FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Assignment of bachelor's thesis

Title:	3D point cloud from multi-camera system
Student:	Tomáš Reinhold
Supervisor:	Ing. Jan Buriánek
Study program:	Informatics
Branch / specialization:	Web and Software Engineering, specialization Computer Graphics
Department:	Department of Software Engineering
Validity:	until the end of summer semester 2022/2023

Instructions

The goal of this thesis is to reconstruct a scene from the video recordings captured by the multi-camera system into 3D animated point cloud. In the processing, benchmark setup will be created and capabilities of modern photogrammetry software will be tested on non-ideal conditions. The resulting point clouds will then be visualized.

Tasks:

1) Analyze current solutions and their capabilities for multi-camera systems
2) Propose a solution that utilizes existing 3D photogrammetry reconstruction software
3) Create a multi-camera benchmark setup
4) Create a pipeline that performs proposed solution (multi-camera system and photogrammetry system)
5) Evaluate results and discuss possible extensions or improvements

---

Electronically approved by Ing. Radek Richtr, Ph.D. on 11 February 2022 in Prague.

Bachelor's thesis

3D POINT CLOUD FROM MULTI-CAMERA SYSTEM

Tomáš Reinhold

Faculty of Information Technology
Department of Software Engineering
Supervisor: Ing. Jan Buriánek
May 10, 2022

Czech Technical University in Prague
Faculty of Information Technology

Contents

List of Figures

List of Tables

Firstly, I would like to thank my supervisor Ing. Jan Buriánek for valuable advice and great guidance throughout the process. Next, I would like to thank my family who supported me and provided me with the necessary equipment that was needed for this thesis.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended. In accordance with Article 46(6) of the Act, I hereby grant a nonexclusive authorization (license) to utilize this thesis, including any and all computer programs incorporated therein or attached thereto and all corresponding documentation (hereinafter collectively referred to as the "Work"), to any and all persons that wish to utilize the Work. Such persons are entitled to use the Work in any way (including for-profit purposes) that does not detract from its value. This authorization is not limited in terms of time, location and quantity. However, all persons that makes use of the above license shall be obliged to grant a license at least in the same scope as defined above with respect to each and every work that is created (wholly or in part) based on the Work, by modifying the Work, by combining the Work with another work, by including the Work in a collection of works or by adapting the Work (including translation), and at the same time make available the source code of such work at least in a way and scope that are comparable to the way and scope in which the source code of the Work is made available.

In Prague on May 10, 2022 .....

Abstract

3D reconstruction, especially photogrammetry is a common way of acquiring static 3D models of the real world. There exist multiple software solutions performing photogrammetry that process overlapping images, for example, taken from multi-camera systems. This thesis aims to create a new pipeline that utilizes already existing photogrammetry software solutions and multi-camera systems. This pipeline starts with synchronized image acquisition of the scene and ends with visualization of an animated point cloud. To understand the demands and capabilities of photogrammetry, a custom multi-camera benchmark system was created. This system was used for photogrammetry experiments and as a starting point for the introduced pipeline. The results show that it is possible to create an animated point cloud even when using a low-cost multi-camera system, like ours created. Using this pipeline on high-end multi-camera systems may introduce new interesting dynamic visualization of captured scene.

Keywords animated point cloud pipeline, multi-camera system, HIKVISION IP surveillance camera, photogrammetry, Meshroom, point cloud visualization

Abstrakt

3D rekonstrukce, zejména fotogrammetrie, je běžný způsob získávání statických 3D modelů reálného světa. Existuje celá řada softwarových řešení, která provádějí fotogrammetrii, jež zpracovává překrývající se snímky, pořízeny například multi-kamerovým systémem. Cílem této práce je vytvořit nový postup, který využívá existující fotogrammetrické softwarové řešení a multi-kamerové systémy. Tento postup začíná se synchronizovaným pořizováním snímků a končí vizualizací animovaného mračna bodů. Pro pochopení nároků a možností fotogrammetrie jsme sestavili vlastní multi-kamerový systém. Tento systém byl využit ve fotogrammetrických experimentech a jako výchozí bod našeho navrhnutého postupu. Výsledky ukazují, že animované mračno bodů je možné vytvořit i za použití levného multi-kamerového systému, jako je ten náš. Použití tohoto postupu na profesionálních multi-kamerových systémech by mohlo přinést zajímavou dynamickou vizualizaci snímané scény.

Klíčová slova postup pro získání animovaného mračna bodů, multi-kamerový systém, HIKVISION IP bezpečnostní kamera, fotogrammetrie, Meshroom, vizualizace mračna bodů

List of abbreviations

AC	Alternating Current
BA	Bundle Adjustment
CCD	Charge-Coupled Device
CLI	Command-Line Interface
CMOS	Complementary Metal–Oxide–Semiconductor
CPU	Central Processing Unit
CSV	Comma-Separated Values
CUDA	Compute Unified Device Architecture
DC	Direct Current
DSLR	Digital Single-Lens Reflex camera
FTP	File Transfer Protocol
GPU	Graphics Processing Unit
GUI	Graphical User Interface
IP	Internet Protocol
LED	Light Emitting Diode
MVS	Multi-View Stereo
NTP	Network Time Protocol
OS	Operating System
PnP	Perspective-n-Point
PoE	Power over Ethernet
RANSAC	Random Sample Consensus
RTSP	Real Time Streaming Protocol
SfM	Structure from Motion
SNTP	Simple Network Time Protocol
URL	Uniform Resource Locator
VITC	Vertical Interval Timecode

Chapter 1
Introduction

3D reconstruction has grown massively in the past few years. We can see it being used in the gaming industry, digitization of cultural heritage, visualizations, metaverses, and many others. Due to this growth, many software solutions and camera systems that perform such reconstructions were created and are still being improved to meet the diverse needs.

The most common reconstruction method is photogrammetry. In essence, it takes a set of overlapping images as input and reconstructs a 3D model from it. Generally, these 3D models are static, but what happens if we introduce time? Performing multiple reconstructions of the same scene in short time intervals gives us multiple static models that, if shown in the right order, could represent motion and animate the scene. This approach of reconstructing multiple models from video recordings is innovative.

Our main motivation is to introduce these animated reconstructions to already existing multi-camera systems. These multi-camera systems then obtain a new form of output (animated point cloud) with minimal additional cost. We want to focus on all systems, not only those made especially for photogrammetry.

In order to introduce a new form of output, we will first analyze and compare current multi-camera systems and photogrammetry software solutions. Then we will create our low-cost multi-camera system. We will experiment with reconstructing scenes with different objects and environments. These experiments will tell us which conditions work the best and how our multi-camera system performs overall. After that, we will implement an animated point cloud pipeline utilizing the current photogrammetry software solution, which allows us to automate the reconstruction process. Finally, we will create a simple application to visualize animated point clouds.

In the second chapter, we analyze key parts of the thesis solution. Starting with multi-camera systems, we look at cameras, synchronization, and calibration. Next, we look at the concept, requirements, and available software solutions for the photogrammetry pipeline. And finally, what we can use to visualize animated point clouds.

In the third chapter, we take advantage of the information we gathered in the second chapter. First, we simulate a multi-camera system and then set up our system with affordable resources. We describe each part of the system individually and how it affects the system as a whole. Then we propose our implementation of the animated point cloud pipeline and describe how each step was implemented.

In the fourth chapter, we evaluate experiments we performed with our multi-camera system and photogrammetry software solutions.

Chapter 2
Analysis

In this chapter, we investigate the individual parts needed in our pipeline. Starting with the multi-camera system, which is necessary for capturing image data sets that are later used. We study the requirements that we put on such a system and how we can achieve them. We also present a basic theoretical foundation with reference to more in-depth sources. In the following section, we introduce photogrammetry, how it works, and what software solutions that perform it already exist. In the last section, we explore the possible solutions or libraries that we can use to visualize an animated point cloud.

2.1 Multi-camera system

Since photogrammetry 2.2 requires many overlapping images captured from different positions, we consider only inward-looking cameras. Now we have two options when it comes to image acquisition.

The first option is to have one or more cameras that we move around the object (e.g., using a robotic arm, unmanned aerial vehicle) or eventually move the object in front of the cameras (e.g., turntable). Sometimes a combination of both is used, see the systems [1, 2]. This solution is useful when we have limited resources, mainly cameras, which are the most expensive elements of the whole system. With controlled motion, we can still preserve information about camera positions that can be later used to improve reconstruction. However, with this option, we cannot achieve image acquisition in the short time intervals that we need for later motion visualization, therefore, it is only suitable for static objects.

The second option, on which we focus mainly, is to have multiple cameras around the object that capture at the same time, see the system [3]. In photogrammetry, we usually want as many images as possible. However, when it comes to more images, we also need more cameras, and since quality reconstruction takes tens or hundreds of images, such a system can be rather costly. However, this option is faster and more reliable because we do not move the camera, therefore, there is less room for errors or imperfections. We choose this option over the first option because we can capture both static and dynamic objects.

All operations are usually controlled from one station (usually a personal computer), where images from all the cameras are also stored. For the second option, we need a powerful station because we store many images at once with every acquisition. Some systems even have substations to overcome the data transfer bottleneck. Assuming an 18 megapixel, 14-bit camera, we can easily calculate that the file size of one raw image will be around 31.5 MB. With 90 cameras, that is, approximately 2.8 GB per 1 acquisition.

2.1.1 Cameras

We can divide cameras into two groups, digital and analog. Since we are interested in computer vision, we focus only on digital cameras.

The cameras used in a typical multi-camera system built for photogrammetry [1, 2, 3] are digital single-lens reflex cameras (DSLR). They usually come with a large image sensor that provides high-resolution images. Unfortunately, these cameras are quite expensive, therefore, we will not use them in our benchmark setup. We aim toward cheaper cameras that put photogrammetry software solutions capabilities on their edge.

One of the cheaper cameras that were used while solving a similar application as ours are action cameras. The focus of this master thesis [4] was reconstructing human faces using a custom-built multi-camera system with ten GoPro Hero 7 Silver cameras.

Internet protocol (IP) surveillance cameras are also one of the cheaper solutions. These cameras aim for the lowest possible bandwidth and achieve it using compression standards such as H.264 or more recent H.265, which both utilize a discrete cosine transform [5]. High lossy compression and low resolution are the exact reason why these cameras are not typically used.

2.1.1.1 Image formation

Image is formed by the light reflected from objects that is focused through the camera optics on the camera sensor (film plane). The aperture modifies the amount of light that is passed on the sensor. The shutter then closes the light path to the image sensor to create the image from the collected light. The time representing how long the shutter was open is called the exposure time. The distance between the optical center of the lens and the camera sensor is called the focal length. Summarized from [6, pp. 184–195].

These mentioned parameters and many others have a crucial impact on image acquisition. Cameras allow us to change these parameters, so knowing them and how they impact the image is necessary. A detailed description is out of the scope of this work, so the reader can study these parameters before proceeding.

The most common image sensor used today is a complementary metal-oxide-semiconductor (CMOS). This sensor slowly replaced the charge-coupled device (CCD) sensor. Both of these matrix sensors respond to wavelengths in the range of 200 nm to 1100 nm (visible to near-infrared). Some of the CMOS advantages over CCD:

- lower manufacturing cost, power consumption, image noise
- higher frame rates, dynamic range

To capture color images with the sensor, we need to separate light into different wavelengths (red, green, blue). The most common technique is using the Bayer filter, which is a color filter directly on the sensor grid. The filter in Figure 2.1 is inspired by the human eye, therefore, it prioritizes green. The other, more expensive, method is to use three sensors, see Figure 2.2, each equipped with a different color filter. Based on [6, pp. 171–179].

2.1.1.2 Mathematical models

Cameras are mathematically represented by models. Basically, the camera projects a 3D real-world on the 2D image plane. The most common model is called the pinhole camera model. This model is only an approximation because the real model is rather hard to obtain due to imperfections of the optics and the whole capturing system. In Figure 2.3 we can see the real-world point X being mapped on the image plane as point x.

The general pinhole camera model can be represented by a matrix P, see 2.1. Where K represent internal camera parameters and $R \mid t$ represents external camera parameters. Parameters of matrix K are called intrinsic and consist of focal length f_x, f_y and principal point c_x, c_y.

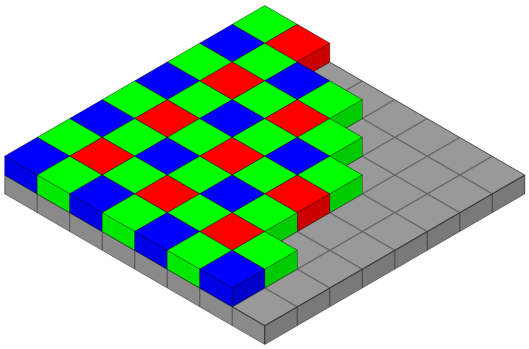


Figure 2.1 Bayer filter [7]

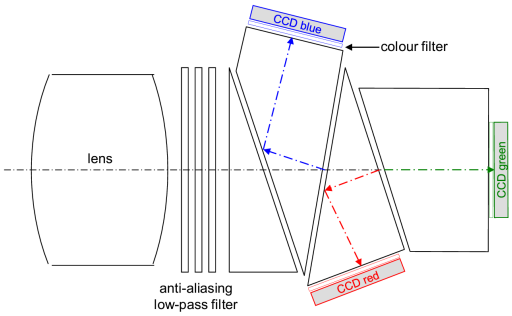


Figure 2.2 Three-sensor camera [6, p. 177]

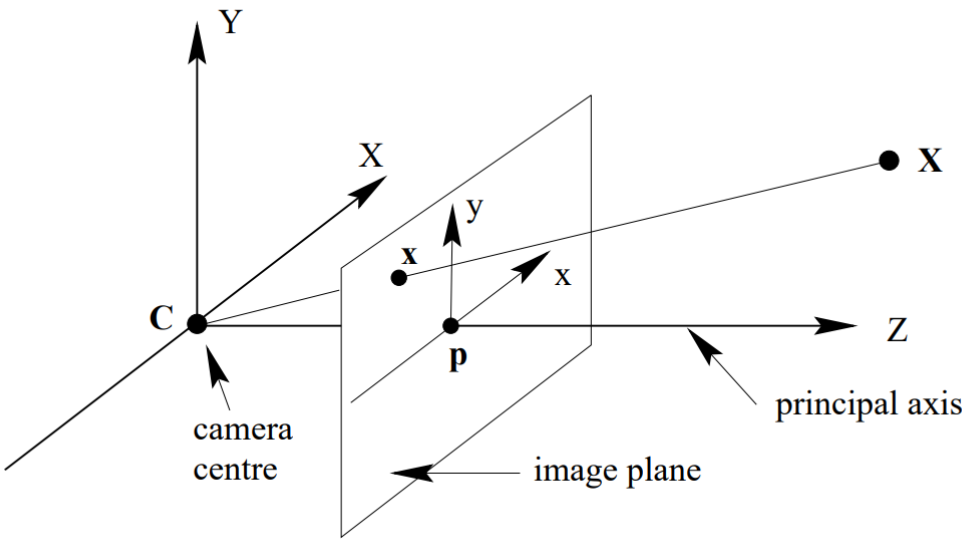


Figure 2.3 Pinhole camera model [8, p. 154]

External parameters consist of rotation R and translation t. A detailed description of why we have these parameters and what role they play can be found here [8, pp. 153–174].

$$P = K[R \mid t] = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_1 & r_2 & r_3 & t_1 \\ r_4 & r_5 & r_6 & t_2 \\ r_7 & r_8 & r_9 & t_3 \end{bmatrix} \tag{2.1}$$

It is important that the intrinsic matrix does not change if we move the camera around. It only changes if we modify the focal length (e.g., replace the lens, alter focus). Two identical cameras with identical lenses usually have different intrinsic matrices. This is caused by manufacturing imperfections.

The pinhole camera model is usually not precise enough because it does not deal with lens distortion. Distortion occurs when the camera projects straight lines as curved lines. One of the models used is the Brown-Conrady model [9]. This model models radial and tangential distortion, two of the most common distortions.

2.1.1.3 Calibration

To estimate the camera (projection) matrix, we usually capture multiple images of the calibration pattern, which is an object with known geometry (e.g., chessboard, asymmetrical circle pattern). After taking these images, we identify 3D scene points with image points, which is done mostly algorithmically [10, pp. 378–401]. With these correspondences, we can estimate the camera matrix and distortion coefficients.

2.1.1.4 Synchronization

In our application, where we want to capture dynamic objects, synchronization is critical. There are multiple ways to achieve image acquisition from all cameras simultaneously. Some are more precise but can only be used with specialized hardware. If we do not use synchronized cameras, the recordings slowly drift apart due to the usage of quartz crystals and other variables.

The first option is to use the generator locking technique (genlock). This solution is widely used in the broadcast world and is very robust. All cameras are usually physically connected to the unit that generates pulses that tell the cameras to produce a frame. This generator unit can be either an external device or one of the cameras. To use this method, we need cameras that support it, therefore, it is not very useful for already existing systems. This approach was used in the system [2].

The second option is to use the timecode (e.g., vertical interval timecode [VITC]). This option is useful in postproduction, where we can find time metadata in all recordings. VITC works by inserting frame numbers, timestamps, and correction bits into the vertical blanking interval of the video signal [11].

The third option utilizes the network time protocol (NTP) or its simpler version (SNTP) to correct the inner time of the cameras. This is useful for cameras where specialized hardware cannot be used. This approach was tested [12] and gave satisfactory results.

2.1.2 Light source

It is recommended to capture images of a well-lit scene with minimal casted shadows. Shadows may help reconstruction in some cases because they bring new points of interest, which we describe in 2.2.1.1. However, they appear in the resulting model, which is mostly unwanted.

We have two options to produce light. We can light the scene all the time as in systems [1, 2] or only while acquisition (camera external or internal flash) as in system [3]. A universal option is to produce light all the time because not all cameras support flash.

Choosing the correct light source is important because some may cause flickering, mainly those powered by alternating current (AC), because they turn on and off very quickly. So, ideally, we want a constant light source like the day light or sources powered by direct current (DC). Most multi-camera systems use DC light-emitting diode (LED) technology. But we still have to be careful because some LED drivers, which control dimming, use methods that interrupt the light source [13].

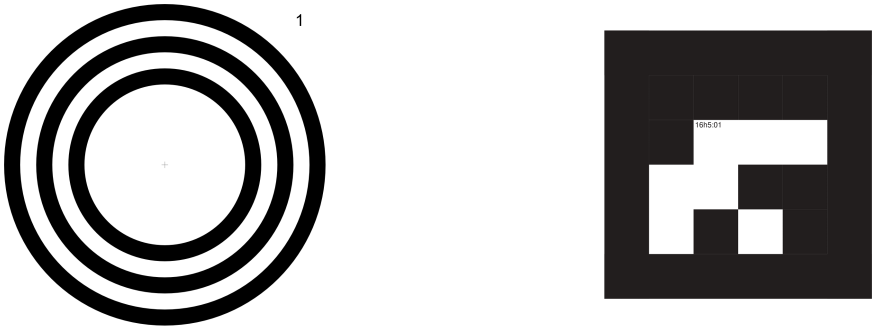
2.1.3 Fiducial markers

Among the objects we want to capture, we usually place reference markers at known positions in the scene. These markers help the camera alignment process and maintain scale and orientation in later image processing and reconstruction. Because there is no default marker, we usually have multiple types of marker so that we can use the captured images in multiple applications. We can divide markers into two groups, circular markers [14, 15] and square markers [16, 17].

CCTag marker [14] (see Figure 2.4a) is an example of a circular marker commonly used in photogrammetry applications. Its strong attribute is its robustness against motion blur. With CCTag, we can accurately locate the center of the markers, which can be used for tracking and motion estimation. We need more than one reference point for many applications, so we usually place at least four markers.

AprilTag marker [16] (see Figure 2.4b) is an example of a square marker, which we can use to find four reference points (corners) from only one marker. It is widely used in augmented reality and robotics.

Both of these markers store a few bits of information, such as identification. In addition to these two markers, there are many more. Another example of circular markers is TRIP [15] and an example of square markers is ARTag [17]. However, CCTag and AprilTag markers are mostly used in photogrammetry software solutions.



(a) CCTag marker **(b)** AprilTag marker

Figure 2.4 Example of fiducial markers

2.1.4 Existing systems

In this section, we present some already existing multi-camera systems with their specifications.

2.1.4.1 Multi-camera system for paleontology

This multi-camera system [1] is mainly used for reconstructing paleontological finds (objects of size 5 cm to 1.2 m). Camera movement is controlled with a custom script that sequentially reads a simple comma-separated values (CSV) file, which stores position, tilt, and whether the camera should capture or not for each camera and turntable rotation.

- 3 cameras: Canon EOS 700D with tilt, of which 2 move vertically, 1 horizontally
- lenses: Canon EF-S 18-55 mm
- motorized turntable
- led lights

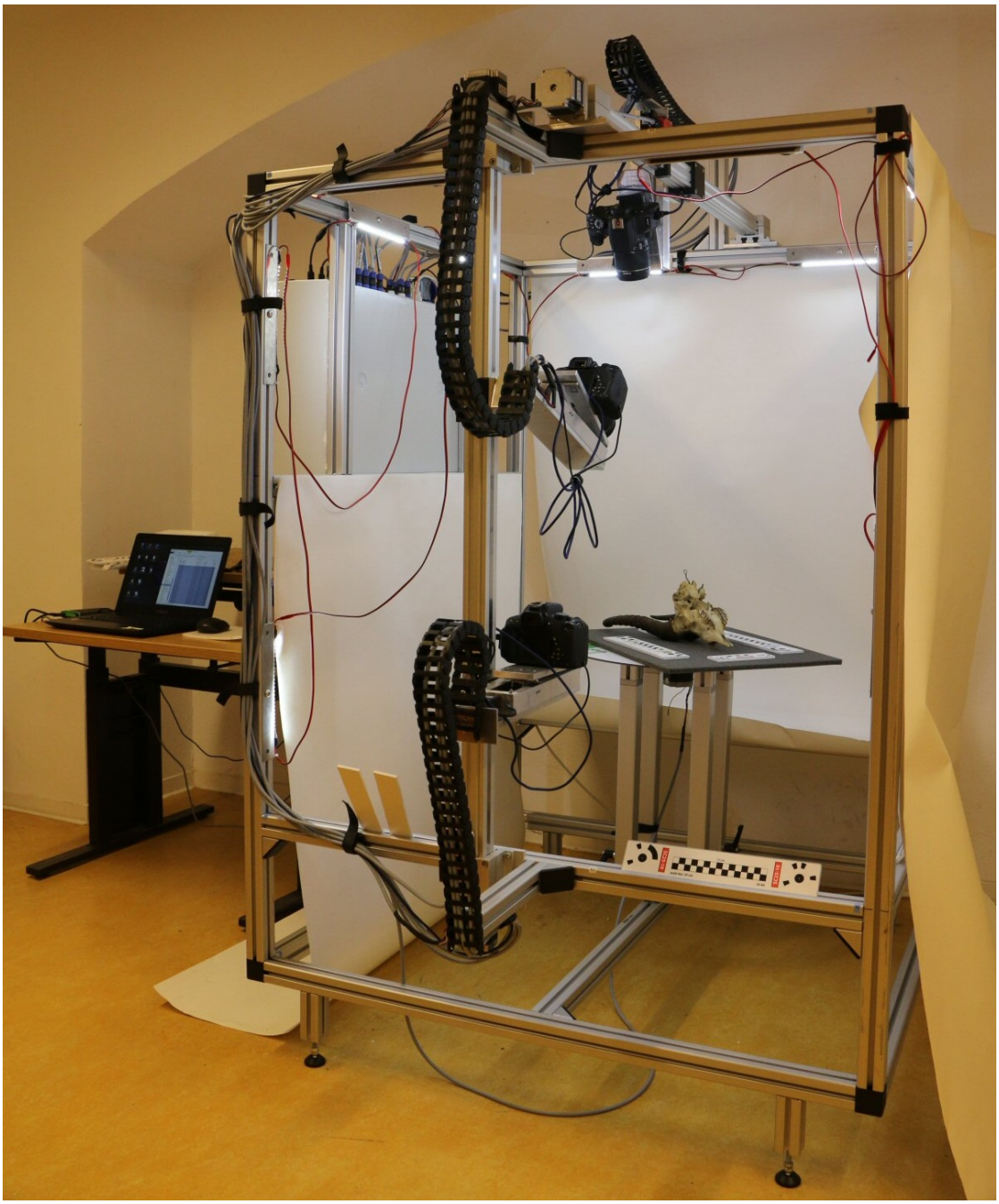


Figure 2.5 Multi-camera system for paleontology built by Technical University Berlin [18]

2.1.4.2 Multi-camera system at University of Maryland

This multi-camera system [3] is capable of reconstructing larger objects, such as human bodies. We can see that this system has data projectors. They are used to project patterns on surfaces. Combining projection with cameras is a special case of photogrammetry called fringe-projection photogrammetry [19].

One recent project that uses this system is the time-lapse 3D reconstruction of plant growth, which is very similar to our idea of dynamic reconstruction. More details can be found in [20].

- 94 cameras: Canon EOS T5i
- lenses: 18-55 mm and 55-250 mm
- 8 flashes: Alienbees B 1600
- 8 data projectors
- 4 computers

Figure 2.6 Multi-camera system at University of Maryland [21]

2.1.4.3 Multi-camera system DI-one

This multi-camera system [2] was created for commerce. It is designed for robust reconstructions of small to medium objects (50 cm^3). It uses many proprietary solutions, and therefore not many specifications are published.

The interesting thing about this system is that it masks the reconstructed object by subtracting background before reconstruction and performs radiometric calibration to accurately represent the colors.

- 31 cameras: Sony, model unspecified
- lenses: unspecified
- motorized basement

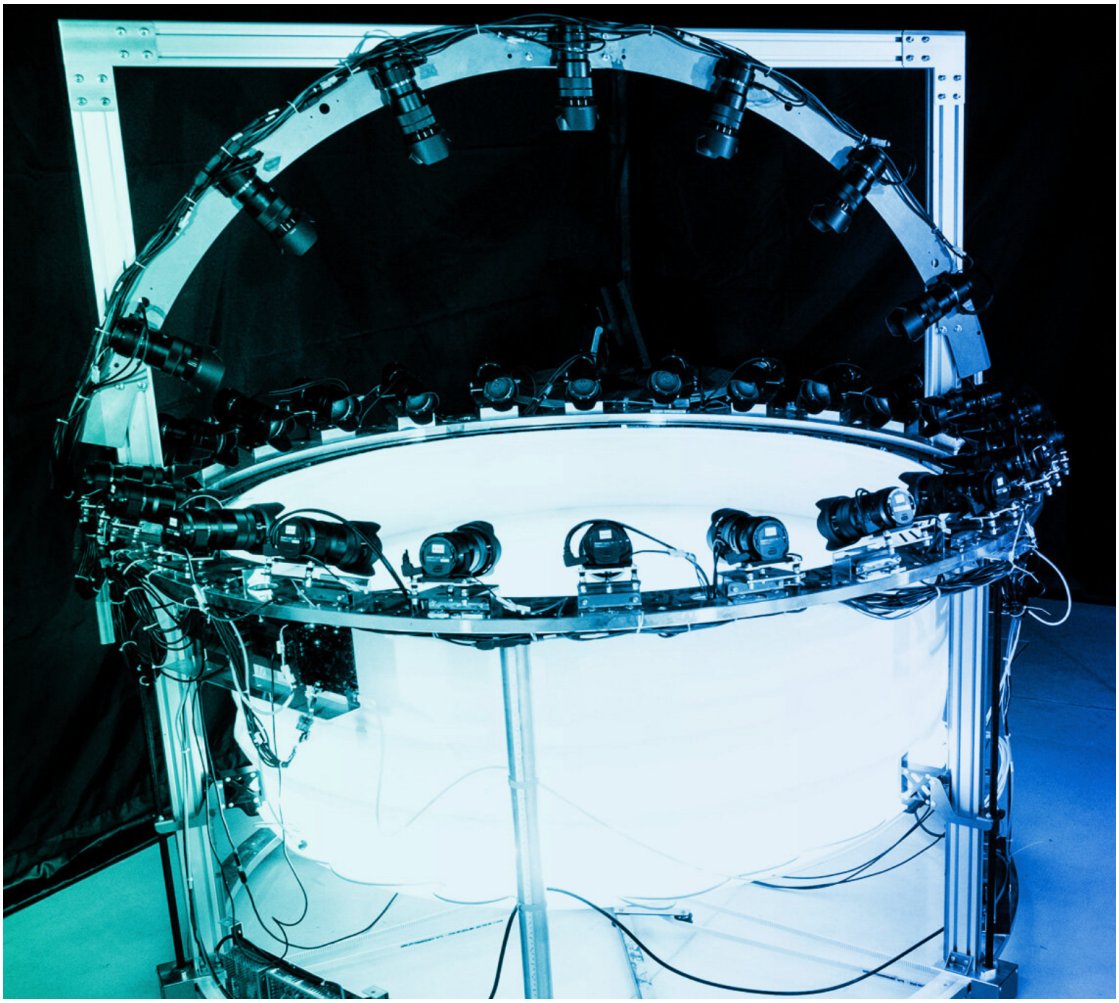


Figure 2.7 The DI-One multi-camera system [2, p. 786]

2.2 Photogrammetry

"Photogrammetry is the science of making measurements from photographs. It infers the geometry of a scene from a set of unordered photographies or videos. Photography is the projection of a 3D scene onto a 2D plane, losing depth information. The goal of photogrammetry is to reverse this process." [22]

According to photogrammetry categorization [6, p. 6] we focus in our work on a close-range, multi-image, digital, offline structure from motion photogrammetry. Furthermore, we consider only this category of photogrammetry.

The typical photogrammetry pipeline, according to [22] consists of two individual computer vision pipelines connected together. The first is called structure from motion (SfM), and the second is multi-view stereo.

2.2.1 Structure from motion

As its name suggests, it is a process of recovering the 3D structure from camera motion. From an unordered set of images, structure from motion creates a sparse point cloud and estimates camera poses and intrinsic camera parameters.

The widely used approach to SfM is incremental SfM [23], which is based on incrementally adding images to reconstruction, see Figure 2.8. Some other approaches are global [24, 25], hierarchical [26, 27, 28], multi-stage [29].

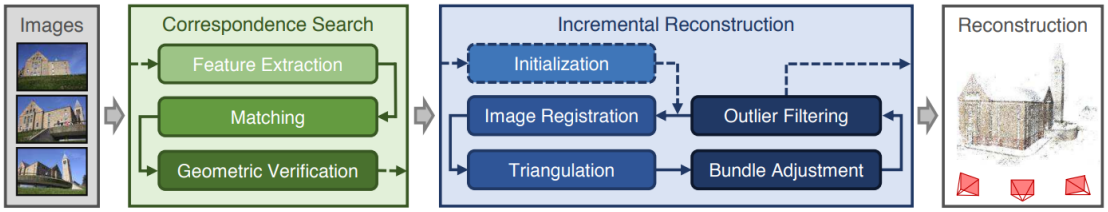


Figure 2.8 Incremental structure from motion [23, p. 4105]

We describe the parts of the incremental SfM pipeline below fairly briefly. The curious reader may find more information on incremental SfM in the second chapter of [23] from which we mainly drew information.

2.2.1.1 Image features

Image feature can be described as an interesting part in an image such as shapes, corners, edges, blobs, ridges. Features are widely used in computer vision applications, for example, in machine learning and pattern recognition. The finding of image features is usually one of the first steps in the SfM pipeline.

We use feature detectors, which are algorithms that detect features in images. These features are then described using feature descriptors. Usually, each feature detector has its feature descriptor. The widely used feature detectors/descriptors are SIFT [30] and AKAZE [31]. Other detectors/descriptors with their comparison can be found here [32]. Our advantage is that we can use slower but better detectors, such as SIFT, which positively affect the reconstruction quality.

When the object we want to reconstruct does not contain many features, we can manually add interesting points, depending on the feature detector, such as a noisy texture. In Figure 2.9, we can see the SIFT features (blue squares) located on a rendered cube with a Musgrave texture.

2.2.1.2 Matching

Image matching is performed after feature extraction. We want to compare and eventually match the descriptors of one image with those of the others. The brute-force approach would be to compare all the descriptors of each pair of images, but some algorithms significantly speed up this process [33]. The result of this step are image pairs with the corresponding features.

2.2.1.3 Geometric verification

In this step, we verify if the matches we found correspond, therefore, if the images actually look at the same object. We try to estimate the transformation that maps the corresponding features of one image to another. The matched features of the previous step usually contain many outliers that we want to eliminate. A common approach is to apply random sample consensus (RANSAC) [34], which filters them. The output of this step is a graph, where the nodes represent images and the edges represent if the connected nodes are verified.

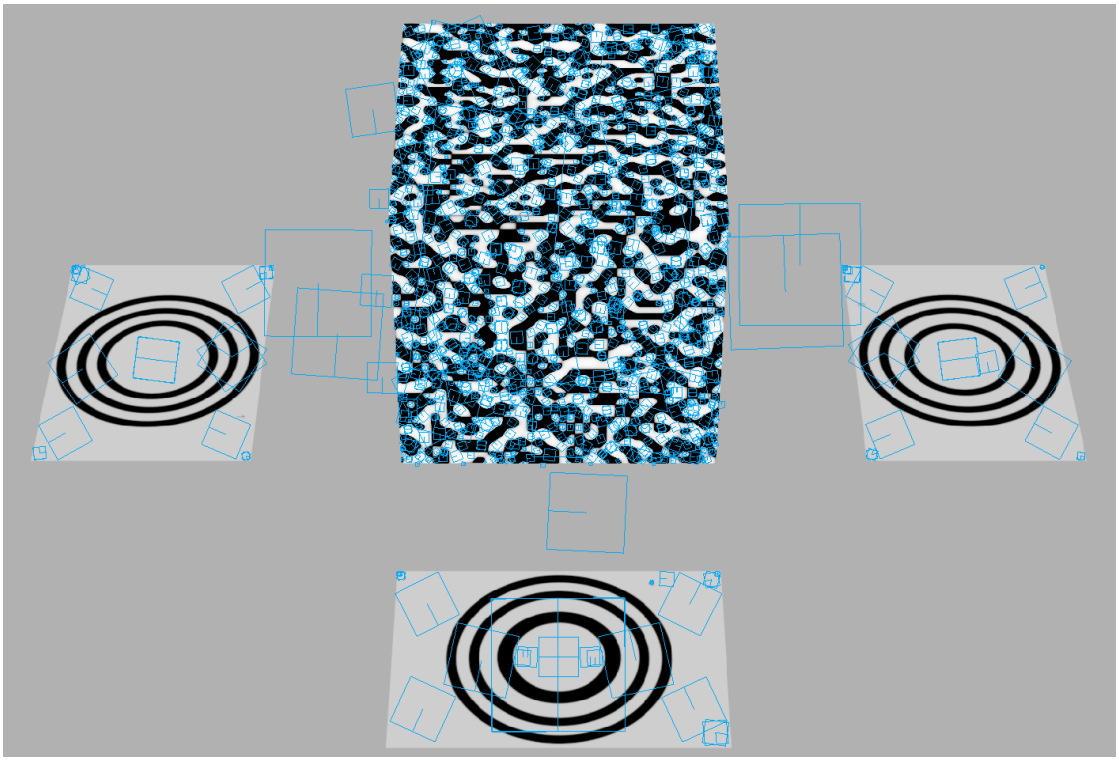


Figure 2.9 SIFT features on noisy texture

2.2.1.4 Initialization

With initialization, the reconstruction itself begins. One pair of images is selected [35] and a two-view reconstruction is performed. The selection of the initial pair has a significant impact on the rest of the reconstruction and partly defines how good it will be.

2.2.1.5 Image registration

New images are added to the current model by solving the Perspective-n-Point (PnP) problem [36] which uses correspondences of previously triangulated 3D points to points in the image that we want to add, resulting in an estimation of the camera pose of this image.

2.2.1.6 Triangulation

Newly registered image may add new points to the resulting reconstruction using triangulation. The only condition is that the newly added points must be seen from at least one already registered image from a different viewpoint.

2.2.1.7 Bundle adjustment

Image registration and triangulation give us rough estimates of 3D points and camera poses. To further improve these estimates, we use an optimization method called bundle adjustment (BA) [37] that tries to minimize reprojection errors and eventually discards these points or poses.

2.2.2 Multi-view stereo

Multi-view stereo algorithms aim to reconstruct dense models from images and their corresponding camera parameters. Here, we can see why multi-view stereo (MVS) comes after SfM. It uses estimated camera poses and tries to reconstruct more detailed, dense models. More detailed information on MVS and its types can be found here [38].

2.2.2.1 Depth maps estimation

The depth of each pixel is calculated in all images. There exist many approaches, such as Semi-Global Matching [39, 40] or ADCensus [41].

2.2.2.2 Meshing

In this step, all depth maps are merged into a structure representing the mesh. Further procedures and filterings (described in [38, pp. 73–98]) are performed to retrieve the final mesh.

2.2.3 Existing software solutions

Writing our implementation of the photogrammetry pipeline would be ineffective and probably would not even compete with existing ones. Therefore, in this subsection, we try to discover and compare multiple existing solutions from which we choose one. The solution must support automation, either a command-line interface (CLI) or scripting, because we will run multiple reconstructions. The solution should also support fiducial markers because we need to scale and orient the models correctly, so the resulting visualization is consistent.

2.2.3.1 3DF Zephyr

3DF Zephyr [42] is a proprietary software solution that comes in three different variants (Zephyr, Zephyr Lite, Zephyr Free), of which only one is free. Each variant has its limitations, and our requirements would require the Zephyr variant, which is the most expensive one. It supports only the Windows operating system (OS) and recommends an NVIDIA CUDA-enabled GPU for better performance.

2.2.3.2 Agisoft Metashape

Metashape [43] is another proprietary software that comes in two variants (standard, professional), both paid. It officially supports Windows, Linux (Debian, Ubuntu), and macOS. We cannot use the standard edition because it lacks support for CLI and fiducial markers. With Metashape, we can also use AMD GPUs since it is not restricted to NVIDIA.

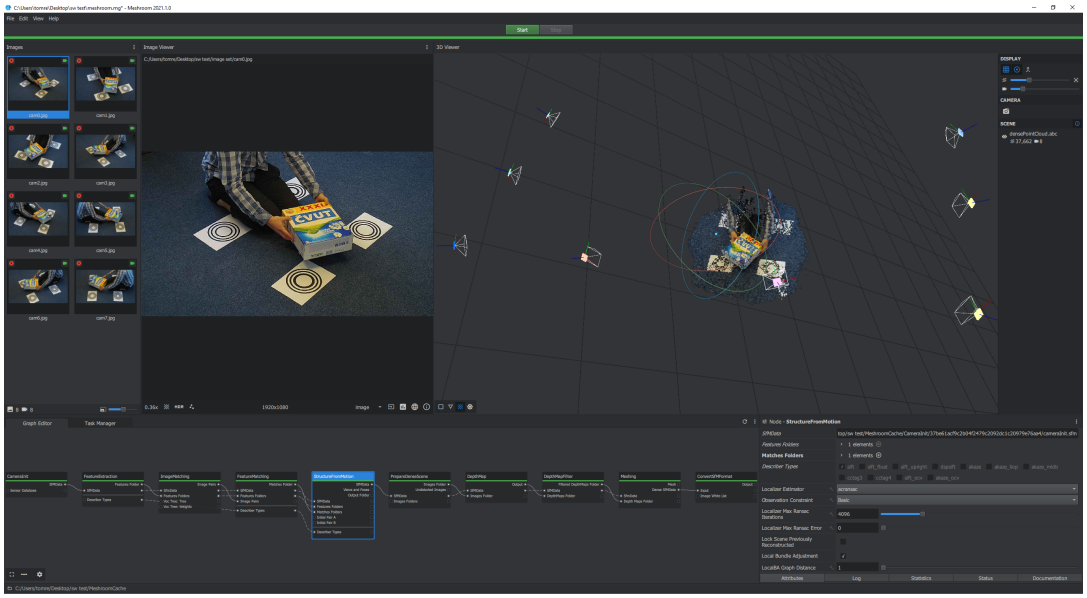
2.2.3.3 Meshroom

Meshroom [44] is a free open-source software solution, written in Python, available from both the graphical user interface (GUI) and CLI. It is written around the AliceVision framework. It uses a nodal system, see Figure 2.10a, that provides many options for customization, including writing custom nodes. The first release came out in 2018, so it is relatively new software, but developers are still improving it, including the community ones. For now, Meshroom only supports the fiducial marker CCTag. It can be run on both Windows and Linux OS, leaving macOS as the only non-supported among the major OS. Unfortunately, to fully utilize its potential NVIDIA CUDA-enabled GPU is necessary. Without it, some nodes do not work.

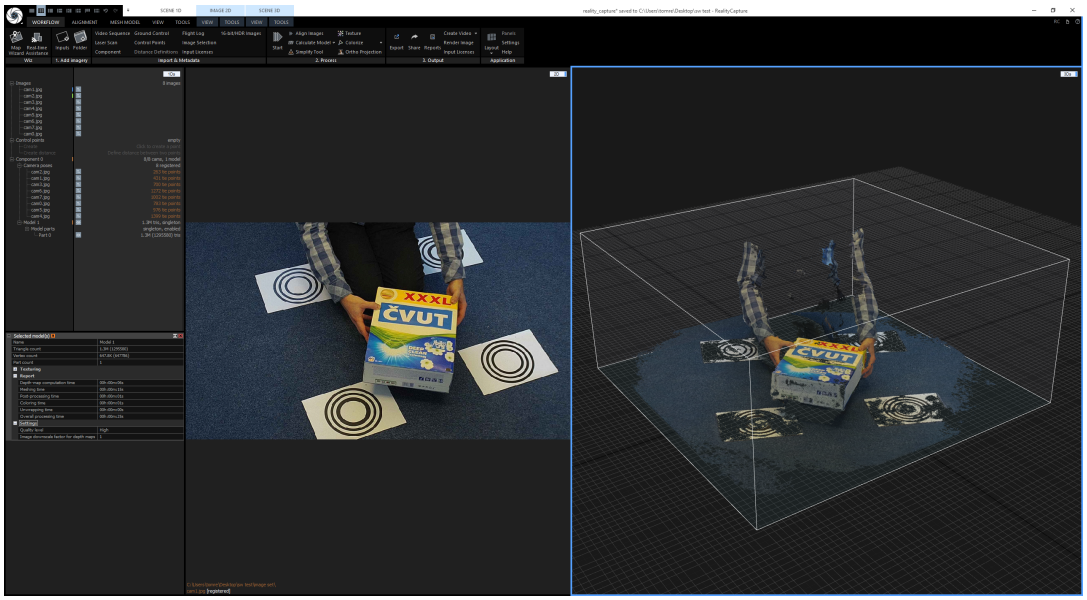
2.2.3.4 RealityCapture

RealityCapture [45] basic version uses a pay-per-input licensing that allows the user to reconstruct the model for free and pay only for model export. The fee depends on the quality of the reconstructed model. Unlike Meshroom it uses more classical controls placed in the top bar of the window, see Figure 2.10b. It requires a Windows OS and an NVIDIA CUDA-enabled GPU. It supports both circular and square fiducial markers (e.g., AprilTag).

Its advantage over the other software solutions is the fast processing speed. This would suit our application, but the automation support comes only with the Enterprise version.



(a) Meshroom



(b) RealityCapture

Figure 2.10 Photogrammetry software solutions GUI

2.3 Visualization

We want to combine all the previous reconstructions into one dynamic visualization in this last part. Since photogrammetry software usually exports point clouds in various formats, we want to find a universal tool that allows us to visualize them.

As the name suggests, the point cloud is a set of points, where each point is represented with coordinates and usually color information. They are often used in the creation of meshed models from 3D reconstruction [46].

2.3.1 Existing software solutions and libraries

Writing our visualizer from scratch would probably be better because our task is atypical. However, for now, when we only experiment with the whole idea, we can take advantage of already existing solutions and modify them according to our needs.

2.3.1.1 Blender

Blender [47] is free and open-source 3D graphics software that supports many formats, including point clouds (with improved rendering from version 3.1). We can write our own add-on that could control the point cloud visualization.

2.3.1.2 Open3D

Open3D [48] is an open-source library for 3D data processing. It can be used with Python and C++ programming languages. It has support for point clouds and visualization, which we can slightly modify to our needs.

2.3.1.3 MeshLab

MeshLab [49] is another free open-source option. It focuses mainly on mesh processing, but also provides support for point clouds and visualization. Its visualization is very minimalistic and does not suit our needs.

2.4 Our solution

We will use eight IP surveillance cameras in our solution because we have easy access to them. As we stated in our goals, we want to introduce new possibilities to most existing multi-camera systems. Thus, a few non-ideal cameras should cover many of them since most systems are better equipped.

These cameras are known for their high field of view, which causes distortion in the captured images. To overcome this, we will use the OpenCV library [50] to estimate the camera parameters and distortion coefficients for each camera. With these parameters and coefficients, we will be able to undistort the captured images.

For 3D reconstructions, we will use Meshroom. We find this open-source software solution ambitious, and thanks to its support for CLI and great customizability, we choose it for our implementation over the others.

Lastly, for our animated point cloud visualizer we will use the Open3D library and for illustrative, more specific visualizations we will use Blender.

Chapter 3
Implementation

In this chapter, we describe how we installed our multi-camera benchmark system and how we implemented the pipeline to create an animated point cloud and its visualization. Programming was done in Python [51], with packages handled by Anaconda [52]. The Anaconda environment initialization file with all necessary packages (primarily for Windows OS) can be found in `other/env.yml`.

3.1 Multi-camera benchmark system

Creating our own benchmark system with synchronized cameras was necessary because it showed us some minimal requirements that systems should have to run our proposed pipeline and photogrammetry in general. The hardware used in our system can be seen below:

- 8 cameras: HIKVISION DS-2CD2723G1-IZ, IP surveillance camera [53]
  - resolution: 2 MP, 1920 x 1080
  - image sensor: 1/2.8" CMOS
  - lens: varifocal 2.8 to 12 mm
  - aperture: F1.6
  - compression: MJPEG, H.264, H.265
- switch: Aruba 2530 48 PoE+ Switch [54]
- 3 m LED strip
  - input voltage: DC 24 V
  - power: 9,6 W/m
  - color temperature: 3000 K
- control station: notebook with external network card

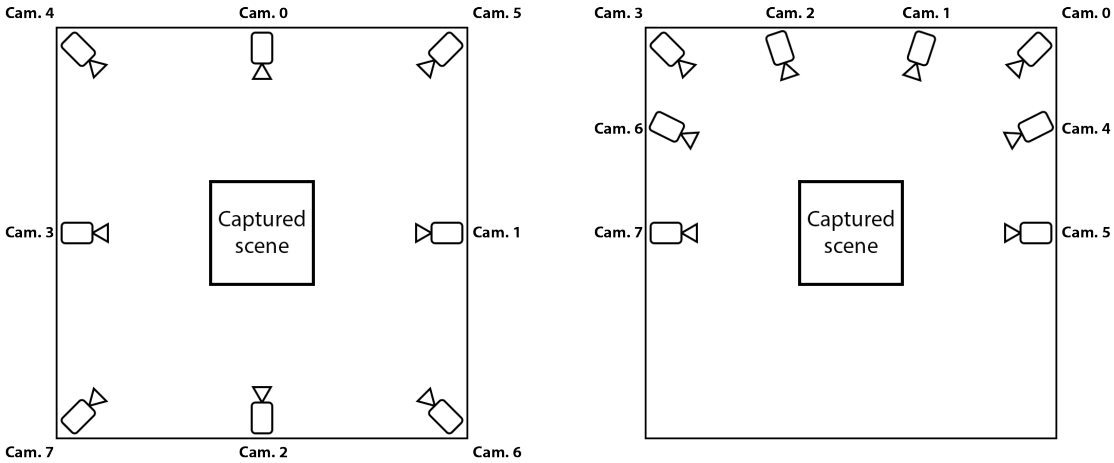
3.1.1 Simulation

Before installing the whole system, we wanted to simulate it in Unreal Engine [55] to see how it can perform under ideal conditions. We created a simple scene with eight cameras with parameters very similar to the cameras we use. We then tried to run reconstructions in Meshroom.

However, none of the reconstructions completed the pipeline. We also tried to run reconstructions in other software solutions to eliminate software problems. On further examination, we found that feature extraction is not working correctly because the render engine created artifacts in the images, thus the whole SfM pipeline did not work. We tried different rendering settings but failed each time. We believe that these artifacts came from post-processing.

After an unsuccessful simulation in Unreal Engine, we switched to Blender [56]. We rendered the scene with disabled shaders and used the Eevee rendering engine without any post-processing, especially dither. The resulting rendered images were used again in Meshroom and successfully completed the entire pipeline with the correct estimation of camera poses.

We tried two different camera layouts, both of which worked. The first layout, we call it the 360 layout, has cameras around the object in approximately 45 ° angle difference between cameras, see Figure 3.1a. The other layout, we call it the 180 layout, has cameras only around half of the object with approximately 25 ° angle difference between cameras, see Figure 3.1b.



(a) 360 camera layout

(b) 180 camera layout

Figure 3.1 Used camera layouts

3.1.2 Installation

We installed our multi-camera system in a room of size approximately 4 x 4 m. The cameras were attached to metal profiles so that they could be easily moved around. The floor of the room was covered with carpet. For some experiments we laid four matte plastic sheets over the carpet.

We connected the cameras to a switch outside the room with Cat.6e cables. Thanks to the power over ethernet (PoE) support on our switch, we only had to run one cable to each camera for both power supply and data transfer.

We first placed the cameras according to the 360 layout, see Figure 3.2a. This layout has more coverage at the cost of smaller overlaps between camera views. After some experimenting, we changed the layout to 180 layout, see Figure 3.2b. This layout has less coverage but more overlapping camera views. The impact of these layouts on the reconstructions can be seen in the experiment 4.2.

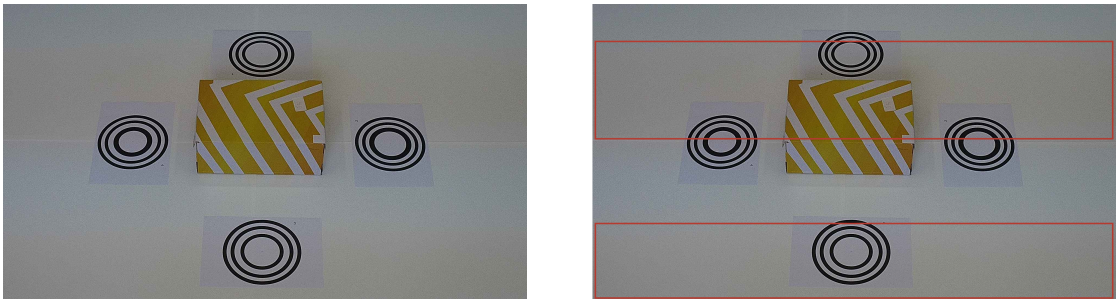
Before installing the LED strip, we tried to use the light fixture in the room. As expected, the cameras flickered, and the captured images were degraded. Changing the camera configuration to PAL 50 Hz helped a little bit but did not solve the problem entirely, see Figure 3.3. So, we cut

the LED strip into four pieces. These pieces were attached to the construction, made of plastic wiring conduits, and hung from the light fixture in the room.

(a) 360 layout

(b) 180 layout

Figure 3.2 Our multi-camera system



(a) Degraded image

(b) Highlighted flickering

Figure 3.3 Camera flickering

3.1.3 Camera configuration

After connecting the cameras, we configured their IP addresses and credentials. This was done in software provided by HIKVISON called iVMS-4200 [57]. After this initial configuration, we had access to the cameras through a web browser interface. At first, we updated the firmware to the latest version (5.6.6). Then we moved the cameras so that they were looking at the center of the room. After that, we set the camera zoom to maximum and started the automatic focus procedure on each camera.

The cameras deliver three independent video streams. We used the first (main) stream, which we set to H.264 lossy compression (with the highest possible image quality and bitrate) and resolution to 1920 x 1080 px at 25 fps.

The compression ratio is large in IP surveillance cameras because their main focus is to store as much footage with the smallest possible file size. Due to this reason, one image captured from our camera takes on average 400 KB. Raw 1920 x 1080, 8-bit image has a size of 6220.8 KB. We can easily calculate the compression ratio using the formula 3.1, which is equal to 15.5. This high compression ratio results in many spatial artifacts that significantly degrade our images. In Figure 3.4, we can see the blocking artifact.

$$Compression\ ratio = \frac{Uncompressed\ size}{Compressed\ size} \tag{3.1}$$

We changed the white balance and the shutter speed to manual so that all cameras have a similar image. We also disabled all filters (e.g., smoothing), this resulted in noisy but more accurate images. There are many less important parameters that we changed, but we cannot list all of them. For this reason, we also exported the configuration files from each camera and stored them on the provided media in `other/camera_configs`.

3.1.4 Acquisition control

We implemented a simple class that communicates with the camera web server. Communication is done through HTTP requests, for which we used the Requests library [58] for Python. The uniform resource locators (URLs) on which these requests are made are defined by ISAPI [59] by HIKVISION. A typical workflow is to make a GET request for which the camera returns the XML file. Modify the XML file and make a PUT request to overwrite the old file.

We implemented two different solutions for synchronized image acquisition and one solution for capturing images on demand (e.g., for capturing calibration images). The comparison of the first two solutions can be seen in the experiment 4.1.

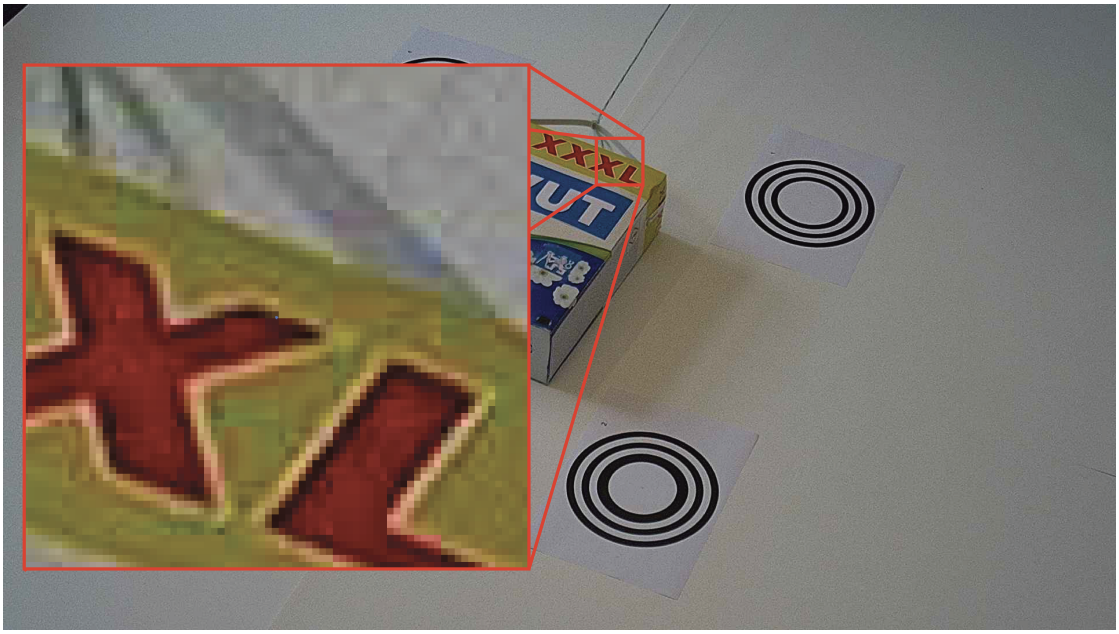


Figure 3.4 Spatial compression artifacts

3.1.4.1 Independent camera acquisition

The first solution uses the file transfer protocol (FTP) and NTP server. The cameras are configured to communicate with both servers. In our implementation, we used the pyftpdlib library [60] to run a simple FTP server on the control station. For the NTP server, we used a nearby public server, for which we had to connect the whole system to the Internet. It should be noted that the Internet connection may raise some security concerns. The NTP server kept all the cameras' time synchronized.

To start the recording, we enable, through our implemented camera class, the built-in function on all cameras that saves the current image on the FTP server in configurable time intervals. Enabling this function is a critical point of this solution because if we do not activate it at ideally the same time, the cameras will never synchronize. To overcome this critical point, we use multiple threads, each communicating with one camera. In this way, we prevented resource blocking while waiting for the cameras to respond. To stop the recording, the built-in cameras' function is disabled.

To summarize this solution, we enable the cameras' built-in function for recording from the control station, and then the cameras independently store captured images on the FTP server.

3.1.4.2 Acquisition from control station

The second solution uses the master/slave model (similar to genlock 2.1.1.4), where the control station acts as a master, asking all cameras to send the current image at the same time. We ask for images through our implemented camera class. Again, to prevent resource blocking, we use multiple threads with a shared clock, each communicating with one camera. This represents a limitation because between each acquisition, each thread must wait for a response with the image payload and store it. On the other hand, we do not need Internet access.

3.1.4.3 Image on demand

The third solution is much simpler. It uses the real time streaming protocol (RTSP) to display the live feed of one camera, and the image is stored only when a user demands it (by clicking on the live feed window). This sends a request to the previewed camera, which responds with the image that is stored. The RSTP stream is only for preview, we do not save images directly from it. This was implemented mainly for calibration purposes.

3.1.5 Calibration

To calibrate the cameras, we used the OpenCV library [50] with support for camera calibration. It is important to note that we used camera calibration only for distortion correction. We did not provide any camera parameters to the photogrammetry software.

We used a chessboard pattern printed on A3 paper, which was adhered to a rigid board. We then captured around 20 images (OpenCV suggests at least 10) for each camera with different rotations and positions of the pattern in the camera view. An example of a calibration image can be seen in Figure 3.5.

For both camera layouts, we performed calibration on all cameras individually. In total, we had 173 calibration images for 180 layout and 184 images for 360 layout. We had to calibrate the cameras for each layout separately because we changed the focus on all cameras after the rearrangement. We then saved the results in files so that we could later use them.

Since we zoomed in all the cameras, it significantly reduced the distortion in the produced images, but not completely. We can see the effect of the image undistortion of the 1 cm grid pattern in Figure 3.6.

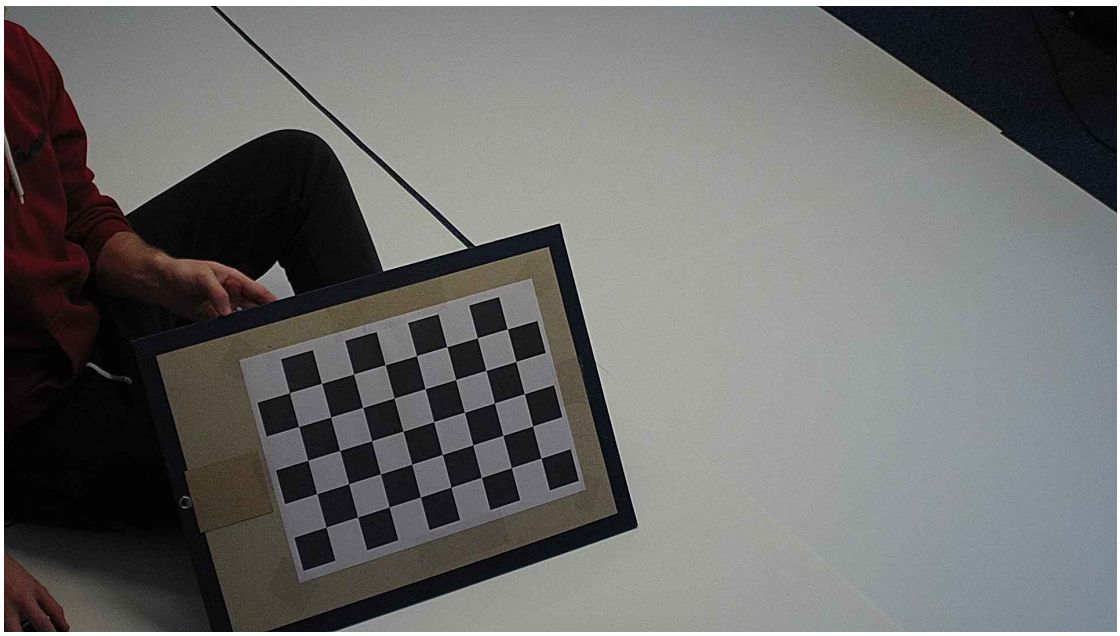
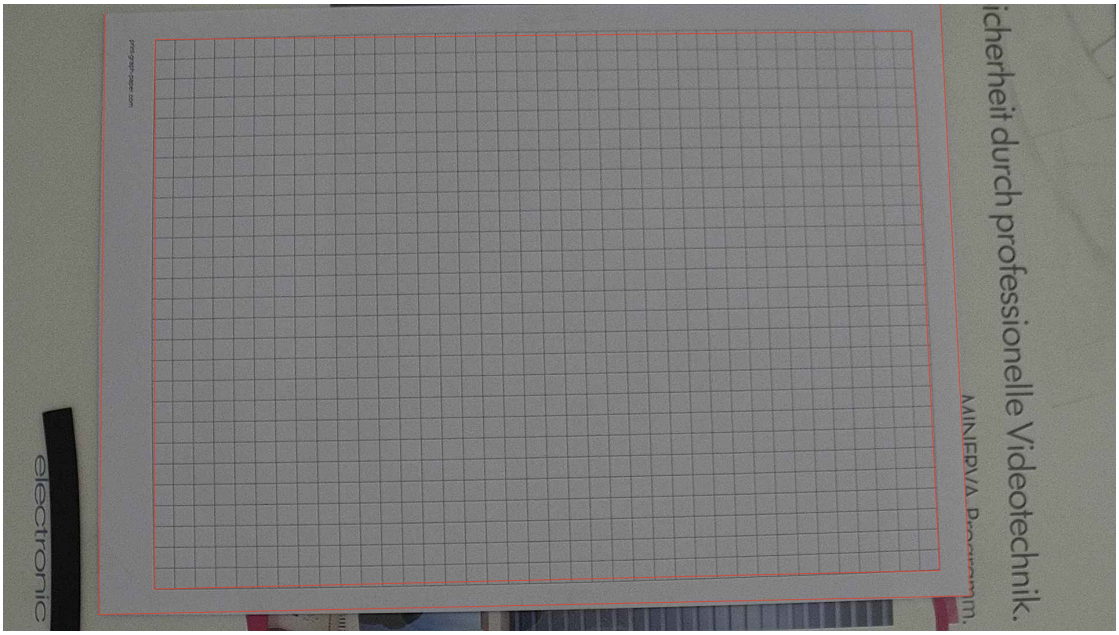
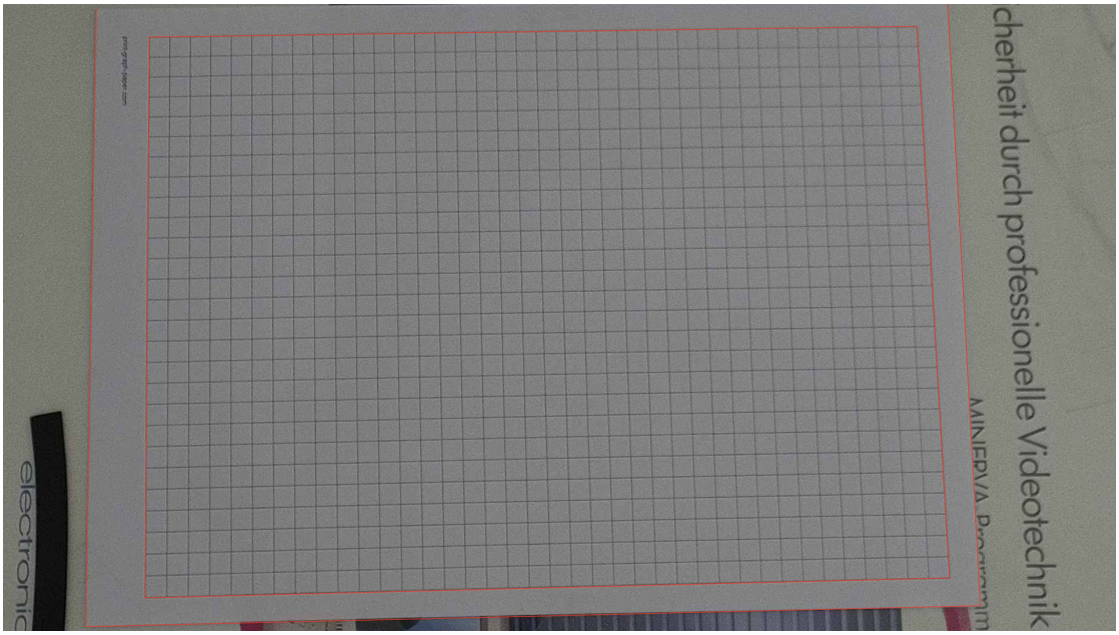


Figure 3.5 Example of calibration image



(a) Before undistortion



(b) After undistortion

Figure 3.6 Image undistortion with highlighted edges

3.2 Animated point cloud pipeline

The whole pipeline can be separated into three individual parts. The first part of the pipeline is the preparation of image sets for the reconstruction pipeline, namely image acquisition, undistortion, and image ordering. The second part is the reconstruction itself. And the third part is the visualization of the results produced by reconstructions. In Figure 3.7, we can see the whole sequence. The first three steps are specific to the multi-camera system used.

The examples can be found in 4.5 and the described usage in the jupyter notebook [61] in `src/pipeline.ipynb`.

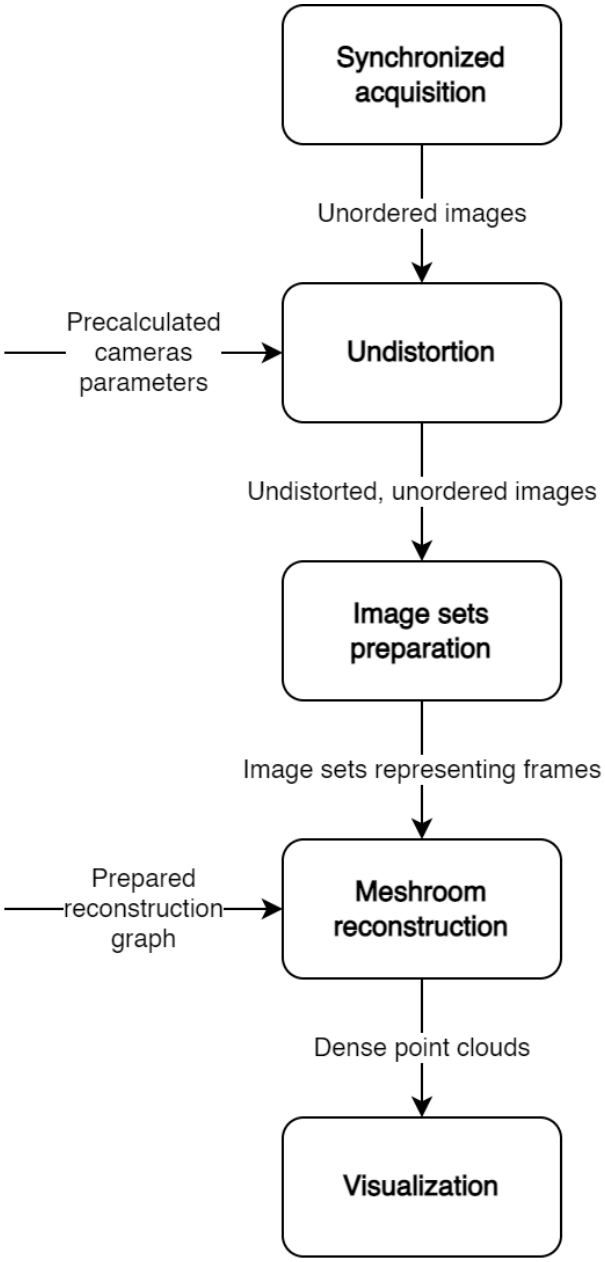


Figure 3.7 Animated point cloud pipeline

3.2.1 Synchronized acquisition

In this step, the scene is captured from all cameras in short time intervals. In our implementation, we ended up using the acquisition from control station described in 3.1.4.3. We set the duration of the recording and the interval between frames. At the end of the recording, each camera has its own directory containing images named after the timestamp at which they were captured, see Figure 3.8a.

The implementation can be found in `src/modules/multicamera/hikvision_camera.py`.

3.2.2 Undistortion

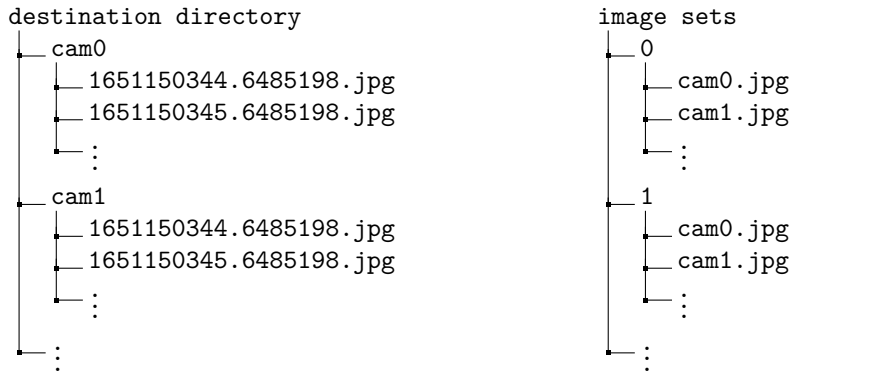
For each camera directory, the camera parameters and distortion coefficients are loaded from the corresponding calibration file. Using these parameters and coefficients, each image is undistorted and copied to a new directory.

The implementation can be found in `src/modules/multicamera/calibration.py`.

3.2.3 Image sets preparation

So far, the images were grouped by cameras, in this step they are grouped by frames. The whole process starts with collecting all possible timestamps (image names). For each timestamp, the corresponding images are copied from all camera directories. If any image is missing, the warning message is printed. This results in new directories named after frames, each of which contains the corresponding images, see Figure 3.8b.

The implementation can be found in `src/modules/multicamera/image_sets.py`.



(a) After image acquisition

(b) After image sets preparation

Figure 3.8 Image directory structures

3.2.4 Meshroom reconstruction

Meshroom provides an executable file called `meshroom_batch` that can be called from the CLI. This file starts the default reconstruction pipeline unless a graph file (with a different photogrammetry pipeline) is provided. This provides great flexibility in reconstructions because this graph file can be changed to suit the captured scene. We can create a new graph file using the Meshroom GUI by setting the nodes and saving the project, see Figure 3.9. The saved project file ending with `.mg` represents the graph.

Our implementation calls `meshroom_batch` with a predefined graph for each set of images. The predefined graph can be freely changed, but it has to contain one publish node, which yields

a point cloud, and one SfMTransform node, which correctly scales and rotates the reconstructions (ideally with CCTags used).

In the current version of Meshroom (2021.1.0) there are some limitations and bugs when using CCTags in the FeatureExtraction node. The images we provide must have a maximum resolution of 6144 px. We also have to set the Max Nb Threads to 1, otherwise the node fails. This slows down the extraction of other features and significantly extends the processing time. For a better understanding of the Meshroom nodes, refer to the manual [62], which can also be found in `other/meshroom_manual.pdf`.

The implementation can be found in `src/modules/meshroom_bash.py`.

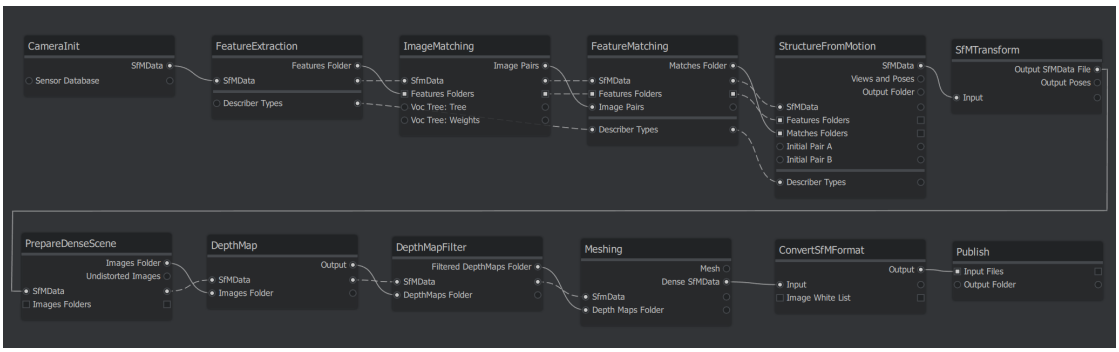


Figure 3.9 Meshroom graph

3.2.5 Visualization

Visualization is the last step of the pipeline, which loads the reconstructed point clouds and shows them one by one in the correct order.

We used a VisualizerWithKeyCallback class from the Open3D library [48]. We configured the visualizer to show different point clouds without resetting the view. We also added basic control over the visualization, such as pause, speed control, and manual skipping.

The implementation can be found in `src/modules/visualizer.py`.

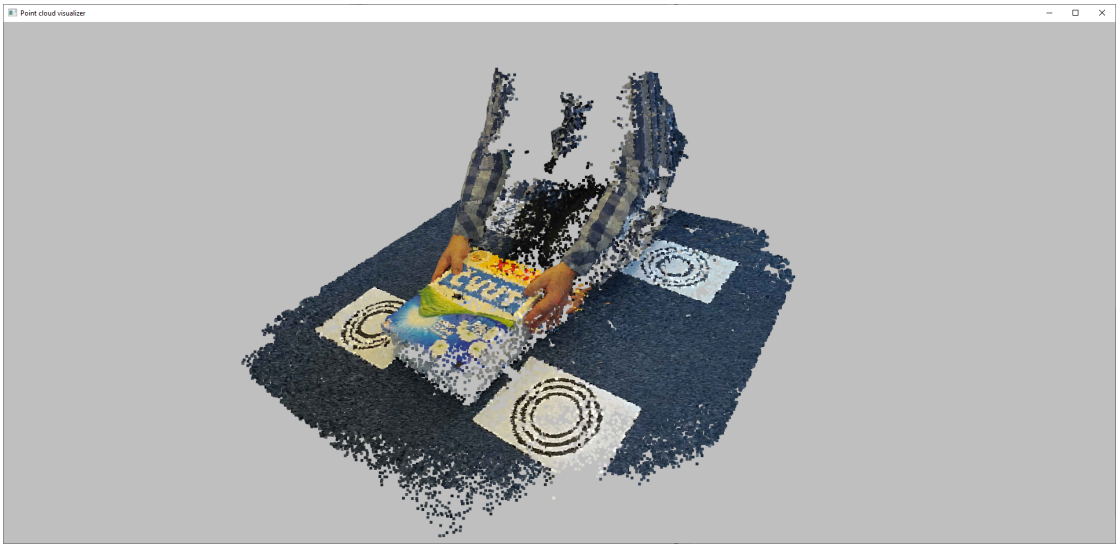


Figure 3.10 Animated point cloud visualizer

Chapter 4
Experiments

In this chapter, we present the experiments we have performed and their results. We have experimented with our multi-camera system, photogrammetry software solutions, and animated point cloud pipeline. Please note that Meshroom uses absolute file paths, therefore, the provided Meshroom project files will need to be adjusted. The Blender project of the simulated scene can be found in `experiments/blender_simulated_scene`.

All reconstructions in the experiments were performed on a computer with the specifications listed below:

- CPU: Intel Core i7-4770
- GPU: NVIDIA GeForce RTX 3060 Ti
- RAM: 16 GB DDR3

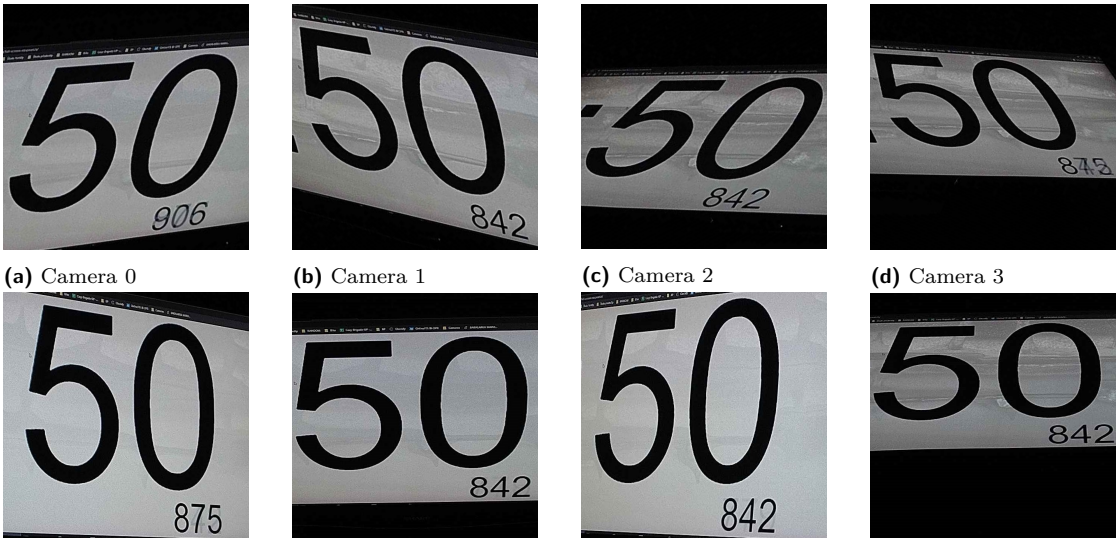
4.1 Camera synchronization solutions comparison

We compared our implemented synchronize acquisition solutions with a simple stopwatch experiment. We placed a screen displaying stopwatches in the middle of the captured scene and ran the recording, with the interval between images set to one second and the duration of recording to 60 seconds.

From the captured images (see Figure 4.1), we could extract the time on the stopwatches and compare the time between each acquisition (frames) and the maximum differences between each camera. Some of the collected times from the acquisition from the control station solution can be seen in Table 4.1 and from the independent camera acquisition solution in Table 4.2.

From the recorded times, we calculated the average time between frames and the average maximum time difference between the cameras. The average times between frames were 0.9999 s for control station and 1.0002 s for independent camera acquisition. We can see that both results are very close to configured 1 s. However, the independent camera acquisition solution failed to save one image (Frame 19) from Camera 1. The average maximum times difference between frames were 83 ms for the control station and 722 ms for independent camera acquisition. In conclusion, the acquisition from the control station solution performs overall better.

All images and recorded times can be found in `experiments/synchronization`.



(a) Camera 0 **(b)** Camera 1 **(c)** Camera 2 **(d)** Camera 3

(e) Camera 4 **(f)** Camera 5 **(g)** Camera 6 **(h)** Camera 7

Figure 4.1 Captured stopwatches from synchronization experiment (rotated, cropped)

Table 4.1 Collected times from control station synchronization

	Cam. 0	Cam. 1	Cam. 2	Cam. 3	Cam. 4	Cam. 5	Cam. 6	Cam. 7
Frame 0	0.906	0.842	0.842	0.842	0.875	0.842	0.842	0.842
Frame 1	1.930	1.834	1.834	1.834	1.834	1.834	1.802	1.834
Frame 2	2.891	2.827	2.827	2.858	2.858	2.827	2.827	2.858
Frame 3	3.914	3.818	3.818	3.882	3.882	3.818	3.818	3.818
Frame 4	4.938	4.842	4.842	4.875	4.875	4.842	4.842	4.842
Frame 9	9.898	9.866	9.835	9.866	9.866	9.835	9.835	9.866
Frame 19	19.882	19.850	19.818	19.882	19.882	19.818	19.818	19.850
Frame 29	29.898	29.866	29.866	29.866	29.898	29.866	29.866	29.866
Frame 39	39.914	39.818	39.818	39.850	39.914	39.818	39.818	39.818
Frame 49	49.898	49.867	49.803	49.867	49.898	49.803	49.803	49.867
Frame 59	59.882	59.851	59.851	59.851	59.882	59.818	59.818	59.851

Table 4.2 Collected times from camera independent synchronization

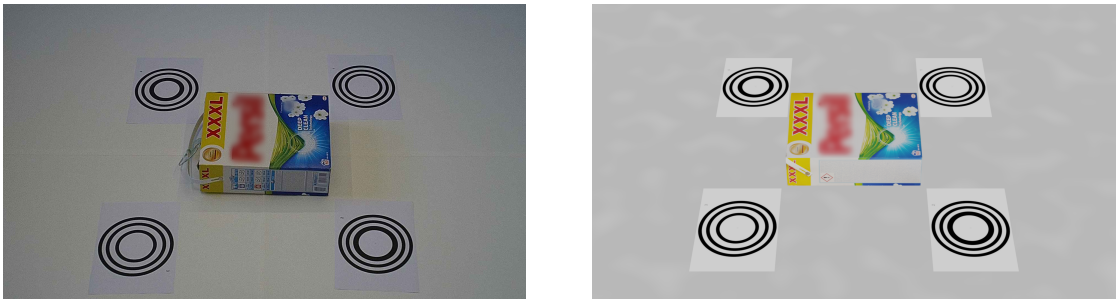
	Cam. 0	Cam. 1	Cam. 2	Cam. 3	Cam. 4	Cam. 5	Cam. 6	Cam. 7
Frame 0	0.062	0.580	0.489	0.705	0.419	0.062	0.384	0.802
Frame 1	1.060	1.579	1.477	1.697	1.441	1.600	1.252	1.796
Frame 2	2.047	2.564	2.466	2.722	2.436	2.047	2.369	2.784
Frame 3	3.071	3.584	3.468	3.584	3.428	3.071	3.392	3.585
Frame 4	4.066	4.544	4.450	4.640	4.450	4.066	4.384	4.866
Frame 9	9.088	9.567	9.349	9.704	9.420	9.088	9.343	9.792
Frame 19	19.073	NONE	19.424	19.717	19.424	19.073	19.395	19.842
Frame 29	29.898	29.866	29.866	29.866	29.898	29.866	29.866	29.866
Frame 39	39.080	39.592	39.428	39.714	39.428	39.045	39.397	39.813
Frame 49	49.090	49.559	49.474	49.735	49.408	49.055	49.376	49.823
Frame 59	59.081	59.585	59.465	59.711	59.465	59.013	59.422	59.812

4.2 Realized camera layouts comparison

We reconstructed two very similar scenes, see Figure 4.2, one real (captured by our multi-camera system) and one simulated (renders). We arranged the cameras according to both 180 layout and 360 layout. For all reconstructions, we used the exact same computational graph. The measured criterion was the number of points in the resulting point cloud and in the cropped point cloud containing only the object of interest. The results can be seen in the graphs 4.3.

As we can see, the 180 layout performs better even in the simulated scene. This shows us that to perform dense reconstruction, we must have large overlapping areas between images, and the 360 layout does not provide them with only eight cameras. However, with the 180 layout, we do not cover one side of the captured scene, which results in a hollow side in the point cloud.

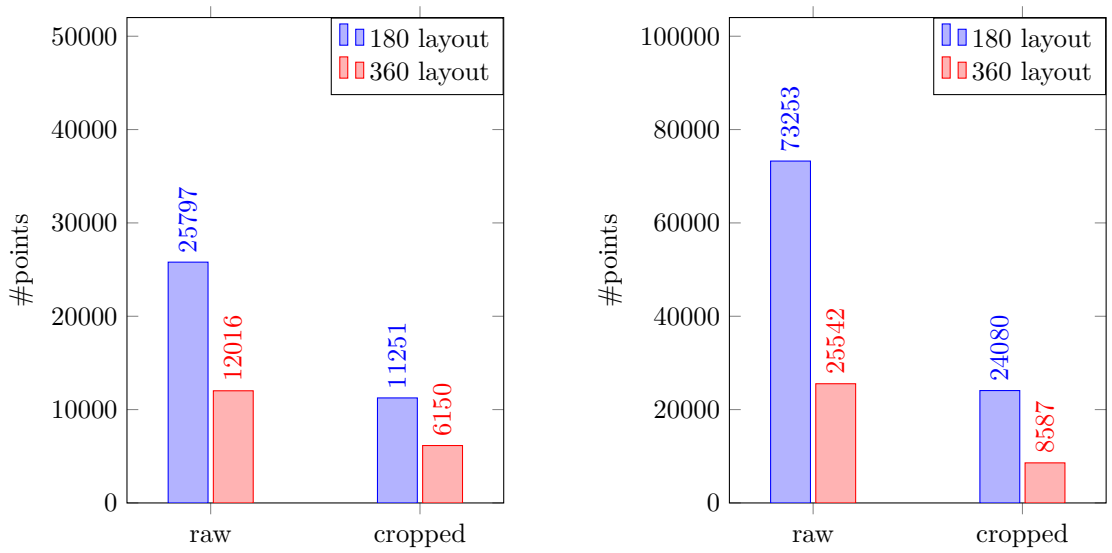
The resulting point clouds, source images, and Meshroom project files can be found in `experiments/layout`.



(a) Captured image

(b) Rendered image

Figure 4.2 Layout experiment scene



(a) Captured images

(b) Rendered images

Figure 4.3 Comparison between realized camera layouts

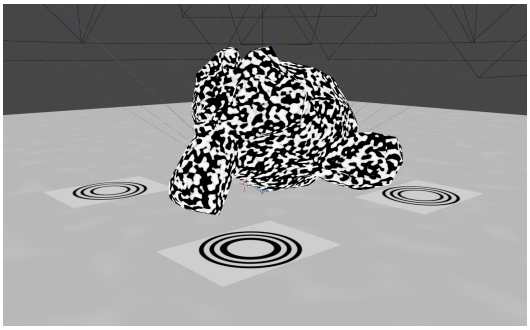
4.3 Dependence of point cloud density on the number of cameras

Since we only have eight cameras, we performed this whole experiment in a simulated scene. We placed a more complex model with Musgrave texture in the middle of the scene and made renders from 4, 8, 16, 32 and 48 cameras that were around the model, see Figure 4.4. The cameras were arranged in one (4, 8, 16 cameras), two (32 cameras) or three (48 cameras) rows. We then reconstructed multiple point clouds from these renders using the exact same computational graph. As in the previous experiment, our measured criterion was the number of points in the resulting point cloud and in the cropped point cloud containing only the object of interest. The results can be seen in graph 4.5.

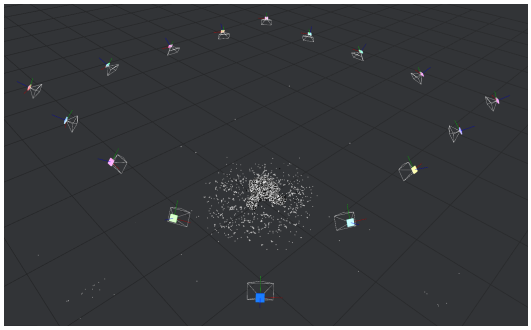
Reconstruction of 4 and 8 cameras failed because SfM could not estimate enough camera poses. The first successful reconstruction was with 16 cameras. For 16 and 32 cameras, SfM correctly estimated all cameras. For 48 cameras, only 46 were estimated. We can see that when we doubled the number of cameras, we got almost twice as many points. This was mainly caused by the fact that the newly added cameras captured lower parts of the object that were before unseen.

We assumed that this experiment would show us the optimal number of cameras, where adding more would not change the point cloud density. However, trying up to 48 cameras did not achieve that, and there was still room for improvement. On the other hand, we found out that having 16 cameras around the object is the recommended minimum, which corresponds to our multi-camera system, where we had 8 cameras around half of the object.

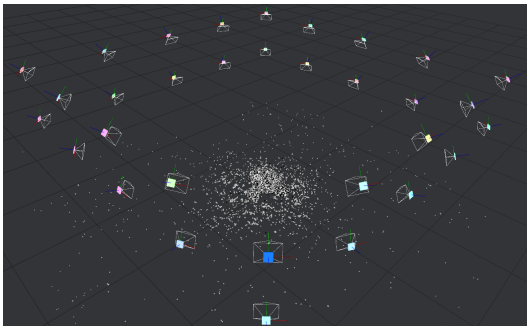
The rendered images, resulting point clouds, and Meshroom project files can be found in `experiments/camera_count`.



(a) Object of interest



(b) 16 cameras



(c) 32 cameras

(d) 48 cameras

Figure 4.4 Camera count experiment scene

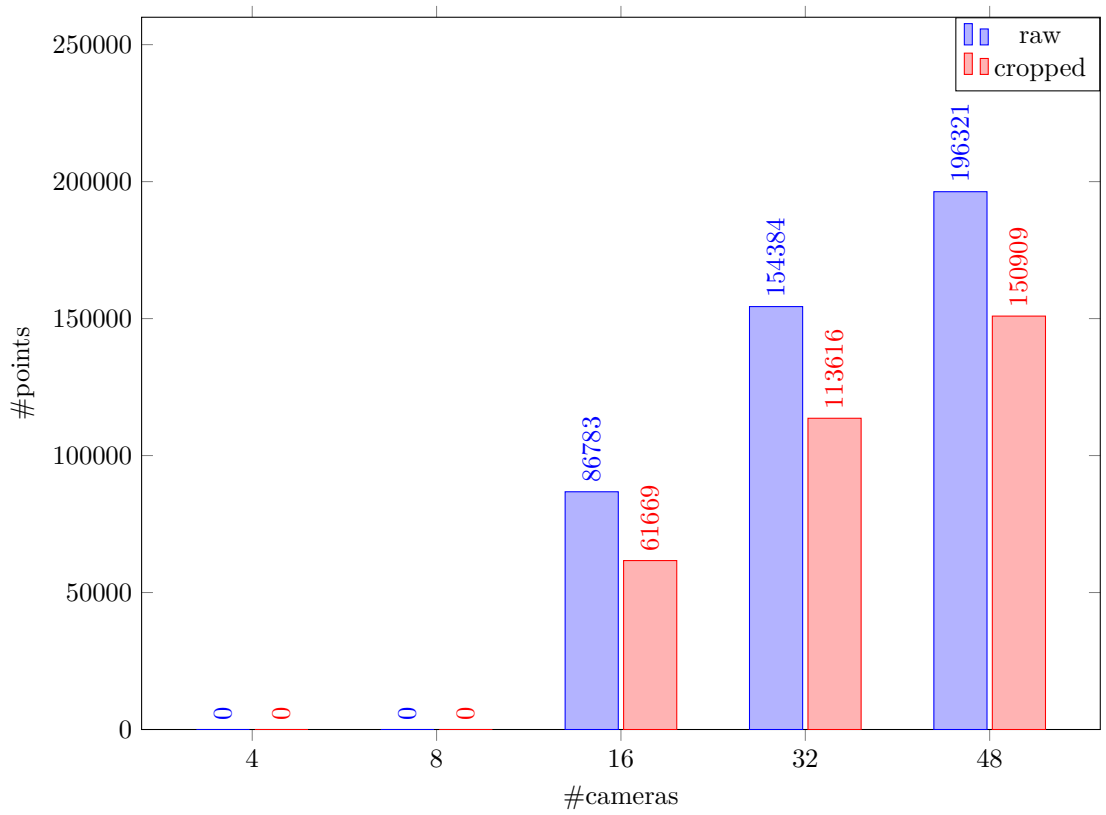


Figure 4.5 Dependence of point cloud density on the number of cameras

4.4 Photogrammetry software comparison

Even when we chose Meshroom as our photogrammetry software solution, our pipeline can be modified to utilize a different one. For this reason, we compared the mentioned software solutions in 2.2.3 so that we can see how open-source software compares with proprietary software. The measured criterion was again the number of points in the resulting point cloud and in the cropped point cloud and the computational time. We used one image set from the experiment 4.5.2 and performed reconstruction with the highest possible quality that these software solutions provide. Number of points can be seen in graph 4.6, computational times in table 4.3. and resulting cropped point clouds in Figure 4.7.

As we assumed, open-source Meshroom is still behind the proprietary competitors in both computational time and point cloud density. Agisoft Metashape excelled in point cloud density and is comparable to 3DF Zephyr in computational time. However, if we take a look at the resulting cropped point clouds, see Figure 4.7, we can see that RealityCapture and Agisoft Metashape results are very similar. RealityCapture achieved this result in less than half the time and fifth of the points compared to Agisoft Metashape. Therefore, RealityCapture would suit our application the best.

The source images, resulting point clouds, and project files can be found in `experiments/software_comparison`.

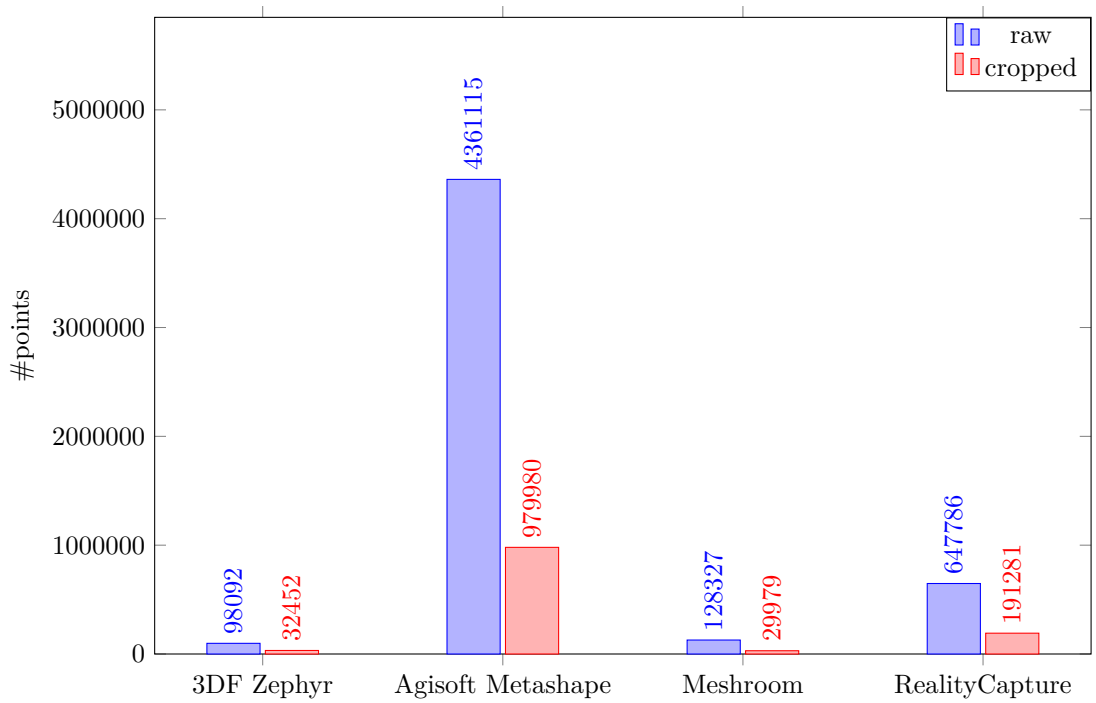


Figure 4.6 Photogrammetry software solutions comparison based on point cloud density

Table 4.3 Computational time of photogrammetry software solutions

	3DF Zephyr	Agisoft Metashape	Meshroom	RealityCapture
Features extr. + match.	2 s	41 s	7 s	1 s
Alignment	2 s	2 s	6 s	1 s
Depth maps	54 s	9 s	70 s	6 s
Dense point cloud	3 s	12 s	33 s	17 s
Total	61 s	64 s	116 s	25 s

(a) 3DF Zephyr

(b) Agisoft Metashape

(c) Meshroom

(d) RealityCapture

Figure 4.7 Resulting cropped point clouds of photogrammetry software solutions

4.5 Dynamic scene reconstruction

For this experiment, we took 30 s recordings from two different scenes using our multi-camera system with cameras arranged in 180 layout. The first scene contained a box that was slowly moved on plastic sheets using rope, see Figure 4.8. The second scene contained the lower half of a human sitting on a carpet interacting with the same box as in the first scene, see Figure 4.10. We then used these two recordings and continued following our proposed animated point cloud pipeline. This experiment showed us how capable our multi-camera system and proposed pipeline are.

4.5.1 Moving box

The Meshroom reconstruction part of our pipeline successfully reconstructed 20 of the 30 provided image sets. The other 10 had to be manually modified, mostly because SfM could not approximate all the cameras, which caused that not all CCTags were localized, and the SfMTransform node failed. One reconstruction took approximately 90 s. In Figure 4.9, we can see half of the point clouds captured from our visualizer.

Using the feature-less white plastic sheets on the ground, we achieved that the resulting point clouds mainly contained the object of interest. However, this caused fewer detected features and a lower success rate (66 %). This showed us that to perform better reconstructions, we need to reconstruct the static part of the scene as well, and that requires a different, feature-rich background.

The pipeline file with all the necessary source images can be found in `experiments/dynamic_reconstructions/moving_box`.

4.5.2 Human interaction

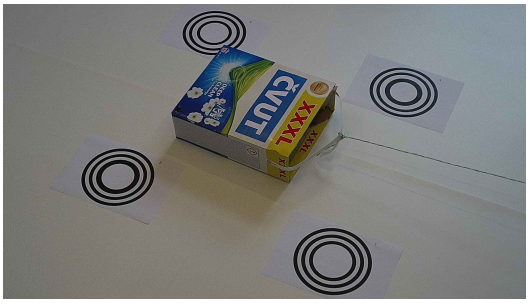
In this scene, we mainly focused on reconstructing a human that is interacting with some object. We chose specific clothing with a well-defined pattern, which helps in feature detection. Compared to the previous moving box scene, there is more motion, which is also faster. Therefore, the reconstructions were severely affected by our camera synchronization. However, unlike in the previous scene, all 30 of the 30 image sets were successfully reconstructed. One reconstruction took approximately 110 s. In Figure 4.9, we can see half of the point clouds captured from our visualizer. In Figure 4.12, we can see image overlaid with uncolored point cloud, which represents the depth information we gathered.

We achieved a better success rate with the carpet on the ground, even when reconstructing a more complicated scene. The reconstructions are still not perfect (e.g., the box texture is applied to the carpet as well), but we get more sense of what was happening in the scene compared to the previous moving box reconstructions 4.5.1.

The pipeline file with all the necessary source images can be found in `experiments/dynamic_reconstructions/human_interaction`.

(a) Camera 0

(b) Camera 1



(c) Camera 2

(d) Camera 3

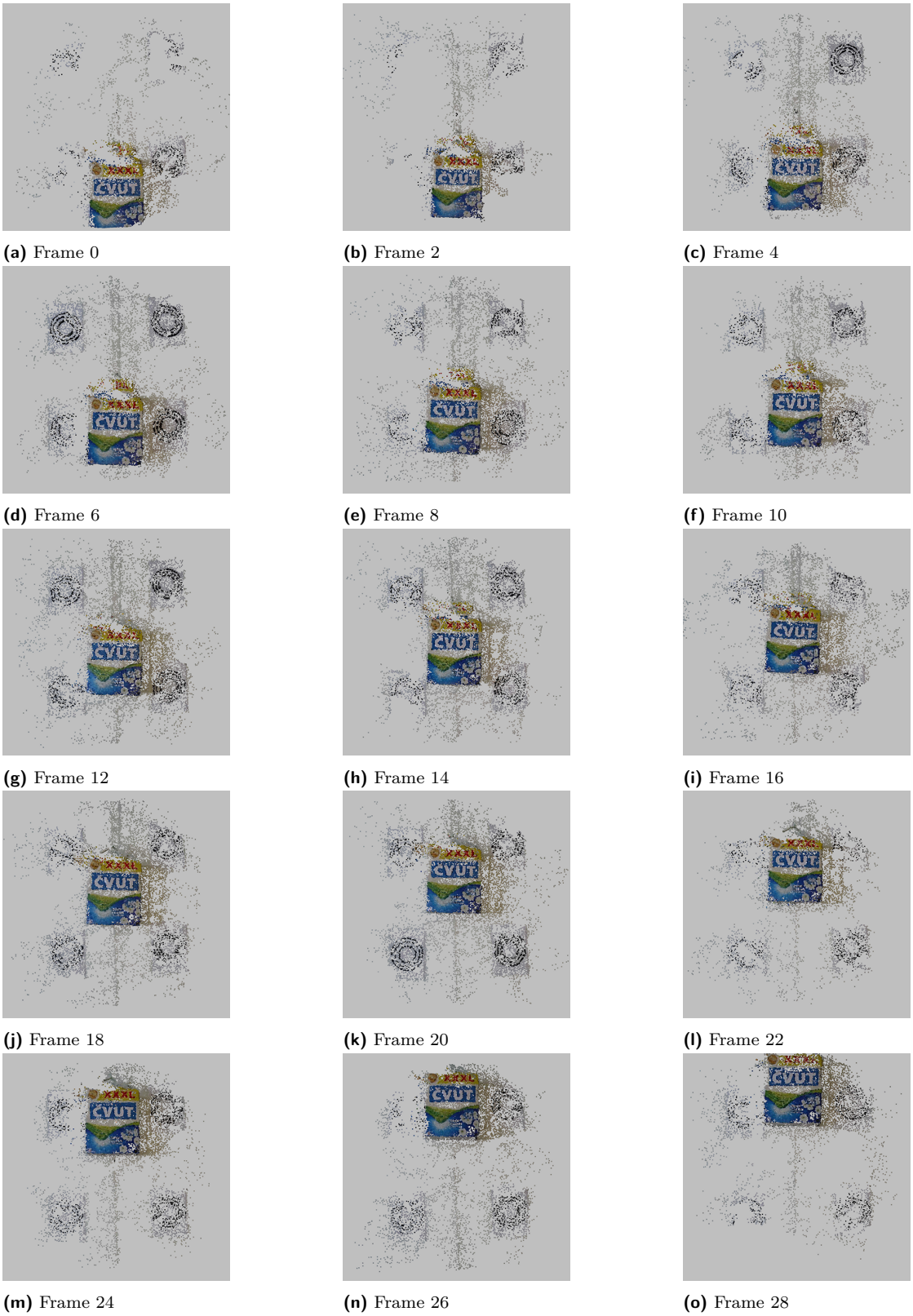
(e) Camera 4

(f) Camera 5

(g) Camera 6

(h) Camera 7

Figure 4.8 Example of an image set for the moving box experiment



(a) Frame 0 **(b)** Frame 2 **(c)** Frame 4

(d) Frame 6 **(e)** Frame 8 **(f)** Frame 10

(g) Frame 12 **(h)** Frame 14 **(i)** Frame 16

(j) Frame 18 **(k)** Frame 20 **(l)** Frame 22

(m) Frame 24 **(n)** Frame 26 **(o)** Frame 28

Figure 4.9 Animated point cloud of moving box

(a) Camera 0

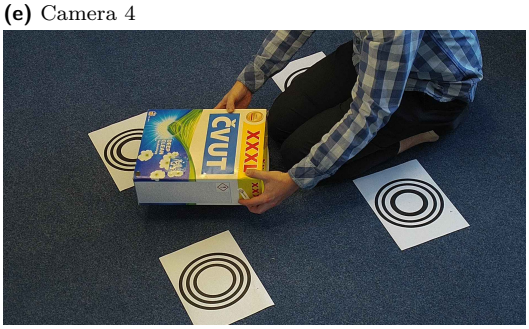
(b) Camera 1

(c) Camera 2

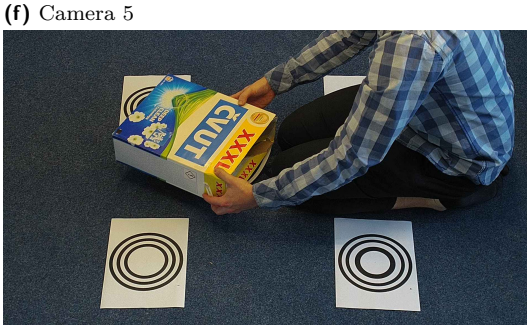
(d) Camera 3

(e) Camera 4

(f) Camera 5

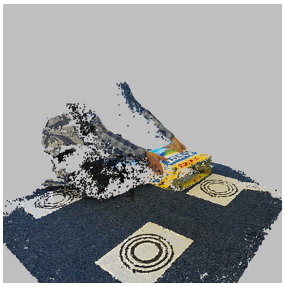


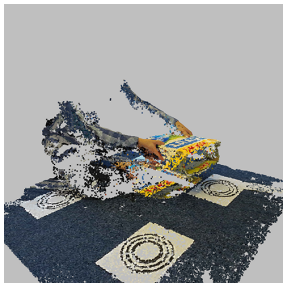
(g) Camera 6

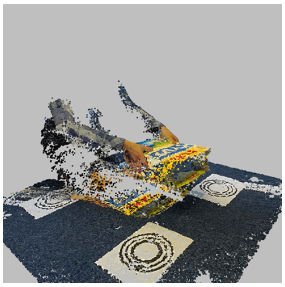


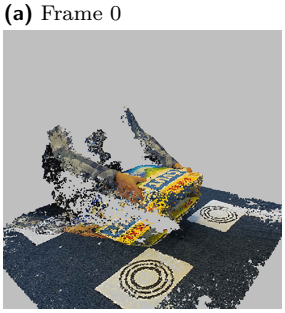
(h) Camera 7

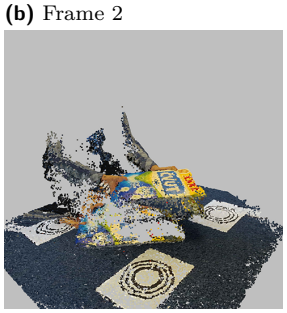
Figure 4.10 Example of an image set for the human interaction experiment

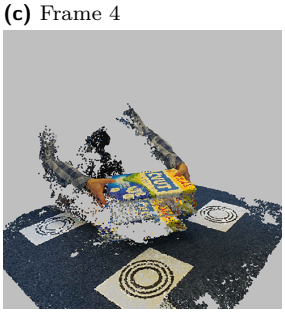

(a) Frame 0

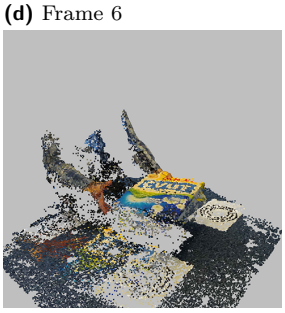

(b) Frame 2

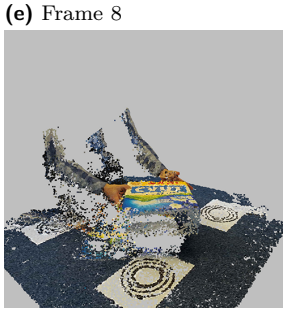

(c) Frame 4

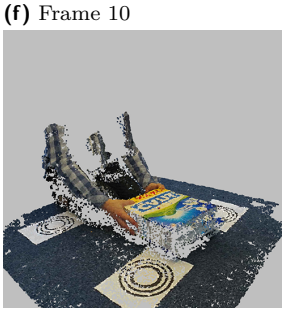

(d) Frame 6

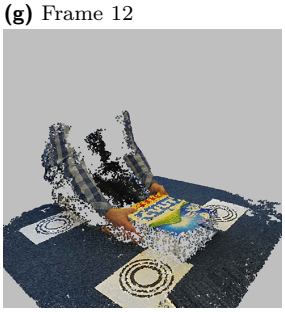

(e) Frame 8

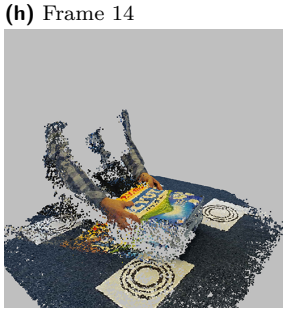

(f) Frame 10

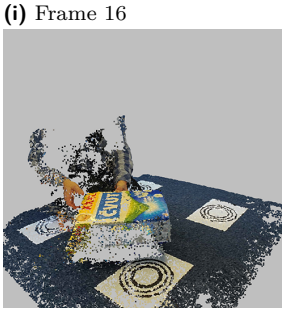

(g) Frame 12

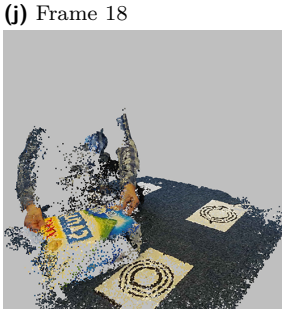

(h) Frame 14

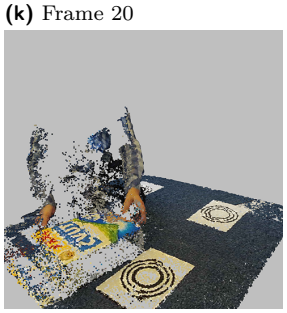

(i) Frame 16


(j) Frame 18


(k) Frame 20


(l) Frame 22


(m) Frame 24


(n) Frame 26

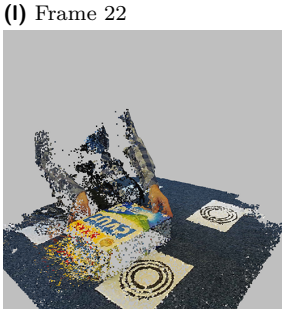

(o) Frame 28

Figure 4.11 Animated point cloud of human interaction with the box

Figure 4.12 Image overlaid with point cloud

Chapter 5
Conclusion

In this thesis, we have focused on creating our own multi-camera system and implementing a reconstruction pipeline that creates an animated point cloud from such systems utilizing an existing photogrammetry software solution.

For this reason, we first studied the multi-camera systems, their individual components, and already existing systems. Then we studied the photogrammetry pipeline and the software solutions that perform it. Finally, we studied existing solutions for visualizing point clouds that can be used to visualize animated point clouds.

Our newly acquired knowledge was applied while building our own low-cost multi-camera system. We implemented basic controls over this system, including two solutions for synchronized acquisition without special hardware, of which one was good enough to be used.

We proposed and implemented an animated point cloud pipeline that utilizes the current photogrammetry software solution. This pipeline starts with synchronized acquisition and ends with visualization of the animated point cloud, for which we implemented our own visualizer.

Several experiments were performed, capturing scenes that contain both static and dynamic objects. It turned out that even our low-cost multi-camera system, together with current photogrammetric software, performs successful and accurate reconstructions, provided that we have enough overlapping camera views. The proposed pipeline worked for both simulated and real image data sets.

This thesis introduced a new way to reconstruct and visualize dynamic scenes. Other multi-camera systems, not only those for photogrammetry, may implement our proposed pipeline with slight modifications. This solution may especially be helpful in reviewing simulated medical procedures (e.g., resuscitation), where surveillance cameras overwatching participants are already installed. This may give new insight into simulation evaluation because the scene could be reviewed from more angles than only cameras provide.

Although our multi-camera system made successful reconstructions using only eight cameras, it would be interesting to see how it will perform with more and better cameras. We could also calibrate the system as a whole, not just individual cameras, as we did. This would result in known external camera parameters that could speed up and improve reconstructions.

The resulting animated point clouds could also be visualized differently, for example, in virtual or augmented reality, where fiducial markers could also be used for alignment. This would introduce a whole new way of reviewing reconstructions.

Bibliography

1. MALLISON, Heinrich. *My "self-built" photogrammetry rig* [online]. Tübingen: Mallison, 2020 [visited on 2022-03-30]. Available from: `https://dinosaurpalaeo.wordpress.com/2020/12/03/my-self-built-photogrammetry-rig`.
2. LANZ, Oswald; SOTTSAS, Fabian; CONNI, Michele; BOSCHETTI, Marco; NOCERINO, Erica; MENNA, Fabio; REMONDINO, Fabio. A versatile multi-camera system for 3D acquisition and modeling. *International archives of the photogrammetry, remote sensing and spatial information sciences*. 2020, vol. 43, pp. 785–790. Available from DOI: `10.5194/isprs-archives-XLIII-B2-2020-785-2020`.
3. *About: Photogrammetry 3D Scanner* [online]. Baltimore: UMBC, 2015 [visited on 2022-03-30]. Available from: `https://photogrammetry.irc.umbc.edu/about`.
4. VAŠTA, Jakub. *Extrakce statického modelu lidského obličeje ze stereo/multiview videa*. Plzeň, 2020. Available also from: `http://hdl.handle.net/11025/41747`. Master thesis. Západočeská univerzita v Plzni, Fakulta aplikovaných věd, Katedra informatiky a výpočetní techniky.
5. WENGER, Stephan. H.264/AVC over IP. *IEEE Transactions on Circuits and Systems for Video Technology*. 2003, vol. 13, no. 7, pp. 645–656. Available from DOI: `10.1109/TCSVT.2003.814966`.
6. LUHMANN, Thomas; ROBSON, Stuart; KYLE, Stephen; BOEHM, Jan. *Close-range photogrammetry and 3D imaging*. 2nd ed. Berlin: De Gruyter, 2013. ISBN 978-3-11-030269-1. Available from DOI: `10.1515/9783110302783`.
7. *The Bayer arrangement of color filters on the pixel array of an image sensor* [online]. San Francisco (CA): Wikimedia Foundation, 2001 [visited on 2022-03-31]. Available from: `https://en.wikipedia.org/wiki/Bayer_filter%5C#/media/File:Bayer_pattern_on_sensor.svg`.
8. HARTLEY, Richard; ZISSERMAN, Andrew. *Multiple view geometry in computer vision*. 2nd ed. Cambridge: Cambridge University Press, 2003. ISBN 978-0521540513.
9. BROWN, Duane C. Decentering distortion of lenses. *Photogrammetric Engineering and Remote Sensing*. 1966, vol. 32, no. 3, pp. 444–462. ISSN 991112.
10. BRADSKI, Gary R.; KAEHLER, Adrian. *Learning OpenCV*. Sebastopol: O'Reilly, 2008. ISBN 978-0596516130.
11. ST 12-1:2014 - SMPTE Standard - Time and Control Code. *ST 12-1:2014*. 2014, pp. 1–41. Available from DOI: `10.5594/SMPTE.ST12-1.2014`.

12. LITOS, Georgios; ZABULIS, Xenophon; TRIANTAFYLLIDIS, Georgios. Synchronous Image Acquisition based on Network Synchronization. In: *2006 Conference on Computer Vision and Pattern Recognition Workshop (CVPRW'06)*. 2006, p. 167. Available from DOI: 10.1109/CVPRW.2006.200.

13. BIERY, Ethan; SHEARER, Thomas; LEDYARD, Roland; PERKINS, Dan; FERIS, Manny. *Controlling LEDs* [online]. Lutron, 2014 [visited on 2022-04-08]. Available from: https://www.lutron.com/TechnicalDocumentLibrary/367-2035_LED_white_paper.pdf.

14. CALVET, Lilian; GURDJOS, Pierre; GRIWODZ, Carsten; GASPARINI, Simone. Detection and Accurate Localization of Circular Fiducials under Highly Challenging Conditions. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 562–570. Available from DOI: 10.1109/CVPR.2016.67.

15. LOPEZ DE IPINA, Diego; MENDONÇA, Paulo R. S.; HOPPER, Andy. TRIP: A low-cost vision-based location system for ubiquitous computing. *Personal and Ubiquitous Computing*. 2002, vol. 6, no. 3, pp. 206–219. Available from DOI: 10.1007/s007790200020.

16. OLSON, Edwin. AprilTag: A robust and flexible visual fiducial system. In: *2011 IEEE International Conference on Robotics and Automation*. 2011, pp. 3400–3407. Available from DOI: 10.1109/ICRA.2011.5979561.

17. FIALA, Mark. ARTag, a fiducial marker system using digital techniques. In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*. 2005, vol. 2, pp. 590–596. Available from DOI: 10.1109/CVPR.2005.74.

18. MALLISON, Heinrich. *My "self-built" photogrammetry rig* [online]. Tübingen: Mallison, 2020 [visited on 2022-03-30]. Available from: https://dinosaurpalaeogerman.files.wordpress.com/2016/04/rig_01.jpg.

19. XIAO, Yong-Liang; XUE, Junpeng; SU, Xianyu. Robust self-calibration three-dimensional shape measurement in fringe-projection photogrammetry. *Optics letters*. 2013, vol. 38, no. 5, pp. 694–696. ISSN 0146-9592.

20. *Ecomimesis* [online]. Baltimore: Lynn Cazabon, 2018 [visited on 2022-04-23]. Available from: https://www.lynncazabon.com/ecomimesis.

21. *As a test subject, Chris Peregoy, Visual Arts, along with his antique camera, is being 3D scanned in the newly installed 90 digital camera rig* [online]. Baltimore: UMBC, 2015 [visited on 2022-03-30]. Available from: https://photogrammetry.irc.umbc.edu/images/ChrisPeregoy.jpg.

22. *Photogrammetry Pipeline* [online]. AliceVision [visited on 2022-04-10]. Available from: https://alicevision.org.

23. SCHÖNBERGER, Johannes L.; FRAHM, Jan-Michael. Structure-from-Motion Revisited. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 4104–4113. Available from DOI: 10.1109/CVPR.2016.445.

24. CRANDALL, David; OWENS, Andrew; SNAVELY, Noah; HUTTENLOCHER, Dan. Discrete-continuous optimization for large-scale structure from motion. In: *CVPR 2011*. 2011, pp. 3001–3008. Available from DOI: 10.1109/CVPR.2011.5995626.

25. MOULON, Pierre; MONASSE, Pascal; MARLET, Renaud. Global Fusion of Relative Motions for Robust, Accurate and Scalable Structure from Motion. In: *2013 IEEE International Conference on Computer Vision*. 2013, pp. 3248–3255. Available from DOI: 10.1109/ICCV.2013.403.

26. GHERARDI, Riccardo; FARENZENA, Michela; FUSIELLO, Andrea. Improving the efficiency of hierarchical structure-and-motion. In: *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. 2010, pp. 1594–1600. Available from DOI: 10.1109/CVPR.2010.5539782.

27. HAVLENA, Michal; TORII, Akihiko; PAJDLA, Tomáš. Efficient Structure from Motion by Graph Optimization. In: Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, vol. 6312, chap. 2, pp. 100–113. Computer Vision – ECCV 2010. ISSN 0302-9743.

28. TOLDO, Roberto; GHERARDI, Riccardo; FARENZENA, Michela; FUSIELLO, Andrea. Hierarchical structure-and-motion recovery from uncalibrated images. *Computer Vision and Image Understanding*. 2015, vol. 140, pp. 127–143. ISSN 1077-3142. Available from DOI: 10.1016/j.cviu.2015.05.011.

29. SHAH, Rajvi; DESHPANDE, Aditya; NARAYANAN, P.J. Multistage SFM: Revisiting Incremental Structure from Motion. In: *2014 2nd International Conference on 3D Vision*. 2014, vol. 1, pp. 417–424. Available from DOI: 10.1109/3DV.2014.95.

30. LOWE, David G. Distinctive Image Features from Scale-Invariant Keypoints. *International journal of computer vision*. 2004, vol. 60, no. 2, pp. 91–110. ISSN 0920-5691.

31. ALCANTARILLA, Pablo; NUEVO, Jesus; BARTOLI, Adrien. Fast Explicit Diffusion for Accelerated Features in Nonlinear Scale Spaces. In: *Proceedings of the British Machine Vision Conference*. BMVA Press, 2013. Available from DOI: 10.5244/C.27.13.

32. TAREEN, Shaharyar Ahmed Khan; SALEEM, Zahra. A comparative analysis of SIFT, SURF, KAZE, AKAZE, ORB, and BRISK. In: *2018 International Conference on Computing, Mathematics and Engineering Technologies (iCoMET)*. 2018, pp. 1–10. Available from DOI: 10.1109/ICOMET.2018.8346440.

33. MUJA, Marius; LOWE, David. Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration. *VISAPP 2009 - Proceedings of the 4th International Conference on Computer Vision Theory and Applications*. 2009, vol. 1, pp. 331–340.

34. FISCHLER, Martin; BOLLES, Robert. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*. 1981, vol. 24, no. 6, pp. 381–395. ISSN 0001-0782.

35. BEDER, Christian; STEFFEN, Richard. Determining an Initial Image Pair for Fixing the Scale of a 3D Reconstruction from an Image Sequence. In: Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, vol. 4174, pp. 657–666. Pattern Recognition. ISSN 0302-9743.

36. LEPETIT, Vincent; MORENO-NOGUER, Francesc; FUA, Pascal. EPnP: An Accurate O(n) Solution to the PnP Problem. *International Journal of Computer Vision*. 2008, vol. 81, no. 2, p. 155. ISSN 1573-1405. Available from DOI: 10.1007/s11263-008-0152-6.

37. TRIGGS, Bill; MCLAUCHLAN, Philip F.; HARTLEY, Richard I.; FITZGIBBON, Andrew W. Bundle Adjustment — A Modern Synthesis. In: Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, vol. 1883, pp. 298–372. Vision Algorithms: Theory and Practice. ISSN 0302-9743.

38. FURUKAWA, Yasutaka; HERNANDEZ, Carlos. *Multi-view stereo: a tutorial*. Boston: Now Publishers, 2015. ISBN 978-1-60198-836-2.

39. HIRSCHMULLER, Heiko. Accurate and efficient stereo processing by semi-global matching and mutual information. In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*. 2005, vol. 2, 807–814 vol. 2. Available from DOI: 10.1109/CVPR.2005.56.

40. HIRSCHMULLER, Heiko. Stereo Processing by Semiglobal Matching and Mutual Information. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 2008, vol. 30, no. 2, pp. 328–341. Available from DOI: 10.1109/TPAMI.2007.1166.

41. MEI, Xing; SUN, Xun; ZHOU, Mingcai; JIAO, Shaohui; WANG, Haitao; ZHANG, Xiaopeng. On building an accurate stereo matching system on graphics hardware. In: *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*. 2011, pp. 467–474. Available from DOI: 10.1109/ICCVW.2011.6130280.

42. *3DF Zephyr* [online]. Verona: 3Dflow, 2022 [visited on 2022-04-11]. Available from: https://www.3dflow.net/3df-zephyr-photogrammetry-software.

43. *Agisoft* [online]. St. Petersburg: Agisoft, 2022 [visited on 2022-04-11]. Available from: https://www.agisoft.com.

44. GRIWODZ, Carsten; GASPARINI, Simone; CALVET, Lilian; GURDJOS, Pierre; CASTAN, Fabien; MAUJEAN, Benoit; LILLO, Gregoire De; LANTHONY, Yann. AliceVision Meshroom: An open-source 3D reconstruction pipeline. In: *Proceedings of the 12th ACM Multimedia Systems Conference - MMSys '21*. ACM Press, 2021. Available from DOI: 10.1145/3458305.3478443.

45. *RealityCapture* [online]. Epic Games Slovakia, 2022 [visited on 2022-04-11]. Available from: https://www.capturingreality.com/realitycapture.

46. *WHAT ARE POINT CLOUDS?* [Online]. Aberdeen: Tech27 systems, 2020 [visited on 2022-04-12]. Available from: https://tech27.com/resources/point-clouds.

47. COMMUNITY, Blender Online. *Blender - a 3D modelling and rendering package*. Stichting Blender Foundation, Amsterdam, 2018. Available also from: http://www.blender.org.

48. ZHOU, Qian-Yi; PARK, Jaesik; KOLTUN, Vladlen. *Open3D: A Modern Library for 3D Data Processing*. arXiv, 2018. Available from DOI: 10.48550/ARXIV.1801.09847.

49. CIGNONI, Paolo; CALLIERI, Marco; CORSINI, Massimiliano; DELLEPIANE, Matteo; GANOVELLI, Fabio; RANZUGLIA, Guido. MeshLab: an Open-Source Mesh Processing Tool. In: SCARANO, Vittorio; CHIARA, Rosario De; ERRA, Ugo (eds.). *Eurographics Italian Chapter Conference*. The Eurographics Association, 2008. ISBN 978-3-905673-68-5. Available from DOI: 10.2312/LocalChapterEvents/ItalChap/ItalianChapConf2008/129-136.

50. OPENCV. *Open Source Computer Vision Library* [software]. 2018. Version 4.0.1. Available also from: https://opencv.org.

51. VAN ROSSUM, Guido; DRAKE, Fred L. *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009. ISBN 1441412697.

52. ANACONDA SOFTWARE DISTRIBUTION. *Anaconda* [software]. 2016. Version 4.10.1. Available also from: https://anaconda.com.

53. HIKVISION DIGITAL TECHNOLOGY. *DS-2CD2723G1-IZ(S): 2 MP IR Varifocal Dome Network Camera* [online]. Hikvision Digital Technology, 2019. Available also from: https://www.hikvision.com/content/dam/hikvision/products/S000000001/S000000002/S000000003/S000000025/OFR000038/M000000137/Data_Sheet/DS-2CD2723G1-IZS_Datasheet_V5.6.0_20210326.pdf.

54. ARUBA NETWORKS. *ARUBA 2530 48 POE+ SWITCH (J9778A): Fixed Port L2 Managed Ethernet Switches* [online]. Hewlett Packard Enterprise Development, 2022. Available also from: https://www.hpe.com/psnow/doc/PSN5384996USEN.pdf.

55. EPIC GAMES. *Unreal Engine* [software]. 2021. Version 4.27.0. Available also from: https://www.unrealengine.com.

56. BLENDER ONLINE COMMUNITY. *Blender* [software]. 2022. Version 3.1.2. Available also from: https://www.blender.org/download.

57. HIKVISION DIGITAL TECHNOLOGY. *iVMS-4200* [software]. Hikvision Digital Technology, 2022. Version 3.6.1.5. Available also from: https://www.hikvision.com/cz/support/download/software/ivms4200-series.

58. REITZ, Kenneth. *Requests: HTTP for Humans* [software]. 2022. Version 2.27.1. Available also from: https://docs.python-requests.org/en/latest/user/install.

59. HIKVISION DIGITAL TECHNOLOGY. *Intelligent Security API (General): Developer Guide*. Hikvision Digital Technology, 2020.

60. RODOLA, Giampaolo. *pyftpdlib* [software]. 2018. Version 1.5.4. Available also from: `https://pyftpdlib.readthedocs.io/en/latest/install.html`.

61. PROJECT JUPYTER. *jupyter notebook* [software]. 2014. Available also from: `https://jupyter.org/install`.

62. MESHROOM CONTRIBUTORS. *Meshroom* [online]. Meshroom Contributors, 2021. Version 2021.0.1. Available also from: `https://meshroom-manual.readthedocs.io/en/latest`.

Contents of the attached media

- `calibration` ..... directory with calibration images and files
- `experiments` ..... directory with data from experiments
- `Meshroom-2021.1.0` ..... Meshroom prebuilt binaries for Windows OS
- `other`
  - `camera_configs` ..... directory with exported camera configurations
  - `printable_cctags` ..... directory with printable CCTags
  - `env.yml` ..... Anaconda environment file
  - `isapi.pdf` ..... HIKVISION ISAPI documentation in PDF format
  - `meshroom_manual.pdf` ..... meshroom manual in PDF format
- `src`
  - `modules` ..... source codes of the implementated modules
  - `thesis` ..... LaTeX source codes of the thesis
  - `pipeline.ipynb` ..... described usage of the implemented pipeline
- `video` ..... recorded animated point clouds in visualizer
  - `human_interaction.mp4`
  - `moving_box.mp4`
  - `simulated_moving_box.mp4`
- `readme.txt` ..... additional information about the media content
- `thesis.pdf` ..... the thesis text in PDF format