_x'] = camera.shift_x
        camera_intrinsic_parameters['shift_y'] = camera.shift_y
        return camera_intrinsic_parameters
    except Exception as e:
        print(e)
        return {}

# Takes a camera object as a parameter and converts its location and
# rotation into COLMAP's coordinate system.
def get_colmap_view_space_details_json(object):
    try:
        world_matrix = object.matrix_world
```

```
        blender_to_colmap_rotation = np.diag([1,-1,-1])
        # Convert from blender world space to view space.
        blender_world_translation, blender_world_rotation, blender_world_scale =
            world_matrix.decompose()
        blender_view_rotation = blender_world_rotation.to_matrix().transposed()
        blender_view_translation =
            -1.0 * blender_view_rotation @ blender_world_translation
        # Convert from blender view space to colmap view space.
        colmap_view_rotation = blender_to_colmap_rotation @ blender_view_rotation
        colmap_view_rotation_quaternion =
            mathutils.Matrix(colmap_view_rotation).to_quaternion()
        colmap_view_translation =
            blender_to_colmap_rotation @ blender_view_translation
        return {
                "name" : object.name,
                "x_pos" : colmap_view_translation[0],
                "y_pos" : colmap_view_translation[1],
                "z_pos" : colmap_view_translation[2],
                "w_rotation" : colmap_view_rotation_quaternion.w,
                "x_rotation" : colmap_view_rotation_quaternion.x,
                "y_rotation" : colmap_view_rotation_quaternion.y,
                "z_rotation" : colmap_view_rotation_quaternion.z
            }
    except Exception as e:
        print(e)
        return {}

# Takes a camera object as a parameter and retreives its location and
# rotation in Blender's world coordinate system.
def get_world_space_details_json(object):
    try:
        world_matrix = object.matrix_world
        blender_world_translation, blender_world_rotation, blender_world_scale =
            world_matrix.decompose()
        return {
                "name" : object.name,
                "x_pos" : blender_world_translation[0],
                "y_pos" : blender_world_translation[1],
                "z_pos" : blender_world_translation[2],
```

```
                "w_rotation" : blender_world_rotation.w,
                "x_rotation" : blender_world_rotation.x,
                "y_rotation" : blender_world_rotation.y,
                "z_rotation" : blender_world_rotation.z
            }
    except Exception as e:
        print(e)
        return {}

# Takes a json of camera intrinsics and writes them to a .txt file
# in the format expected by COLMAP. Populates the cameras.txt file for manual
# camera input to COLMAP.
def write_camera_intrinsics_to_text_file(file_path, camera_details):
    try:
        write_file = open(file_path,'w')
        formatted_string = "1 SIMPLE_RADIAL {} {} {} {} {} 0.000000".format(
            str(camera_details['render_resolution_x']),
            str(camera_details['render_resolution_y']),
            "{:.6f}".format(camera_details['focal_length_pixels']),
            "{:.6f}".format(camera_details['focal_point_x']),
            "{:.6f}".format(camera_details['focal_point_y']),
            )
        write_file.writelines(formatted_string)
        write_file.close()
    except Exception as e:
        print(e)

# Takes a list of view space camera pose jsons and writes them to a .txt file
# in the format expected by COLMAP. Populates the images.txt file for manual
# pose input to COLMAP.
def write_camera_poses_to_text_file(file_path, camera_details):
    try:
        write_file = open(file_path,'w')
        camera_details.sort(key=lambda x : x['name'].strip('progcam'))
        for i, camera in enumerate(camera_details):
            formatted_string = "{} {} {} {} {} {} {} {} 1 {}.png\n".format(
                str(i + 1),
                str(camera['w_rotation']),
                str(camera['x_rotation']),
```

```
                str(camera['y_rotation']),
                str(camera['z_rotation']),
                str(camera['x_pos']),
                str(camera['y_pos']),
                str(camera['z_pos']),
                camera['name'])
            write_file.writelines(formatted_string)
            write_file.writelines("\n")
        write_file.close()
    except Exception as e:
        print(e)

# Takes a list of world space camera pose jsons and writes them to a .txt file
# in the format expected by COLMAP. Populates a text file which COLMAP can interpret
# as additional geo location data about the cameras.
def write_camera_world_space_poses_to_text_file(file_path, camera_details):
    try:
        write_file = open(file_path,'w')
        camera_details.sort(key=lambda x : x['name'].strip('progcam'))
        for i, camera in enumerate(camera_details):
            formatted_string = "{}.png {} {} {}\n".format(
                camera['name'],
                str(camera['x_pos']),
                str(camera['y_pos']),
                str(camera['z_pos']),
                )
            write_file.writelines(formatted_string)
            write_file.writelines("\n")
        write_file.close()
    except Exception as e:
        print(e)

# Takes a camera object and renders a still image of what is visible
# in the camera viewport.
def render_still_image(scene, camera, camera_name, render_path):
    try:
        # Capture still image from camera we placed.
        scene.camera = camera
        scene.render.image_settings.file_format='PNG'
```

```
        scene.render.filepath = "{}/{}".format(render_path, camera_name)
        bpy.ops.render.render(write_still=1)
    except Exception as e:
        print(e)

# Generates a list of 3D camera positions following a circular pattern.
def generate_camera_positions_circle(number_of_cameras, orbit_heights):
    try:
        camera_coords = []
        # Setup boundary conditions.
        angle_increase_step = 360 / number_of_cameras
        for orbit_height in orbit_heights:
            angle = 0
            for i in range(number_of_cameras):
                camera_coords.
                    append(
                        (20 * math.cos(angle),
                        20 * math.sin(angle),
                        Sorbit_height))
                angle += math.radians(angle_increase_step)
        return camera_coords
    except Exception as e:
        print(e)
        return []

# Generates a list of 3D camera positions following a square pattern.
def generate_camera_positions_square(number_of_cameras, orbit_heights):
    try:
        camera_coords = []
        # Setup boundary conditions.
        x_y_scene_boundary = 20
        number_of_sides = 4
        side_length = x_y_scene_boundary * 2
        for orbit_height in orbit_heights:
            # Place corner cameras.
            camera_coords.
                append((x_y_scene_boundary, x_y_scene_boundary, orbit_height))
            camera_coords.
                append((x_y_scene_boundary, -x_y_scene_boundary, orbit_height))
```

```
            camera_coords.
                append((-x_y_scene_boundary, x_y_scene_boundary, orbit_height))
            camera_coords.
                append((-x_y_scene_boundary, -x_y_scene_boundary, orbit_height))
            # Place remaining cameras.
            remaining_cameras = number_of_cameras - 4
            number_of_cameras_per_side = int(remaining_cameras / number_of_sides)
            camera_placement_step = side_length / (number_of_cameras_per_side + 1)
            for i in range(1, number_of_cameras_per_side+1):
                # -y cameras.
                camera_coords.
                    append(
                        (-x_y_scene_boundary + i*camera_placement_step,
                        -x_y_scene_boundary,
                        orbit_height))
                # +y cameras.
                camera_coords.
                    append(
                        (-x_y_scene_boundary + i*camera_placement_step,
                        x_y_scene_boundary,
                        orbit_height))
                # -x cameras.
                camera_coords.
                    append(
                    (-x_y_scene_boundary,
                    -x_y_scene_boundary + i*camera_placement_step,
                    orbit_height))
                # +x cameras.
                camera_coords.
                    append(
                        (x_y_scene_boundary,
                        -x_y_scene_boundary + i*camera_placement_step,
                        orbit_height))
        return camera_coords
    except Exception as e:
        print(e)
        return []

# Generates a list of 3D camera positions with cameras placed randomly
```

```
# along the supplied orbits.
def generate_camera_positions_random(number_of_cameras, orbit_heights):
    try:
        camera_coords = []
        # Setup boundary conditions.
        x_y_scene_boundary = 20
        for orbit_height in orbit_heights:
            for i in range(number_of_cameras):
                camera_coords.append(
                    (x_y_scene_boundary * random.random(),
                    x_y_scene_boundary * random.random(),
                    orbit_height))
        return camera_coords
    except Exception as e:
        print(e)
        return []

# Generates a list of 3D camera positions following a random pattern.
def generate_camera_positions_full_random(number_of_cameras):
    camera_coords = []
    # Setup boundary conditions.
    x_y_scene_boundary = 20
    z_scene_boundary = 8
    for i in range(number_of_cameras):
        camera_coords.append(
            (x_y_scene_boundary * random.random(),
            x_y_scene_boundary * random.random(),
            z_scene_boundary * random.random()))
    return camera_coords

# Places cameras around the scene using the list of camera coords and captures
# an image from each camera.
def capture_scene(model_of_interest_name, render_file_path, camera_coords):
    placed_camera_world_space_details = []
    placed_camera_view_space_details = []
    camera_intrinsic_parameters = []
    for i, camera_location in enumerate(camera_coords):
        camera_name = "progcam" + str(i)
        print("Camera coords: x={}, y={}, z={}".
```

```
            format(camera_location[0], camera_location[1], camera_location[2]))
        camera_position_info = capture_single_camera(
            model_of_interest_name, camera_name, render_file_path, camera_location)
        camera_intrinsic_parameters.
            append(camera_position_info["camera_intrinsic_parameters"])
        placed_camera_view_space_details.
            append(camera_position_info["colmap_view_space"])
        placed_camera_world_space_details.
            append(camera_position_info["blender_world_space"])
    return {
        "camera_intrinsic_parameters":camera_intrinsic_parameters,
        "placed_camera_view_space_details":placed_camera_view_space_details,
        "placed_camera_world_space_details":placed_camera_world_space_details
        }

# Creates a new file in the specified path.
def touch(new_file_path, file_name):
    try:
        output_path = os.path.join(new_file_path, file_name)
        open(output_path, "w").close()
    except Exception as e:
        print(e)

# Outputs both intrinsic and extrinsic camera parameters to files for use with
# COLMAP and to peform additional analysis.
def output_scene_details(workspace_path, scene_details):
    output_path = os.path.join(workspace_path, "ground_truth")
    write_dict_to_file(output_path + "/camera_intrinsic_parameters.json",
        scene_details['camera_intrinsic_parameters'])
    write_dict_to_file(output_path + "/camera_positions_view_space.json",
        scene_details['placed_camera_view_space_details'])
    write_dict_to_file(output_path + "/camera_positions_world_space.json",
        scene_details['placed_camera_world_space_details'])
    write_camera_intrinsics_to_text_file(output_path + "/cameras.txt",
        scene_details['camera_intrinsic_parameters'][0])
    write_camera_world_space_poses_to_text_file(output_path + "/geo_location.txt",
        scene_details['placed_camera_world_space_details'])
    write_camera_poses_to_text_file(output_path + "/images.txt",
        scene_details['placed_camera_view_space_details'])
```

```
    touch(output_path, "points3D.txt")

# Places and captures the image from a single camera at the specified location.
def capture_single_camera(
    model_of_interest_name, camera_name, output_path, location):
    # Setup camera info.
    target_obj = bpy.data.objects[model_of_interest_name]
    # Create camera.
    print("Creating new camera : " + camera_name)
    cam_data = bpy.data.cameras.new(camera_name)
    camera_obj = bpy.data.objects.new(camera_name, cam_data)
    # Get the camera instrinic parameters.
    camera_intrinsic_parameters =
        get_camera_intrinsic_parameters_json(cam_data, bpy.context.scene)
    camera_obj.location=location
    print("Camera location : ", location)
    bpy.context.collection.objects.link(camera_obj)
    # Hide camera so it does not appear in the viewport of other cameras.
    camera_obj.hide_set(True)
    # Add tracking constraint on the object of interest.
    constraint = camera_obj.constraints.new(type='TRACK_TO')
    constraint.target = target_obj
    # Capture still image from camera we placed.
    render_still_image(bpy.context.scene, camera_obj, camera_name, output_path)
    # Remove the tracking constraint from the camera for manipulation.
    world_matrix = camera_obj.matrix_world.copy()
    camera_obj.constraints.remove(constraint)
    camera_obj.matrix_world = world_matrix
    # Get the camera coordinates and rotation.
    camera_colmap_view_space_details =
        get_colmap_view_space_details_json(camera_obj)
    camera_world_space_details = get_world_space_details_json(camera_obj)
    # Remove the camera from the scene.
    bpy.data.objects.remove(camera_obj, do_unlink=True)
    return {
                "colmap_view_space":camera_colmap_view_space_details,
                "blender_world_space":camera_world_space_details,
                "camera_intrinsic_parameters":camera_intrinsic_parameters,
            }
```

```
# Places a specified number of cameras around the scene following a set pattern.
def place_cameras(model_of_interest_arg,
    workspace_path, number_of_cameras, placement_pattern, orbit_heights):
    model_of_interest_name = DEFAULT_MODEL_OF_INTEREST_NAME
    if (model_of_interest_arg != None and model_of_interest_arg != ""):
        model_of_interest_name = model_of_interest_arg
    print("Model of interest : ", model_of_interest_name)
    render_file_path = os.path.join(workspace_path, "images")
    camera_positions = []
    if placement_pattern == Pattern.circle:
        camera_positions =
            generate_camera_positions_circle(number_of_cameras, orbit_heights)
    elif placement_pattern == Pattern.square:
        camera_positions =
            generate_camera_positions_square(number_of_cameras, orbit_heights)
    else:
        camera_positions =
            generate_camera_positions_random(number_of_cameras, orbit_heights)
    scene_details =
        capture_scene(model_of_interest_name, render_file_path, camera_positions)
    output_scene_details(workspace_path, scene_details)
    print("Completed placing cameras.")

def create_workspace_paths(workspace_path):
     # Create paths required for project.
    paths = []
    paths.append(os.path.join(workspace_path, "images"))
    paths.append(os.path.join(workspace_path, "ground_truth"))
    paths.append(os.path.join(workspace_path, "sparse"))
    paths.append(os.path.join(workspace_path, "dense"))
    for path in paths:
        if os.path.exists(path) == False:
            os.makedirs(path)

def parse_camera_placement_pattern_string(camera_placement_pattern_string):
    if camera_placement_pattern_string == "square":
        return Pattern.square
    elif camera_placement_pattern_string == "circle":
```

```
        return Pattern.circle
    else:
        return Pattern.random

def main():
    argv = sys.argv
    argv = argv[argv.index("--") + 1:]  # get all args after "--"
    print(argv)  # --> ['example', 'args', '123']

    if (len(argv) != 5):
        print("An incorrect number of arguments was supplied.")
        exit(1)

    # Update the camera placement functions to take a list of orbit heights as
    # a parameter

    model_of_interest_name = argv[0]
    number_of_cameras_per_orbit = int(argv[1])
    orbit_heights_string = argv[2]
    orbit_heights = json.loads(orbit_heights_string)
    camera_placement_pattern_string = argv[3]
    camera_placement_pattern =
        parse_camera_placement_pattern_string(camera_placement_pattern_string)
    workspace_path = argv[4]

    create_workspace_paths(workspace_path)
    place_cameras(
        model_of_interest_name,
        workspace_path,
        number_of_cameras_per_orbit,
        camera_placement_pattern,
        orbit_heights)

    print("BLENDER Execution Complete!")

if __name__ == '__main__':
    main()
```

A1.4 reconstruct.sh

```
#!/usr/bin/env bash

# Global variables
scriptLocation="$( dirname -- "${BASH_SOURCE[0]}" )"
workspacePath=""

function showUsage() {
    echo "A script to conduct sfm evaluation experiments."
    echo "Usage: experiments.sh [-h] [-w workspacePath]"
    echo "Flags: "
    echo "      -h : Help, gives usage instructions."
    echo "      -w : Path to the workspace location where results \
        will be stored."
    exit 1
}

# reconstructModel(inputDir)
function reconstructModel() {
    inputDir=$1
    echo "Reconstructing : ${inputDir}"
    colmap.bat feature_extractor \
        --database_path "${inputDir}\database.db" \
        --image_path "${inputDir}\images"
    colmap.bat exhaustive_matcher \
        --database_path "${inputDir}\database.db"
    #colmap.bat mapper \
    #    --database_path "${inputDir}\database.db" \
    #    --image_path "${inputDir}\images" \
    #    --output_path "${inputDir}\sparse"
    colmap.bat point_triangulator \
        --database_path "${inputDir}\database.db" \
        --image_path "${inputDir}\images" \
        --input_path "${inputDir}\ground_truth" \
        --output_path "${inputDir}\sparse"
    colmap.bat image_undistorter \
        --image_path "${inputDir}\images" \
        --input_path "${inputDir}\sparse" \
```

```
        --output_path "${inputDir}\dense" \
        --max_image_size 2000
    colmap.bat patch_match_stereo \
        --workspace_path "${inputDir}\dense" \
        --PatchMatchStereo.geom_consistency true
    colmap.bat stereo_fusion \
        --workspace_path "${inputDir}\dense" \
        --input_type geometric \
        --output_path "${inputDir}\dense\fused.ply"
    colmap.bat poisson_mesher \
        --input_path "${inputDir}\dense\fused.ply" \
        --output_path "${inputDir}\dense\meshed-poisson.ply" \
        --PoissonMeshing.trim 8
    colmap.bat delaunay_mesher \
        --input_path "${inputDir}\dense" \
        --output_path "${inputDir}\dense\meshed-delaunay.ply"
}

while getopts "hw:" flag; do
  case "${flag}" in
    w) workspacePath="${OPTARG}" ;;
    h) showUsage
       exit 1 ;;
  esac
done

if [[ "${workspacePath}" == "" ]]; then
    echo "Missing workspace path. Please supply the path to the directory \
        containing the models to be reconstructed."
    exit 0
fi

# Tidy up paths
workspacePath=${workspacePath%/}
echo $'\n'
echo "----------------------------------------------------------------------"
echo "Workspace Path : ${workspacePath}"
echo "----------------------------------------------------------------------"
echo $'\n'
```

```
for reconstructionPath in "${workspacePath}"/*/; do
    reconstructionPath=${reconstructionPath%/}
    reconstructionPath=$(echo "$reconstructionPath" | sed -r 's/[/]+/\\/g')
    reconstructModel "${reconstructionPath}"
done

echo "Execution Complete!"
```