TU WIEN Informatics

Stereoscopic Panorama Stitching

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Visual Computing

eingereicht von

Tobias Sippl, B.Eng.
Matrikelnummer 0820552

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Ao.Univ.Prof. Dipl.-Ing. Mag. Dr. Margrit Gelautz

Wien, 9. Februar 2022

Tobias Sippl	Margrit Gelautz

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

TU WIEN Informatics

Stereoscopic Panorama Stitching

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Visual Computing

by

Tobias Sippl, B.Eng.
Registration Number 0820552

to the Faculty of Informatics

at the TU Wien

Advisor: Ao.Univ.Prof. Dipl.-Ing. Mag. Dr. Margrit Gelautz

Vienna, 9th February, 2022

Tobias Sippl	Margrit Gelautz

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Erklärung zur Verfassung der Arbeit

Tobias Sippl, B.Eng.

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 9. Februar 2022

Tobias Sippl

Kurzfassung

Virtual-Reality-Headsets sind heute leichter verfügbar als jemals zuvor. Das Feld der Telepräsenz, welches versucht dem Benutzer das Gefühl zu vermitteln an einem anderen Ort zu sein, hat besonderes Interesse an diesen Technologien, da sich die Interaktion über Virtual-Reality-Headsets wesentlich natürlicher gestaltet, als über klassische Bildschirme. In dieser Arbeit wird die Ausarbeitung und Implementierung eines Frameworks besprochen, welches es ermöglicht, in einem großteils automatisierten Prozess aus zeitgleich von Kameras aufgenommenen Bildern mit einigen Überlappungen, ein vollständiges Panoramabild zu erzeugen. Um die Verwendung einer Vielzahl an Kamerasystemen zu ermöglichen, wird ein Kameramodell verwendet, welches verschiedene Linsentypen unterstützt. Ein auf Bundle-Adjustment basierender Algorithmus unterstützt dann das Ausrichten der Kameras auf Basis von Feature-Point Übereinstimmungen. Außerdem wird ein Moving-Least-Squares-Verfahren angewandt, um auffällige „Nahtstellen“ zwischen den Bildern zu reduzieren. Aus diesem Ergebnis lässt sich ein Mapping exportieren, woraus in Echtzeit neue Panoramabilder erzeugt werden können - vorausgesetzt, das Kamera-Setup wird nicht verändert. Der Algorithmus wird anhand eines synthetischen Datensets evaluiert, das mehrere Szenen und Konfigurationsmöglichkeiten einschließt und fast hundert unterschiedliche Test-Szenarien beinhaltet. Anschließend wird analysiert, wie sich unterschiedliche Veränderungen an Setup oder Schritten des Algorithmus auf die Effizienz des Algorithmus auswirken. Es wurde festgestellt, dass eine Kombination aus Feature-Point-Matching, Bundle-Adjustment und Moving-Least-Squares-Verfahren ein visuell ansprechendes Panoramabild ermöglicht. Außerdem wurde beobachtet, dass Cubemap Reprojection die Ergebnisse, die ein normaler Feature-Point-Matching-Agorithmus bei Anwendung auf radial verzerrte, kreisförmige Bilder erzielt, verbessern kann.

Abstract

Virtual reality devices are more easily available today than ever before. The field of telepresence, the objective of which it is to allow a person to feel as if they are present at a distant location, is interested in virtual reality devices, as they present a much more natural interface than a common screen. Through a virtual reality headset, a full panoramic image can be examined by looking in the direction interesting to the viewer. This thesis describes the design and implementation of a framework which allows to create a full panorama from images taken simultaneously by cameras with some overlap, in a largely automated way. We use a camera model which covers a variety of high field-of-view camera systems in order to allow for a large base of different lens types. Next, a full panoramic image is stitched with a bundle adjustment strategy on the basis of feature-point matching, and a moving least squares approach is used to reduce noticeable seams. The result can be exported to a template, from which new panoramic images can be created in real time, given an unaltered camera setup. We evaluate the algorithm against a synthetic dataset which consists of multiple scenes and configurations resulting in almost a hundred different test cases. We analyze how various changes to the setup or steps in the algorithm affect the efficacy of the algorithm. We find that a combination of feature-point matching, bundle adjustment and a moving least squares approach produces a visually pleasing panoramic image. Further, we observe that cubemap reprojection can improve the results of the applied feature-point matching algorithms on radially distorted circular images.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction and Problem Statement

1.1 Motivation

Recently there has been a renewed interest in Virtual Reality (VR) and Augmented Reality (AR), which has resulted in a new generation of Head Mounted Displays (HMD) and investment in the research and development of closely related fields. The most important part of any VR system is the HMD, consisting of two main components. Firstly, a display positioned closely to the user's eyes in combination with an optical lens to cover a large field of view. Secondly, a suite of sensors (depending on the type of HMD this may vary) which allows the positional and rotational tracking of the device [CC13].

In HMDs, the spatial information can be used to adjust the images to the viewing direction of the user unlike in other conventional displays. The turning of a user's head can be directly translated into rotational information and the view in the virtual world can be updated accordingly. The user's entire field of view is covered by the screen and the controls are highly intuitive. For most users, this produces the illusion of being at the place which is shown inside the HMD [CC13]. In telepresence and teleoperation, these are highly sought-after properties. The immersive experience combined with the intuitive controls results in a strong feeling of remote presence.

For any application of this technology, a remote location has to be recorded by a camera. If the user then moves their head, the viewing direction should change. Synchronizing the remote cameras to head movement would introduce considerable lag, resulting in cybersickness [PCT+17], which in turn would render the system useless. Instead, a panorama which is combined from multiple cameras is transmitted to the user for exploration through the HMD.

Choosing the appropriate camera setup is a complex task as the requirements differ for each individual system. The general boundaries of such a system are set by the desired field of view of the panorama and the visual quality expected of the result. Among other properties, each camera is defined by a field of view, a sensor resolution and a frame rate. As these characteristics are interdependent, a higher resolution results in a lower frame rate, while a higher field of view reduces the image quality. Each additional camera also introduces a visual discontinuity to the panorama and increases the hardware requirements when combining the images. For a stereoscopic panorama, two cameras must observe the same space, doubling the amount of cameras necessary if the entire panorama is to be displayed in stereo.

After an appropriate setup has been chosen, the cameras must be calibrated to remove distortions and arranged properly to produce a panorama. Aligning the images by hand is a time-consuming task which can be automated and improved by computer vision algorithms.

This panorama can then finally be transmitted and viewed remotely. To support a wide variety of remote systems, a combination of techniques is chosen to work on embedded platforms in order to reduce the hardware requirements on the remote end. This thesis tackles the problem of realizing this pipeline and developing tools for rapid testing of varying setups, allowing users to create and view stereoscopic panoramas.

In the process of developing this pipeline, various key issues have to be solved. Describing the problems which arise from this kind of setup and subsequently providing solutions will be of special interest to the scientific community and practicioners engaged in computer vision or working on similar topics.

1.2 Aim of the Paper

The primary goal of this paper is to design and implement a tool which allows for the rapid calibration and configuration of a stereoscopic panorama which is stitched from multiple cameras with the intent to display it inside a head mounted display. The configuration of the panorama should be assisted by computer vision algorithms and usable by non-expert users. The result produced by the tool is a map which produces a panorama from the source images of the cameras. This map, from here on referred to as panorama generation template, can then be used to stitch a panorama from multiple cameras in real time, unless the camera configuration or the surroundings are changed.

1.2.1 Generalizability

The tool should be able to accomodate a wide range of cameras, especially those used in high field of view photography and robotics. Various camera layouts should be allowed within the tool, without principal restrictions to the amount of cameras while allowing for (partial or full) stereo-vision. Finally, the panorama-map should be in a simple format so that other software can easily make use of it.

1.2.2 Interactivity

Configuring a panorama typically involves a visual component. Cameras are often rearranged, their calibrations modified or blending parameters adjusted. In those cases, the possibility to visually inspect the panorama is desired. A responsive graphical user interface (GUI) should immediately reflect any modifications and allow to change any associated parameter. Generating a panorama is a computationally expensive step, which has to be executed each time a change is made to any part of the setup. For multiple cameras and a panorama size at typical screen resolutions, a common CPU would not be able to produce the panorama in real time. By processing the panorama on a GPU, which is optimized for parallel computation, real time updates can be achived on common hardware. Nowadays many modern processors contain an integrated GPU.

1.3 Contribution

The primary contribution is to provide an overview of the challenges which arise when developing multi-camera computer vision software with high field of view cameras and suggest possible solutions. Many modern computer vision tasks require a similar setup and thus produce similar problems. To this extent, the thesis covers all steps, from describing the components which make up a panorama system, to the algorithms used to calibrate and stitch the panorama. This thesis also contains an algorithm based on common techniques used in computer vision which can be used to stitch a spheric panorama..

1.4 Structure of the Thesis

Chapter 2 introduces the topic of image alignment and stitching and explains some of the challenges and solutions that exist in the field. Chapter 3 considers the hardware which is typically used when producing panoramas in a live fashion. This includes the cameras, lenses and displays used in such a setup. Chapter 4 then explains the pipeline which was established while working on this thesis. The graphical user interface which allows for interaction with the pipeline is shown and explained in detail in Chapter 5. The results produced by the pipeline will then be demonstrated and analysed in Chapter 6 before a conclusion and considerations for future work will be presented in the final Chapter 7.

CHAPTER 2

Background and Related Work

2.1 Image Alignment and Stitching

According to Szeliski [Sze06], image alignment (registration) algorithms can discover the correspondence relationships among images with varying degrees of overlap. Image stitching then uses these estimated relationships in order to combine multiple images in a seamless manner (see Figure 2.1 for a schematic example) while taking care of potential problems like blurring or ghosting caused by parallax and scene movement as well as varying image exposures. These algorithms have a long history in computer vision due to their many applications. Common examples of everyday use are the stitching of satellite images into continuous seamless aerial maps [AAMDG12] or panoramic images formed from multiple shots taken by mobile phones [XP10].

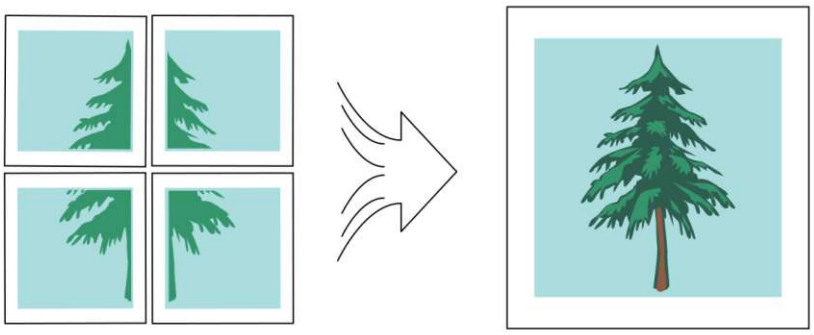


Figure 2.1: Four images are combined into a single larger image.

An early example for a widely used image registration algorithm is the optical flow tech-

nique by Lucas and Kanade [LK81] which is able to find similar regions under rotation as well as scaling and shearing distortions between multiple images.

The general problem of stitching a continous image covering a very wide field of view is well researched [BL07]. The specific problem this thesis is focused on is the forming of a panoramic image from multiple overlapping images with a similar but not identical optical center. The steps involved in order to solve such a task vary according to the type of implementation but generally consist of a camera calibration, followed by registering or aligning the images before they are merged into a single panorama. An example for such an operation is shown at the end of this chapter in Figure 2.3.

2.2 Motion Models & 3D Transforms

In our work, a combination of 2D motion models and 3D transforms is used to establish a model for transforming image information between two images which share some overlap. Our model, described in Section 4.6, accounts for various forms of distortions, two of which, skew and principal point offset, are modelled by 2D transformations. The rotations between cameras will be accounted for by 3D transformations, which are also based on 2D motions. Another important step in our model is central projection, which describes how an image is recorded by a pinhole camera and can also be modelled as a 3D transformation. The following definitions are those provided by Szeliski [Sze06].

Translation

Translation moves every point p from an image by the same vector t and can be written as $\boldsymbol{p}' = \boldsymbol{p} + \boldsymbol{t}$ or

$$\boldsymbol{p}' = [\boldsymbol{I}\ \boldsymbol{t}]\tilde{\boldsymbol{p}}, \tag{2.1}$$

where $\boldsymbol{I}$ is the (2×2) identity matrix, $\boldsymbol{t}$ is the 2D vector by which the point is to be translated and $\tilde{\boldsymbol{p}} = (x, y, 1)$, the homogenous 2D coordinate.

Scaling

Scaling transforms the position along a line from the image center through $\boldsymbol{p}$ depending on the distance from the center. It can be written as $s\boldsymbol{p}$ or

$$\boldsymbol{p}' = \left[s\boldsymbol{I}\right] p \tag{2.2}$$

where s is a 2D vector by which p will be scaled.

Rotation

Rotation rotates evey point p around the center by θ degrees and can be written as $\boldsymbol{p}' = \boldsymbol{R}\boldsymbol{p}$ where

$$\boldsymbol{R} = \begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix}. \tag{2.3}$$

The combination of translation and rotation is also known as the 2D rigid body motion or the 2D Euclidean transformation:

$$\boldsymbol{p}' = \begin{bmatrix} \boldsymbol{R} & \boldsymbol{t} \end{bmatrix} \tilde{\boldsymbol{p}}. \tag{2.4}$$

The combination of scaling, rotation and translation is called similarity transform and given as:

$$\boldsymbol{p}' = \begin{bmatrix} s\boldsymbol{R} & \boldsymbol{t} \end{bmatrix} \tilde{\boldsymbol{p}}. \tag{2.5}$$

Projective Transform

Projective transform, also known as perspective transform or homography, produces a result which preserves straight lines. It is written as

$$\tilde{\boldsymbol{p}}' \sim \tilde{\boldsymbol{H}}\boldsymbol{p}, \tag{2.6}$$

with $\sim$ describing a similarity up to scale, $\tilde{\boldsymbol{H}}$ is any (3×3) matrix.

2.2.1 3D Transformations

Central Projection

A camera maps a point $\boldsymbol{P} = (X, Y, Z)$ through its optical center, the pinhole, to the 2D coordinates $\boldsymbol{p} = (x, y)$ on a plane at distance f from the pinhole along the z-axis. This process is called central projection and is written as:

$$x = f\frac{X}{Z}, y = f\frac{Y}{Z}. \tag{2.7}$$

Figure 2.2 illustrates this projection.

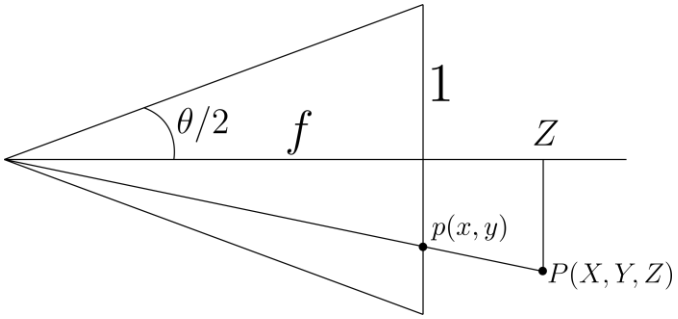


Figure 2.2: Central Projection of a point P in 3D space to p in 2D image space.

The projective matrix is commonly given as:

$$\tilde{\boldsymbol{p}} \sim \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \boldsymbol{p} = \left[\boldsymbol{K}|\boldsymbol{0}\right] \boldsymbol{p} = \boldsymbol{C}\boldsymbol{p}. \tag{2.8}$$

To also account for non-square pixels, skew and a variable optic center location, K, can be expanded to:

$$\boldsymbol{K} = \begin{bmatrix} \alpha_x & \gamma & u_0 \\ 0 & \alpha_y & v_0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{2.9}$$

Here, $\alpha_x = f * m_x$ and $\alpha_y = f * m_y$ where m_x and m_y are the scale factors in pixels, γ is the skew coefficient between the x and y axis. u_0 and v_0 denote the principal point or center of the image.

If a position is imaged by a camera under perspective projection, the result is produced by a combination of a 3D rigid-body motion $\boldsymbol{E}$ and a perspective projection $\boldsymbol{C}$:

$$\tilde{\boldsymbol{p}} = \boldsymbol{C} \begin{bmatrix} \boldsymbol{R} & \boldsymbol{t} \\ \boldsymbol{0}^T & 1 \end{bmatrix} \boldsymbol{p} = \boldsymbol{C}\boldsymbol{E}\boldsymbol{p} \tag{2.10}$$

2.2.2 Lens Distortion

Lenses normally induce some kind of distortion, usually in the form of radial distortion, which either contracts the image towards the image center, or expands it away from it.

A polynomial is used in order to model this type of distortion with 2 or 3 parameters:

$$\begin{aligned} x' &= x(1 + \kappa_1 r^2 + \kappa_2 r^4), \\ y' &= y(1 + \kappa_1 r^2 + \kappa_2 r^4), \end{aligned} \tag{2.11}$$

where r is the distance of point (x, y) from the image center ($r^2 = x^2 + y^2$) and κ_1 and κ_2 are the polynomial parameters. The lens model used in this thesis additionally considers tangential distortion and the entire list of paramters can be found in Section 4.3.2.

2.3 Alignment

With a combination of lens distortion (eq. 2.11) and camera transformations (eq. 2.10) we can transfer positions from one camera to another. This is then the basis for constructing panoramas. In most cases, the exact position, rotation and distortion of a camera are not known at the time of recording an image. Estimating these parameters is handled in the alignment or registration step of stitching algorithms. The first type of approaches for image alignment are called direct or pixel-based methods, which directly operate on the image pixel information. The currently more broadly used set of approaches are feature-based approaches, which first extract a set of robust features to be matched between the images. The two categories of approaches are explained in more detail in subsections 2.3.1 and 2.3.2.

2.3.1 Direct Methods

Direct methods use an error metric which describes how well two images overlap. An example for a simple metric would be the color difference between two overlapping images at each pixel. A search technique must then be used in order to find the alignment which minimizes that error. Simply testing all possible alignments will usually be too time-consuming and coarse-to-fine methods or Fourier-transform-based methods can be used to speed up the process [Ana89] [BAHH92] [OBS99].

2.3.2 Feature Registration

The major approaches in current image stitching applications are based on feature points. First, feature points are extracted from all involved images and then a global relationship between matching features is established. Generally speaking, a feature is an image region which has some properties a feature detector is designed to respond to. Some examples would be corners, image intensity or texture. After a feature has been found, a feature descriptor will then be computed to describe it. Depending on the requirements, varying properties are sought after when designing the detectors and descriptors. Typically, features are to be robust against noise, illumination, scale, rotation and occlusion so that multiple images taken of the same object from varying angles, distances or by different

cameras, provide the same or similar features. The feature descriptor, which encodes the area around the feature, should then allow a measure of similarity when comparing descriptors. Examples of feature detector and descriptors are SIFT, SURF, BRISK and ORB. Detector-only algorithms like FAST or descriptor-only algorithms like FREAK also exist [Low04] [RRKB11] [AOV12]. Matching features between two images will in most cases contain some false matches. A common strategy to resolve this issue is called random sample consensus or RANSAC [FB81] [RFP08]. A transformation is estimated from a subset of matches and then used to transform all features into the other image. The amount of transformed features which lie within a radius of their detected matches are then counted. This step is repeated a number of times with varying subsets and the result with the most inliers is used.

Parameter Estimation

With a set of feature correspondences, the parameters for the motion model now can be estimated. Usually a least squares error is minimized in the form of:

$$E_{LS} = \sum_i ||\boldsymbol{r}_i||^2 = ||\tilde{\boldsymbol{p}}'_i(\boldsymbol{p}_i; params) - \hat{\boldsymbol{p}}'_i||^2. \tag{2.12}$$

The total error is the sum of all errors from i point correspondences. Each individual error is a distance derived from the two positions of a point correspondence. $\tilde{\boldsymbol{p}}'_i$ is the position of a point in the first image transformed to the second image and $\hat{\boldsymbol{p}}'_i$ is the position of the matching point in the second image. This can be extended with a loss function [Bar19] in order to reduce the influence of outliers or feature matches with less accuracy.

2.3.3 Global Registration

To establish a relationship between multiple images, a global strategy must be chosen to reduce the alignment and pose errors when transforming them to a combined image [CT00]. A common way of achieving this is the use of bundle adjustment. Where previously an error metric was chosen to minimize the error between two images only, bundle adjustment considers all feature correspondences at once. The formulation follows Szeliski [Sze06], although slightly modified. Extending the parameter estimation equation 2.12 to include all cameras results in:

$$E_{ALL} = \sum_i \sum_{jk} ||\tilde{\boldsymbol{p}}_{ik}(\hat{\boldsymbol{p}}_{ij}; \boldsymbol{R}_j, \boldsymbol{K}_j, \boldsymbol{R}_k, \boldsymbol{K}_k) - \hat{\boldsymbol{p}}_{ik}||^2. \tag{2.13}$$

Here the total error is summed over all features i which appear in both images j and k. where $\tilde{\boldsymbol{p}}_{ik}$ is the predicted position of feature i in image k with $\hat{\boldsymbol{p}}_{ik}$ being the observed position in image k. This formulation optimizes for the rotational parameters $\boldsymbol{R}_j$ and $\boldsymbol{R}_k$ and the intrinsic camera parameters $\boldsymbol{K}_j$ and $\boldsymbol{K}_k$.

2.3.4 Parallax & Artifact Removal

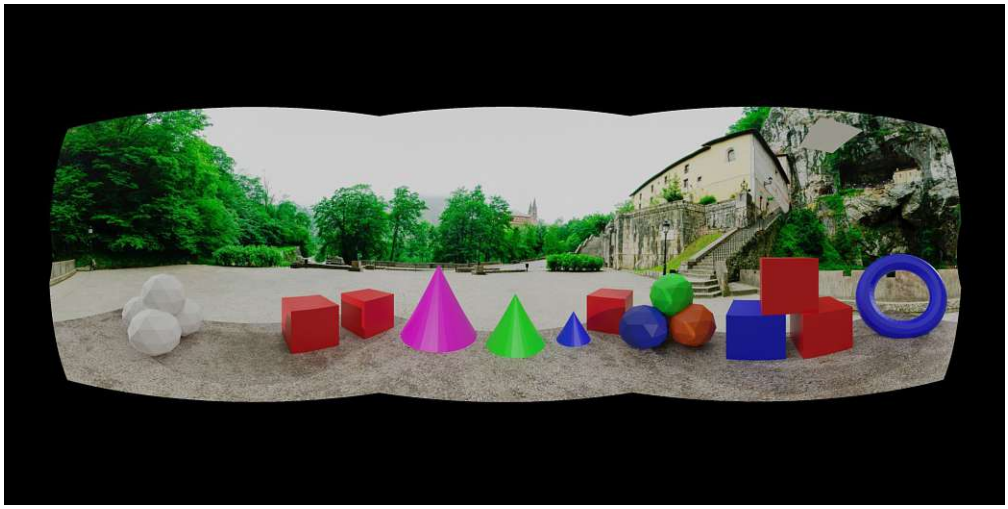
After finding the camera positions and orientations, there may be still some visual issues after stitching the panorama. Recording the same scene from different positions can produce 3D parallax, as usually the closer an object is, the more a change in camera position will affect its appearence in the image. Another form of artifact often present is ghosting, which occurs if an object is moved in-between taking images. There might also be unmodelled distortion or more, depending on the camera setup. 3D parallax can be reduced by full bundle adjustment and reprojection to a 3D scene reconstruction [Sze06]. Ghosting can be removed by selecting pixels only from images in order to only include the object once [PAG11]. Unaccounted lens distortion may be dealt with by chosing a more suitable camera model.

2.3.5 Compositing and Blending

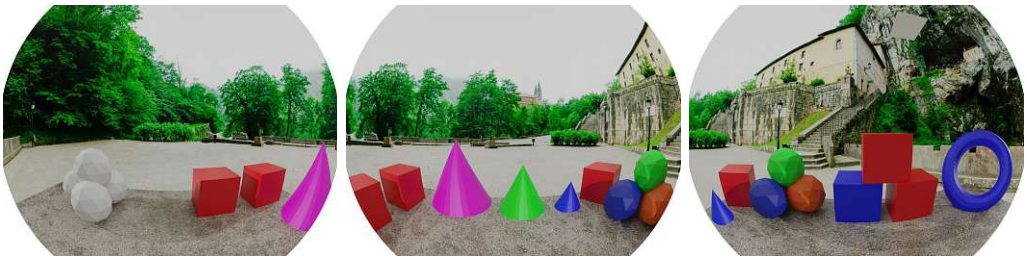
With all images in place and properly aligned, the images must be synthesized into a single panorama. Depending on the use of the panorama, a projection surface has to be chosen. While a sphere can hold a full panorama, it cannot be simply shown in full on typical screens. Often a cylindrical surface is chosen, as it can be unrolled to a rectangular image. While a cylindrical surface is a good choice for a full horizontal or vertical panorama, it cannot show a full horizontal and vertical panorama and will heavily distort towards the poles. Fortunately the field of cartography provides many projection types with various properties which allow the projection of spherical to flat coordinates [Sny93] and thereby provides us with a range of options to choose from. With a projection surface chosen, the images still have to be blended which might simply be done by feathering:

$$C(\boldsymbol{p}) = \sum_k w_k(\boldsymbol{p})\tilde{\boldsymbol{I}}_k(\boldsymbol{p}) / \sum_k w_k(\boldsymbol{p}), \tag{2.14}$$

Each point p in the blended image C is the result of summing up the image information at warped point $\tilde{\boldsymbol{I}}_k(\boldsymbol{p})$ of all images k. If point p is not part of an image k, w_k will evaluate to 0, otherwise to 1. In most cases, simple averaging will produce non-satisfying results, and more sophisticated methods can be used. Some examples would be Laplacian pyramid blending, gradient domain blending, exposure compensation or Poisson blending [SUS11].



(a)



(b)

Figure 2.3: Result and source images of an image stitching operation. (a) shows an image stitched from three separate images which are shown in (b).

CHAPTER 3

Hardware Components

The aim of this thesis is to generate a panoramic view, which requires hardware to take images as well as a display to output these images. This chapter introduces the types of cameras, lenses, displays and other hardware for which our pipeline is intended. The setup assumed for the rest of the thesis is a camera rig consisting of four cameras (see Figure 3.1), two of which are facing in the same direction, set apart at roughly eye distance. The other two cameras are pointing to the side in order to provide panoramic vision. Each pair of adjacent cameras has some overlap in their field of view which is used for camera registration. The pipeline is not limited to four cameras, more could be included when creating the panorama.

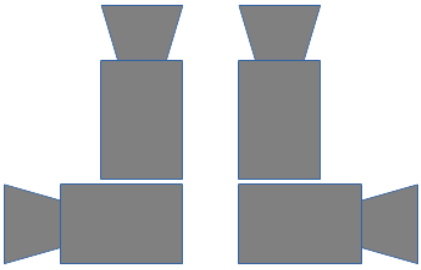


Figure 3.1: A top down illustration of an exemplary camera setup consisting of four cameras, two of which are facing the same direction.

The most common type of lens, as found on many consumer grade cameras or mobile phones, produces a field of view between 40° and 50°. Using such a lens, many images would be needed to create a full panorama. In order to reduce the amount of cameras or images, lenses with higher field of view can be used. Wide angle lenses are able to

capture a higher field of view but tend to be expensive due to their often complex lens design. A subset of camera lenses, the catadioptric systems, combines mirrors and lenses to produce varying effects. Systems where a mirror ball or a parabolic mirror reflects light into a camera often capture a field of view around 180°. These systems are not very common and more expensive. By far the most common type of lens currently used for high field of view photography are fisheye-lenses. They can be found on robots or security cameras and are available for camcorders or regular consumer grade cameras.

3.1 Fisheye-Lenses

The field of view produced by fisheye-lenses (see Figure 3.2 (a)) ranges from around 100° to 180° with theoretical designs reaching up to 310°, [Mar04]. This type of lens is popular when a high field of view is required due to their availability, low cost and ease of use. A full panorama can be recorded with as little as two cameras carrying fisheye-lenses, although most applications use more than two, as distortion and price increases with higher field of view. It must also be noted that the pixel density will decrease substantially with very wide angle lenses. Figure 3.2 (b) shows an image recorded through a fisheye-lens without any remapping. Normal or wide angle lenses produce images which can be viewed without further manipulation, while high field of view fisheye-images usually first have to be remapped.

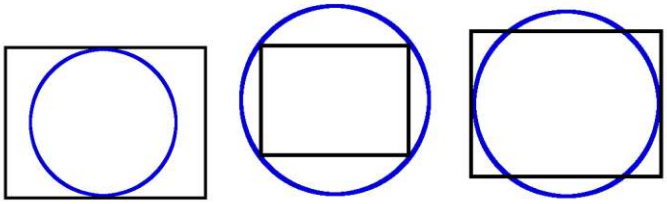
(a) (b)

Figure 3.2: (a) Fisheye-lens mounted to a camera[1]. (b) Image taken through a fisheye-lens of Oude Kerk in Amsterdam[2].

[1]Image by Morio. https://commons.wikimedia.org/wiki/File:Fisheye-Nikkor_Auto_6mm_f2.8_lens_2015_Nikon_Museum.jpg, https://creativecommons.org/licenses/by-sa/4.0/deed.en

[2]Image by Daniel Teoli. https://commons.wikimedia.org/wiki/File:Circular_fisheye_view_of_Oude_Kerk_Amsterdam_Daniel_D._Teoli_Jr..jpg, https://creativecommons.org/licenses/by-sa/4.0/deed.en

Due to the fact that most camera sensors use a 4:3 ratio, a fisheye-lens can be mounted in three different ways as demonstrated in Figure 3.3. Fitting the entire circle inside the sensor as seen in Figure 3.2, which covers the entire field of view but leaves many unused pixels, is called circular. Full frame uses the entire sensor, but reduces the field of view substantially at the vertical and horizontal limits. Finally, cropped circle is the compromise and only intersects the image at either the horizontal or vertical border of the sensor. What is important to consider is the pixel density, that is the ratio of available sensor pixels per degree of view. The combination of high field of view and unused pixels can reduce the pixel density considerably.



(a) (b) (c)

Figure 3.3: Three different approaches to mounting a fisheye-lens to a camera sensor. The blue circle is the image produced by the fisheye-lens, while the black rectangle is the camera sensor the image is projected onto. (a) Circular mounting wastes a large amount of sensor area. (b) Full frame mounting uses the entire sensor but cuts off part of the projected image. (c) Cropped circle mounting is a tradeoff between (a) and (b).

Although they may look similar, commercial fisheye-lenses come in various forms which produce different projection types. The common rectilinear photographic lens without distortion is described by the following projection formula:

$$r = f tan(\theta) \tag{3.1}$$

r denotes the distance from the center to the edge (radius) in the image plane measured from the optical center, f the focal length and θ the angle of incidence. Figure 3.4 shows this relation by projecting an object point onto the image plane.

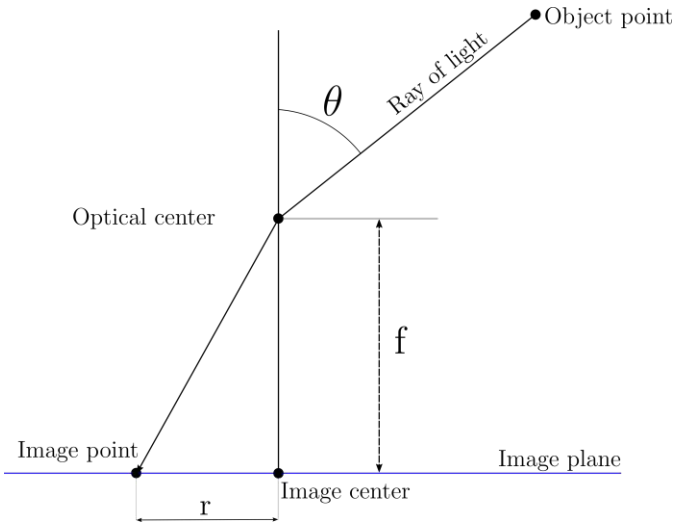


Figure 3.4: An object point is projected from world space to the image plane.

The relation between θ and r depends on the type of fisheye-lens. According to [Miy64], the two most common kinds of lenses are the equidistant and equi-solid type, defined through the projection formulas:

$$r = f\theta \tag{3.2}$$
$$r = 2fsin(\theta/2) \tag{3.3}$$

The equidistant projection from eq. 3.2 produces an equal scale for angular distances, while in equi-solid angle projection from eq. 3.3 imaged areas stay identical in size, regardless of their position in the image. Two of the lesser available types of fisheye-lenses are the orthographic type in eq. 3.4 and the stereoscopic type in eq. 3.5:

$$r = fsin(\theta) \tag{3.4}$$
$$r = 2ftan(\theta/2) \tag{3.5}$$

The former is used due to its property of maintaining planar illuminance, while the latter preserves angles between curves. In practice, lenses never exactly match their formula [Bet05] and are usually not perfectly manufactured, which adds some distortion to the projection function. This is made worse by imperfect mounting of the lens to the camera,

which further impacts the image quality. By using a model which accounts for and corrects these errors, image quality can be increased.

3.2 Cameras

Instead of consumer type cameras, industry video cameras are assumed to be used for the pipeline. Figure 3.5 shows a typical industry video camera without a lens. Compared to common consumer cameras, they lack control elements or internal storage and directly output images via a connector.

Figure 3.5: A typical industry camera without a lens mounted to it. Visible on the back are a USB connector for image data output and a Hirose connector used for synchronized image taking. [1]

Important factors when choosing a camera are resolution, aspect, frames per second and price. As can be expected, there is a direct correlation between price, resolution and frames per second. A high resolution camera will often have fewer frames per second and a higher cost. At the time of writing, a "DFK 38UX304" camera produced by "The Imaging Source" boasting a resolution of 4096 x 3000 could produce RGB24 images at a rate of 10 per second, while at 640 x 480 152 frames are available per second. For viewing a live feed, a higher frame rate is better, as it will produce a more fluid video. An important consideration when chosing a camera-lens combination is the pixel density. For example, a 40° lens with a quadratic 4 megapixel sensor has a pixel density of 50 pixels per degree. Mounting a 180° fisheye-lens on the same camera will reduce the average pixel density to just 11 pixels per degree. Depending on the lens type, the density may vary heavily towards the rim of the lens, further degrading the density in certain regions.

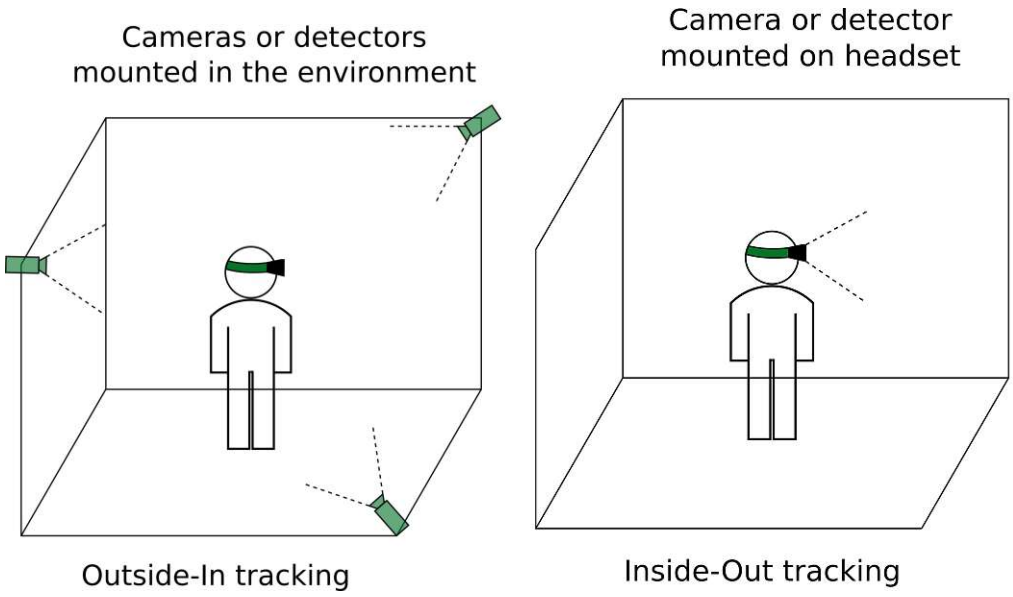
[1] Image by Theimagingsource https://s1.www.theimagingsource.com/application-1.6115.90614/img/hero/pph/pph38usb31.png, https://www.theimagingsource.de/produkte/industriekameras/usb-3.1-farbe/dfk38ux304/

Among some propietary connection options, many cameras can be attached via USB or RJ45. A resolution of 2048 x 2048 at 30 frames per second with 8 bit per pixel and color channel, a camera will occupy 120 mebibyte per second on the bus it is attached to. When producing a panorama, the images should be taken simultaneously. Moving objects will otherwise appear at different positions. If images are blended from unsynchronized recordings, the result may contain ghosting, where the same object appears multiple times at different locations. When stitching a panorama, heavy distortions might be present where the object moved. Many industry cameras therefore have an additional connector which allows for sending a trigger signal simultaneously to all attached systems. Another source for temporal distortions is the electronic shutter of the camera, which is either a rolling or a global shutter. The global shutter, which is to be preferred in such a setup, simultaneously records the incoming light at every pixel of the sensor. In contrary, a rolling shutter records line after line with a short delay in between. As a result, an object which moves at sufficient speed where the lines are currently being recorded may appear elongated, compressed or torn.

3.3 Virtual Reality Headset

The panoramas produced under this thesis are to be viewed through a virtual reality headset (see Figure 3.6 for an example of a typical HMD), which is a specific type of head mounted display (HMD). This is primarily expressed by the choice of camera resolution and pixel density. The main advantages of viewing a panorama through a HMD are the simple interaction and the strong feeling of remote presence. The recent push by the industry to establish Virtual Reality as a consumer product resulted in a new palette of consumer grade head mounted displays. The key feature of these devices is their ability to act as a window into a virtual place. To do this, the location and rotation of the headset is tracked and the image updated accordingly. Most head mounted displays will combine the data from multiple sensors to derive the spatial parameters. Gyrometers and accelerometers can be used for small and quick motions, but need compensation for drift (small measurement errors accrued over a long period of time). Inside-out tracking is the most commonly used solution, which records the surroundings with cameras mounted on the headset. Distinct features in the HMD's environment are then tracked over time to find the headset's position and rotation. Outside-in tracking has cameras positioned in the environment, locating the headset as demonstrated in Figure 3.7. The main advantage of inside-out tracking is that no additional hardware except for the HMD itself is needed. This is particularly useful in cases where no additional hardware can be installed in advance.

Figure 3.6: Oculus rift headset, which is a typical example for most virtual reality headsets in terms of size and headmounting straps.



(a) (b)

Figure 3.7: Two different approaches to tracking. (a) Outside-in tracking uses cameras or sensors with fixed positions on the outside which are detected by the headset. (b) Inside-out tracking has cameras or other sensors built into the headset, which track the movement of the outside relative to the headset. [2]

Most devices are fitted with one or two displays mounted close to the eyes of the user. Some form of lens or a combination of lenses is used to increase the perceived field of view.

[2] Image inspired by `https://delight-vr.com/xr-glossary/`.

Horizontally, the per eye resolution of most devices is around 1500 pixels with an aspect ratio close to 1:1. The Oculus rift shown in Figure 3.6 provides 1280 horizontal and 1440 vertical resolution. The diagonal field of view usually ranges between 100°and 130°. This results in an angular pixel density of 14, that means that 14 pixels are available per degree of vision. Table3.1 lists typical real world screens and their angular pixel density at their viewing distances. The ratio to HMD demonstrates the factor by which these devices have more pixels available than an average HMD.

Table 3.1: Pixel densities of various common devices. The aspect ratio for all devices is 16:9, the numbers are rounded and the device data are estimations based on currently available devices. The HMD's horizontal resolution is assumed to be 1280 at 90°FOV. Pixel density refers to the pixels available per degree of view at the given viewing distance.

Device	HMD	Cell Phone	Computer	TV	Cinema
Diagonal Device Size	8cm	16cm	60cm	120cm	15m
Horizontal Device Resolution	1280	1920	1920	1920	2048
Viewing Distance	3cm	30cm	50cm	3m	20m
Screen FOV at Distance	90°	13°	27°	10°	20°
Pixel Density	14	148	71	192	102
Ratio to HMD	1	10	5	14	7

As Table 3.1 demonstrates, the resolution of HMDs is still relatively low compared to other devices. The issue is most apparent when displaying details at range or trying to read text. In order to not further reduce the visual quality, the pixel density of the camera lens combination should then be at least equivalent to the HMD which is to be used. Another important factor when designing a system around HMDs is the cybersickness these systems may induce in their users. This term describes a combination of many factors which mostly result in a feeling of discomfort but might also express itself as nausea, headaches and more. It heavily varies between users and may render HMDs unusable for some. Some important factors to reduce cybersickness are a high frame rate between 60 and 90 frames per second, a high visual fidelity and stability of the tracking, which results in a more stable image inside the display.

CHAPTER 4

Pipeline Implementation

The goal is to produce a panorama from which a static panorama generation template can be produced. This panorama generation template is then to be used to stitch a panorama from multiple cameras in order to view a remote place in real time. Without a panorama generation template, stitching an image in real time is hardly possible due to the time-consuming algorithms necessary for such an operation. This chapter gives an overview of the pipeline developed for this thesis and then describes it in detail. This includes the camera class, which acts as a container for all necessary information required to generate a panorama. The intrinsic camera calibration and the viewer, which displays the panoramas and the images it is combined of. This is followed by a section about registration and stitching, which improve the panorama, before a description of the panorama generation template concludes the chapter. Most parts of the pipeline are accessible through a graphical user interface (GUI), which is introduced in Chapter 5.

4.1 Pipeline Overview

The first step of the pipeline for panorama generation is to intrinsically calibrate each camera, then their relative positions to each other are estimated. The images are reprojected to a panoramic surface by the transform model, where they are blended into a single panorama. The result can be exported to a static panorama generation template, which assigns each source pixel in each camera image a target position in the panorama and an alpha value. The pipeline is shown in Figure 4.1.

Figure 4.1: A high level overview of the panorama generation pipeline.

Figure 4.2 gives a more detailed look at the pipeline, also including calibration steps and the required models.

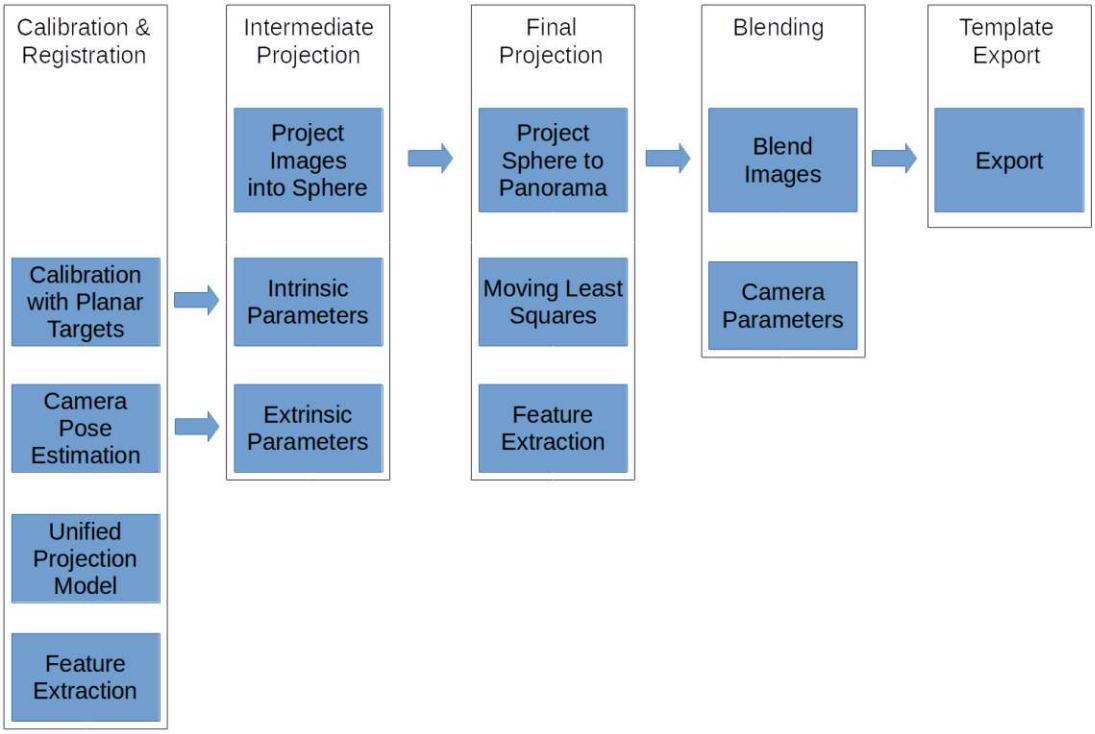


Figure 4.2: The full pipeline including the necessary information and processes required to advance between steps.

The calibration and registration steps are based on feature extraction and the unified projection model by Mei [MR07]. The intermediate projection then reprojects the images onto a sphere based on the parameters found. While projecting the images onto a panorama, a moving least squares approach is used to correct minor parallax errors. The images are then blended into a single panorama and finally the result may be exported to a panorama generation template.

4.2 Languages, Frameworks and Libraries

4.2.1 Languages

C++ was used as the main language to realize the pipeline with the exception of the viewer, for which a combination of C++, to communicate with the OpenGL API, and GLSL (OpenGL Shading Language), to program the shader stages, were used.

4.2.2 Qt

As a framework for desktop applications, Qt provides various platform-independent implementations of containers and a wide array of other functionalities. Of these functionalities, the ability to write and load files, store and load settings as well as the multithreading libraries were used.

4.2.3 OpenGL Widget

The Qt framework provides an OpenGL widget, which can be placed anywhere inside the GUI. It provides a surface which can be drawn to by OpenGL while providing helpful abstractions and wrappers. It was used when implementing the viewers.

4.2.4 Ceres-Solver

The ceres-solver is a C++ library for modelling and solving large optimization problems. It is used for the constructing and solving of the system during the registration step. The library automatically derives the functions to be optimized, provides various implementations of loss functions and a choice of solvers.

4.2.5 OpenCV

OpenCV is a free library with a focus on computer vision algorithms. The API is available in C, C++, Python and Java. It provides the implementation of many common computer vision algorithms and containers for typical data formats like images or vectors. The C++ API of the OpenCV library was used for the calibration of the cameras, for the extraction and matching of feature points and when generating the panorama generation template.

4.2.6 OpenGL

OpenGL is a specification and API for platform independent programming of GPUs. The main advantage gained by using the GPU instead of the CPU is the execution speed of parallelizable code. While a typical CPU has between 2 and 8 cores, a GPU can have multiple thousand cores which simultaneously execute threads. Only a small subset of the OpenGL pipeline stages are used for this thesis, namely the vertex stage, geometry stage and fragment stage. Figure 4.3 shows a condensed overview of the OpenGL pipeline in which the used shader stages are marked green. Each of these stages can be programmed by a shader program written in OpenGL Shading Language (GLSL).

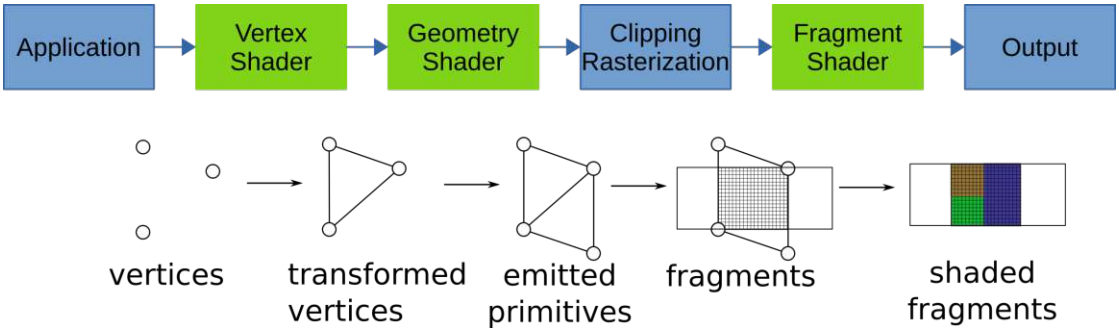


Figure 4.3: A simplified depiction of the OpenGL pipeline. The application sends vertices to the pipeline, which are then transformed in the various stages.

The application passes a set of vertices to the vertex stage of the pipeline which can transform them and then pass them further down in the form of primitives. There are many types of primitives, however only points and triangle-strips were used. In the geometry stage, extra primitives may be created based on a shader program. In the clipping stage, any primitive outside of the window to be rendered is clipped and the results rasterized based on the resolution of the surface which is to be rendered. Finally, for each fragment generated this way, a shader program is executed in the fragment stage.

4.3 Transform Model & Camera Class

Our transform model transforms between four distinct spaces, as illustrated in Figure 4.4: image space, which is the two-dimensional space of the image directly recorded by each camera, camera space, the three-dimensional space which is oriented along each camera's forward and up directions, world space, another three-dimensional space which in our case is oriented along the rig's forward and up directions and finally, panorama space, which is the two-dimensional space in the equirectangular panorama. If a relationship can be established between each of the spaces, image data can be transformed from image space to panorama space and vice versa.

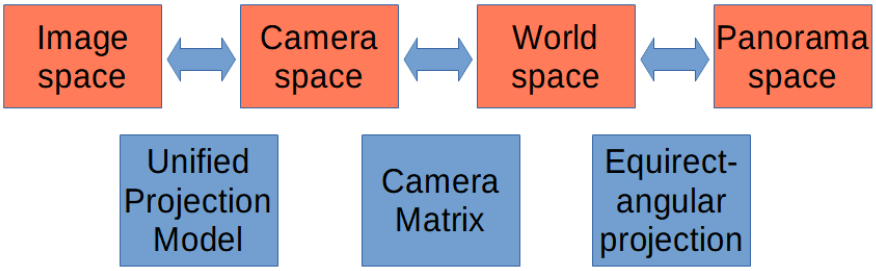


Figure 4.4: The upper row shows the spaces the transform model transitions between, while the lower row contains the models used to realize the corresponding transformations.

Three models are used to establish these relationships. To transform from image to camera space and back, the unified camera model is used. This model was primarily designed for catadioptric cameras, but also works for a wide range of other lens types including fisheye-lenses. The widely used camera matrix can then be used to transform between camera and world space, while equirectangular projection can transform between world and panorama space.
Each of the aforementioned models depends on a set of parameters which define the transformations. The equirectangular projection depends on the type of sphere which it projects from. The camera matrix is defined by its position in relation to world space, namely the extrinsic parameters. Finally, the unified projection model contains the intrinsic parameters, which describe how the camera lens projects lightrays.

4.3.1 Camera Class

The camera class includes all data needed in order to project an image into the panorama. It includes the parameters for the aforementioned transform model and is extended by image and solving parameters:

- Intrinsics parameters
- Extrinsic parameters
- Image to be used in the viewer
- Blending parameters
- Solver parameters

The rest of the processing pipeline then serves to find proper values for these parameters. The class also provides an event which informs subscribers of changes to the camera. This provides a robust way to propagate changes to the camera through the program.

4.3.2 Intrinsic Camera Model

To model the camera parameters, the unified camera model by Mei [MR07] is used. Two parameters are used to define the radial distortion, k_1 and k_2, and another two for the

tangential distortion, k_3 and k_4. One parameter defines the mirror parameter ξ. Five more parameters, which describe the image center cx, cy, the focal length fx, fy and the skew, are used for the pinhole model. In addition to Mei's model, a circular radius parameter is added, which describes the size of the circular image in relation to the aspect of the image. Whenever the image is reprojected, only pixels inside this radius are used.

4.3.3 Extrinsic Camera Model

The extrinsics are given by a forward and an up-vector which are normal to each other and define the rotation of the camera. The vectors are in relation to a base coordinate system defined by the rig.

4.3.4 Image and Blending Parameters

The image is stored as a path on the local file-system and is loaded when needed by the GUI. Blending between multiple cameras in the panorama is controlled by a simple value which defines over what distance the images will be blended into each other. Each camera is defined as either a stereo-left, stereo-right or side-camera, which controls how the image will be blended when generating the panorama.

4.3.5 Solver Parameters

The solver parameters influence how the cameras are handled while optimizing the camera parameters. Cameras can be set to reproject to a cubemap first, which can improve the detection and matching of feature points for very high field of view cameras. The solver can also be instructed to sectionize the image, which splits the image in a gridlike fashion before matching. For stereo-cameras, a disparity factor can be set, which adds an additional horizontal offset to control the intensity of the stereo effect.

4.4 Intrinsic Camera Calibration

The intrinsic calibration uses the Omnidirectional Camera Calibration module for OpenCV [Bra00], which is an implementation of Mei's approach [MR07], using planar targets containing circles in a grid layout of known dimensions. Figure 4.5 shows an example of such a grid. In order to calibrate the camera, such a grid is first printed to a piece of paper (or any flat surface), then some pictures of the grid are taken with the camera to be calibrated. Based on the positions of the circles in the images, the camera parameters can be found. To find the circles in the images, usually a blob detection algorithm is used.

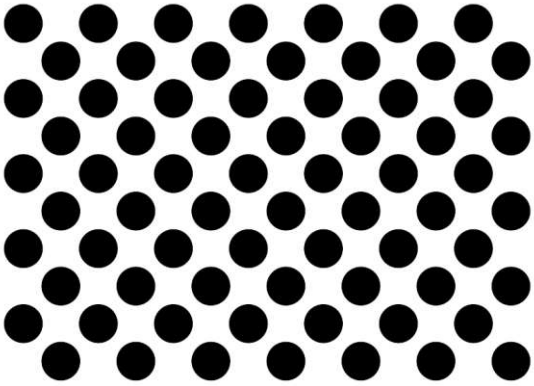


Figure 4.5: A grid of circles with regular intervals often used in camera calibration.

4.4.1 Blob Detection

The circles of the grid are detected by a blob detector, which first converts the image into multiple binary images by applying a preconfigured threshold and then combines pixels into blobs within a threshold range. The minimum and maximum sizes of blobs are configurable and depend on the source image resolution. For images of four megapixels, a range from 50 to 5000 pixels appeared to be a good choice. Blobs are then checked for circularity and discarded if not within a configurable range. As circles will be elongated if observed through a fisheye-lens and close to its rim, the algorithm accounts for elongated circles. The centers of all blobs detected this way are then passed on to the grid detection.

4.4.2 Parameter Initialization and Optimization

Next, we try to find corners which contain the calibration grid as defined previously. If the search is not successful, the image is discarded and not used for calibration. Based on the grids, the parameters of the uniform projection model are initialized according to Mei [MR07]. Once all the images have been processed, a Levenberg-Marquardt optimization algorithm [Mor78] is used to find and optimize the intrinsic parameters (see 4.3.2).

Multithreading

Loading the images presented itself as a time-consuming operation. Loading many images and then running the calibration algorithm on them could often take a minute or more. Repeating this process for each camera was a frustrating process. In order to speed up the loading of images and grid detection, multithreading was used. In most situations, this would reduce the time by a factor of the cores available on the system (typically 8 in our experiments).

4.5 Viewer

As previously mentioned, the viewer depends on OpenGL in order to generate the images and only a small subset of the OpenGL functionality is used. The structure and usage of OpenGL are briefly introduced here, as they will be referenced again later in the thesis.

4.5.1 OpenGL Pipeline

The viewer and its modes are implemented in OpenGL to make use of the parallel processing power of graphics processing units. This means that all parameters and the raw images are passed to the graphics card, which will then calculate the result. In all viewer modes, there are at most three stages of the OpenGL pipeline in use, as described below.

Vertex Stage

The vertex stage is only used to generate a quad which covers the entire screen to be rendered. As OpenGL will clip anything outside of the range from -1 to 1 along both the x and y axis before the fragment stage, these are also the coordinates of the quad's corners. Each vertex can be accompanied by additional data, which will be interpolated in the fragment stage. The vertices representing the corners are then passed down the pipeline.

Geometry Stage

This stage can be used to generate extra geometry, called primitives in OpenGL. This stage is only used for drawing helpers in the GUI and is included for completeness.

Fragment Stage

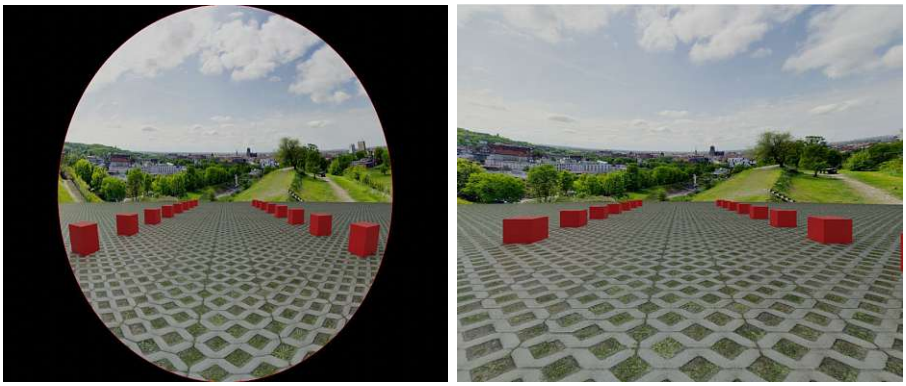
The fragment shader is executed for every fragment covered by geometry passed down from the previous stages. In this thesis, this is mostly a screen covering quad. As a result, the shader code is executed once for every fragment, or pixel, of the entire screen. When generating the panorama, the screen quads vertices handed down from the vertex stage are accompanied by their panorama coordinates. The panorama coordinates range from 0 to 2π horizontal and 0 to π in the vertical. As a result, the upper left corner then has the OpenGL coordinates of $(-1, 1)$ with the panorama coordinates $(0, 0)$ in contrast to the lower right corner with $(1, -1)$ and $(2\pi, \pi)$ respectively. When rendering each fragment covered by the quad (in other words the entire screen), these values will be interpolated and then used to generate the sphere coordinates of the position in the panorama. Almost all calculation is done in this stage of the pipeline.

4.5.2 Viewer Modes

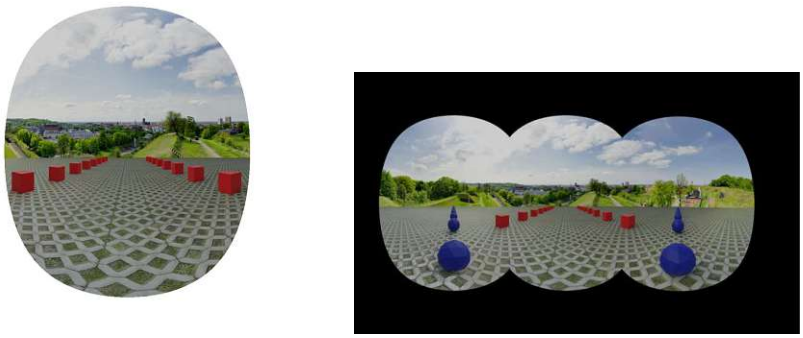
The viewer has four distinct modes, which are shown in Figure 4.6:

1. Direct View
   - Shows the image unaltered
   - Displays the circular radius
2. Rectified View
   - Reprojects the image to a perspective view
   - Reduces the image to 90° FOV
3. Equirectangular View
   - Reprojects the image to an equirectangular map
4. Panoramic View
   - Shows the entire panorama

Let us assume for now that the cameras are already properly aligned. The camera registration algorithm will be explained in more detail in a later section.



(a) Direct View (b) Rectified View



(c) Equirect View (d) Panoramic View

Figure 4.6: The modes of the viewer. (a) Direct view directly shows the image without any modification. (b) Rectified view produces a perspective 90° field of view image. (c) Equirectangular view projects a single image onto an equirectangular surface applying distortion correction. (d) Panoramic view combines all cameras currently active.

4.5.3 Direct View

The direct view simply shows the image unaltered. First, a screen filling quad is generated in the vertex shader and passed on to the fragment shader where the fragments are textured with the image loaded for the currently selected camera. Additionally this mode shows the radius of the circular image, which is rendered as a thin red ring.

4.5.4 Equirectangular View

The equirectangular view will display an undistorted image projected onto an equirectangular surface.

Screen Quad

A screen quad is generated and passed to the fragment shader in order to produce a fragment for each pixel covered by the quad. As the resulting surface covers the full panorama, the coordinates range from $(0, 0)$ to $(2\pi, \pi)$. Note that the coordinate range simply mirrors longitude and latitude. This means that a fragment is generated for every possible direction on a sphere, where the sample rate depends on the resolution of the surface. In the case of a 2000 by 1000 equirectangular surface, a total of 2000000 fragments will be produced. Each position is then transformed into its spherical position according to equirectangular projection as based on Snyder [Sny93]. Each position on a sphere with radius 1, centered on the origin, is a normalized direction vector.

Projection and Distortion Correction

The direction vectors are then projected into image space according to Mei's model [MR07] and the color value at the position is used to paint the fragment. This projection undistorts the image depending on the intrinsic camera parameters which were found during the calibration step. The forward vector of the camera is equivalent to the center of the equirectangular projection, thus no further rotation has to be applied in comparison to the panoramic view.

4.5.5 Rectified View

The rectified view displays an image with a 90°horizontal and vertical field of view. As in the equirectangular view, a screen filling quad is generated and passed on to the fragment stage. Here, the corners of the quad do not represent the panoramic surface but the side of a cube, which is why the coordinates here range from $(-1, -1)$ to $(1, 1)$. These coordinates are then expanded with a z coordinate of -1 to produce the side of a cube and normalized before proceeding with the projection according to the unified projection model as before. Note that on a unit sphere, this is equivalent to a 90°by 90° section.

4.5.6 Panoramic View

The panoramic view operates similarly to the equirectangular view, except there are now multiple cameras participating to generate a panoramic image. As before, a full screen quad is generated and passed to the fragment shader stage, in order to generate directions covering a sphere. The direction vector generated from the full screen quad is then modified according to the relative camera rotation to world space. This modified direction vector is then passed to the Unified Projection Model in order to undistort the camera's image.

4.5.7 Culling and Blending

As the image is to be displayed on a non-stereo display, only one stereo-camera is active when drawing the panorama. Each fragment may be "seen" by any number of cameras, which results in one of the following cases:

Fragment is covered by no cameras
The fragment is painted black.

Fragment is covered by two stereo cameras and any number of non-stereo cameras
The fragment is painted by the active stereo camera.

Fragment is covered by one stereo camera
The fragment is painted black, to avoid the visual discomfort if only one eye is able to see this part of the image.

Fragment is covered by one stereo camera and any number of non-stereo cameras
The fragment is blended between the non-stereo cameras.

Fragment is covered by any number of non-stereo cameras
The fragment is blended between the non-stereo cameras.

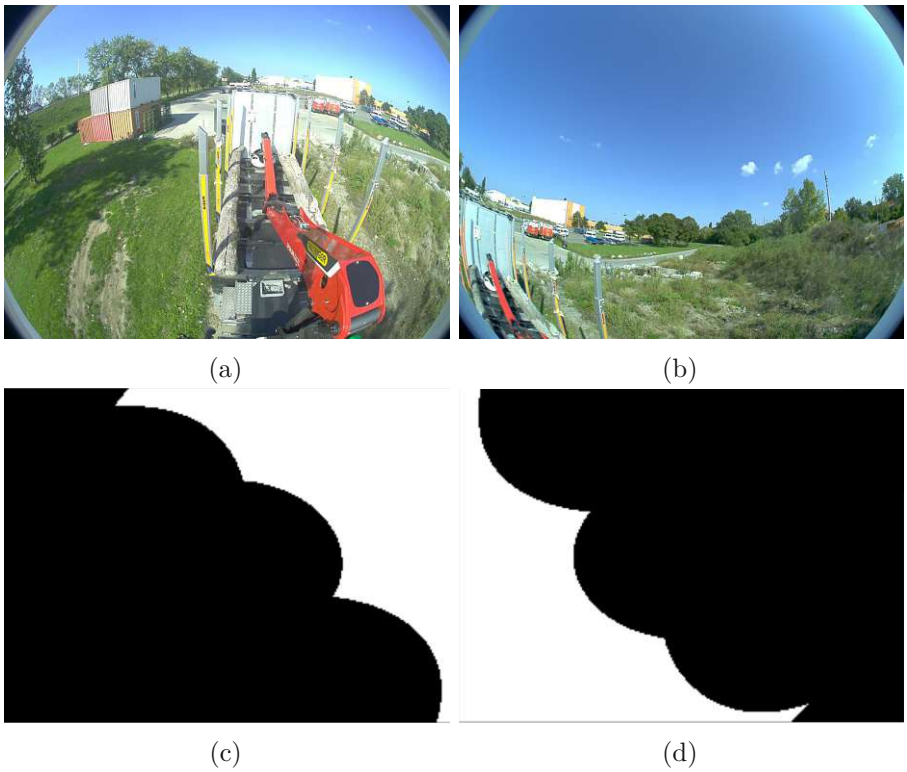
The images are then blended in a linear fashion over a distance which can be set individually for each camera.

4.6 Registration

After calibrating all cameras, the system can automatically adjust the camera orientations and parameters. Features are extracted and matched between cameras and a least squares method is used to further optimize camera parameters by minimizing a distance error.

4.6.1 Feature Point Extraction and Matching

The registration step depends on features matched between neighbouring cameras. ORB [RRKB11] features are extracted and matched with help of the OpenCV library. Unfortunately, these features are not optimized for circular images and heavily distorted features closer to the rim are often impossible to match. While features exist which were designed to work under these conditions, open source implementations are often not available. With calibrated cameras, the images may first be reprojected to a cubemap on which regular feature detectors and descriptors work as expected. As this is a costly step, it should be avoided if possible. In any case, the features are only extracted in the overlapping region in order to reduce the amount of bad matches. Two distorted images and their overlapping regions can be seen in Figure 4.7.



(a) (b)

(c) (d)

Figure 4.7: Two distorted images and their overlapping regions. (a) is the stereo camera image. (b) is the side-camera image. (c) and (d) are the masks for these images. Feature points will only be extracted from the white portions of the masks.

To calculate the overlapping region, the user first has to manually orient the cameras. This step could also be automated by applying a RANSAC algorithm. The image is then stepped through at a reduced resolution and each position is tested if it could be visible in the neighbouring camera. This includes checking if the position is inside the camera's

circular image radius, then projecting it into the neighbouring camera and again checking if it is then inside this camera's circular image radius. This process is described in Figure 4.8. If a pixel is inside the right camera's image by some margin, the position as well as its surroundings are marked as a region to collect features from.

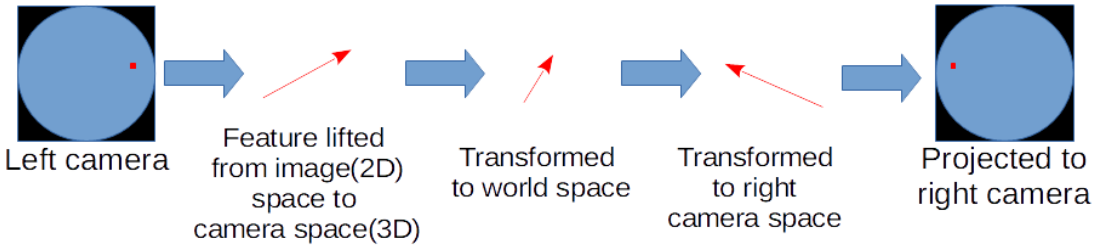


Figure 4.8: A point is transformed from the left camera to world space and then projected into the right camera.

After the overlap mask has been created, a configurable amount of features are extracted from both images. Employing a knn matching algorithm [Bra00], the two best matches for each feature are selected. To remove ambiguous matches, a ratio test is applied to each match. Only if the Hamming distance of the first match is sufficiently smaller than that of the second best match, it is assumed to be distinct enough, otherwise it is discarded.

An issue which often arises when extracting features from an image is that certain areas are very feature-rich while others are not. This may cause a bias towards the feature-rich area. To avoid this issue, the image is split into a grid and only a limited amount of features is used per cell. This process is repeated for each camera in the rig and all feature matches produced this way are collected and passed to the optimization stage.

4.6.2 Optimizing Camera Orientation and Parameters

The optimization is formulated as a least squares problem, which is to be solved by the ceres solver [SK]. Two approaches were implemented, the findings of which will be presented in the results section (see Section 6.3.1). In both cases, one camera must be designated as the root camera, which will disable the updating of its orientation. As a result, other cameras will be oriented along it while the root camera stays unmodified. In order to solve the optimization problem an error metric needs to be established. The two following methods, based on Global Registration [Sze06] were implemented and tested.

Ray Projection Error

The first error metric implemented is the ray projection error. First a ray (3D direction) is generated from a feature match. Any of the two cameras involved is selected and the

position of the feature is projected into camera space and then world space using the unified projection model. During the optimization steps, this vector is then projected into each of the cameras and the distance from the projection point to the feature position is the error metric. Both the camera orientations and the vector itself are allowed to vary during the optimization.

Panorama Projection Error

The second metric implemented measures the distance between the two projection positions of a feature match. For each camera, the matching feature is first projected from image to camera space and rotated into world space (see Figure 4.9). These positions are then projected to panorama space and the distance between the two is used as an error metric.

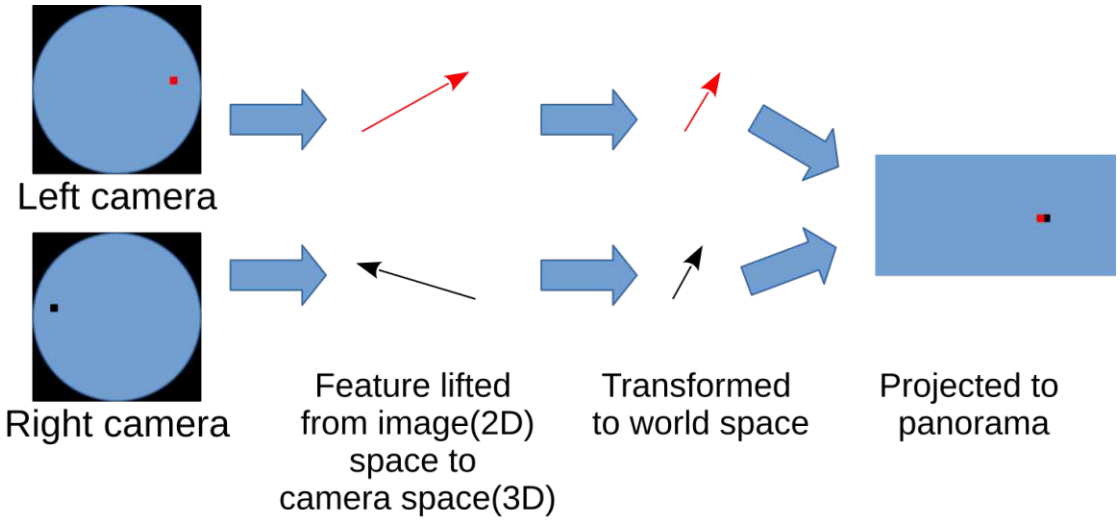


Figure 4.9: A feature match between two cameras is lifted to world space and then projected into the panorama.

Only if the cameras are perfectly calibrated and aligned, the positions match. The error can be expressed by the following formulation:

$$s = \sum \|f_e(f_{c1}(p_1) * R_{w1}) - f_e(f_{c2}(p_2) * R_{w2})\| \tag{4.1}$$

s denotes the error or distance between two matching feature points. The functions f_{c1} and f_{c2} are projection functions which lift the 2D positions of points p_1 and p_2, two points of identical real world position, from image space to a 3D vector in camera space. The subscript numbers denote to which camera the function or point is related to. Here

1 is the left camera and 2 is the right camera. A total of 10 parameters, the intrinsic calibration parameters, are required for this projection. The function is derived from Mei's model [MR07] and is a combination of the lens distortion and projection functions introduced in Chapter 2. R_{w1} as well as R_{w2} are the camera rotation matrices from eq. 2.2.1, which transform the 3D vectors from individual camera spaces to a common world space. Each of those transformations consists of 3 parameters and describes the rotations between the cameras. The function f_e is the equirectangular projection taken from Snyder [Sny93], or in other words, the transformation from world space to panorama space.

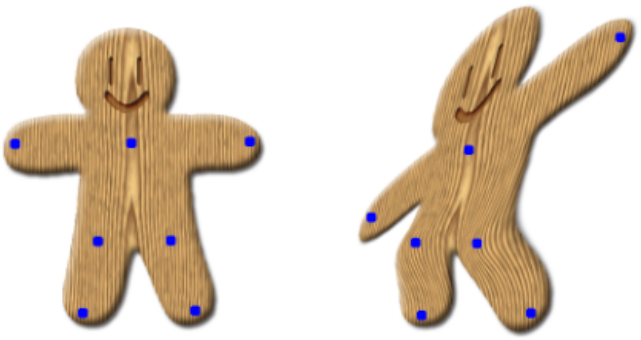
Additionally, a Cauchy loss function is added to deal with outliers and mismatches:

$$S = \sum_i log(1 + s_i) \tag{4.2}$$

The total error S is then the sum of all individual errors s_i modified by the Cauchy loss function. Each pair of matching points then produces one equation. Taking all the equations of matching points between all cameras results in a system of linear equations. Solving this system yields the parameters for the rotation matrix and camera distortion which produce an optimized panorama, as the error is minimized. This system is solved using the ceres solver [SK], which employs Levenberg-Marquardt optimization algorithm [Mor78].

4.6.3 Moving Least Squares

After the previous stage, it is very likely that feature point matches do not overlap perfectly. In order to further improve the visual quality of the panorama, the projected images are slightly warped in order to overlap the feature point matches in the final result. Each feature match represents two points in the panorama in very close proximity. The algorithm simply chooses one of the points and moves it so that both overlap in the panorama image. Then the image is deformed according to Schaefer [SMW06]. An example for such a deformation can be seen in Figure 4.10.



(a) (b)

Figure 4.10: An image with control points, before and after manipulation. (a) The original image with control points as blue squares. (b) Affine transformation of the original image after moving the control points. Image taken from Schaefer [SMW06]

4.7 Panorama Generation Template Export

Once satisfied with the panorama, it can be exported to binary map files. The result is a color map and an alpha map per camera as well as a text-file containing additional information (see Figure 4.11 for a schematic representation). The color map is of the same resolution as the panorama and each entry points to the source position in the camera image. When multiplied with the alpha value and combined with the other cameras, the panorama can be constructed. An additional file is generated for each camera, describing image sizes, the panorama size and a region of interest. This region is a rectangular subset of the panorama and contains the area which will be updated by the associated camera.

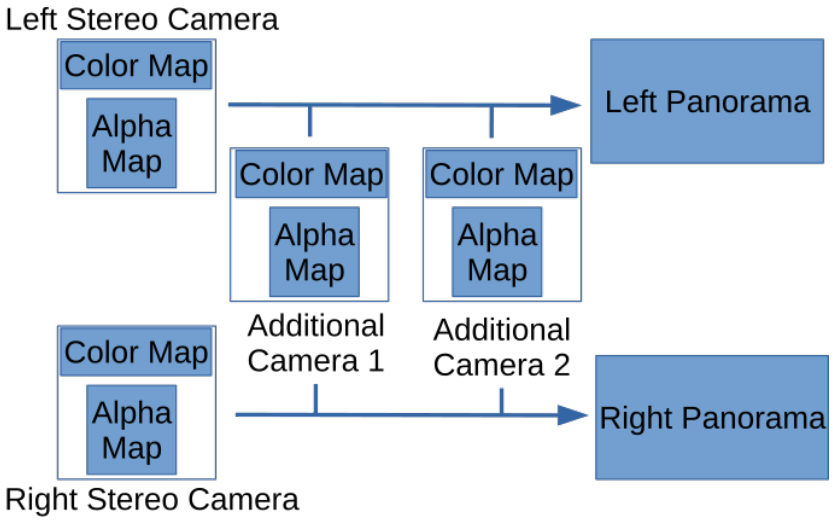


Figure 4.11: Reconstructing the panoramas for the left and right eye.

The export is functionally implemented in C++ and projects the images based on the transform model from Section 4.3. To speed up the process, a worker thread is started per available core on the host and the work is split into equally sized chunks of the panorama.

CHAPTER 5

GUI Implementation

This chapter introduces the GUI which was developed alongside the panorama reconstruction algorithm in order to provide an interactive way to generate, inspect and modify panoramas. The most prominent feature of the GUI is the viewer, which can display the panorama but also has multiple other modes of operation. The viewer is embedded in the main window, which also shows information about the currently selected camera, and allows the modification of other options. The camera calibration, registration and panorama generation template export are handled in additional windows in order to avoid overloading the main window. Before presenting the individual parts of the GUI, an overview of the workflow (Section 5.2), which reflects the pipeline, is given.

5.1 Language and Framework

The graphical user interface is based on the Qt framework [The21]. It can be used to build cross-platform applications, which run on most major desktop, mobile and embedded platforms. It is accompanied by a suite of tools which allow for the rapid layouting of graphical interfaces. The Qt framework also provides OpenGL surfaces, which were used in this work to implement the viewer. The functionality of the GUI was written in C++ with heavy reliance on the C++11 functionality.

5.1.1 Threading

Qt provides a main thread which updates and renders the interface. If any operations are executed from the main thread, they will block any updates to the GUI, which will in turn become unresponsive. Until the operations are finished, no updates can be sent to the GUI and, depending on the operating system, the window might be greyed out or the user might be asked if they want to terminate the program. In order to avoid these issues, all time-consuming operations are executed in a separate thread.

5.1.2 Qt Event System

Qt relies on an event system to propagate information in an asynchronous fashion. For example, clicking a button will insert an event into Qt's event queue, of which all registered listeners will then be informed and can act upon. Similarly, an event is triggered by most other interactions with the GUI or own events can be registered. The event system is also used for threading, where the communication between threads is handled by passing events.

5.2 Workflow

The workflow follows the same structure as the pipeline. The cameras are first added to the program, which defines the rig. Each camera is then calibrated by loading a set of images containing planar targets. To proceed with the registration step, images of a static scene must be loaded before the algorithm can be executed. The moving least squares approach can then be applied to further optimize the panorama. Once the result is satisfying, it can be exported to a panorama generation template, which allows the recreation of the panorama with different images.

In summary, the steps required to produce a panorama are:

1. Add cameras, configure rig
2. Intrinsic calibration of the cameras
3. Registration
4. Transformation
5. Export panorama generation template

The various elements of the GUI will be introduced below in the same order. First the main window where cameras are added and configured will be shown. This will be followed by the intrinsic calibration window, which acts as an interface to calibrate the cameras. Registration and panorama generation template export are both available through popups which can be accessed again from the main window.

5.3 Main Window

At startup, the main window of the GUI, shown in Figure 5.1, is presented to the user. It is split into a dropdown menu at the top and three sections in the body. In order to construct a new panorama, cameras have to be added through the camera list interface in the top left corner (red). Each camera then needs to be calibrated, by first clicking on its name and then selecting "calibrate camera", which will open a separate window to handle the process. Upon selecting a camera, the bottom left section (green) will be updated to show the selected cameras properties. This part of the GUI provides three

tabs, each of which shows different camera properties. The largest part of the main window is covered by the viewer (blue), which, depending on the currently set mode, can be selected through the dropdown menu situated to its lower left and will display either the panorama or some rendition of the image set for the currently selected camera.

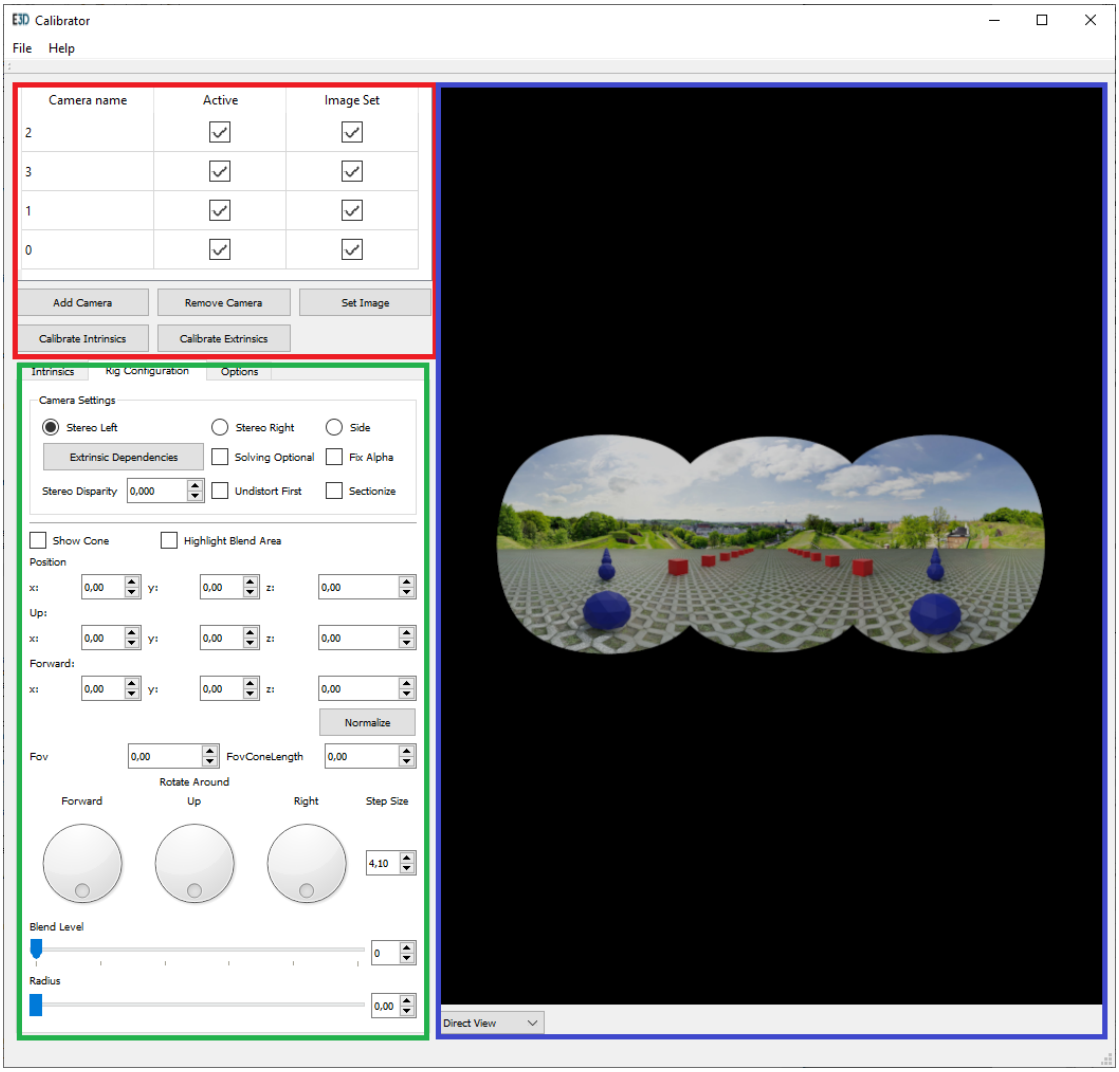


Figure 5.1: The main window of the GUI is split into three sections. The camera list (red), the camera properties (green) and the viewer (blue).

5.3.1 Camera List Section

The camera list, as seen in Figure 5.2, shows all cameras added to the current configuration. When creating a new panorama or adding a new camera to an existing one,

this part of the GUI allows adding, removing and calibrating cameras. The name for each camera can be set by clicking it, which will transform the list item into a textbox. The "Active" checkbox can be toggled by clicking it, which will activate or deactivate a camera in the panorama. The "Image Set" checkbox displays if the camera has an image assigned for use with the viewer.

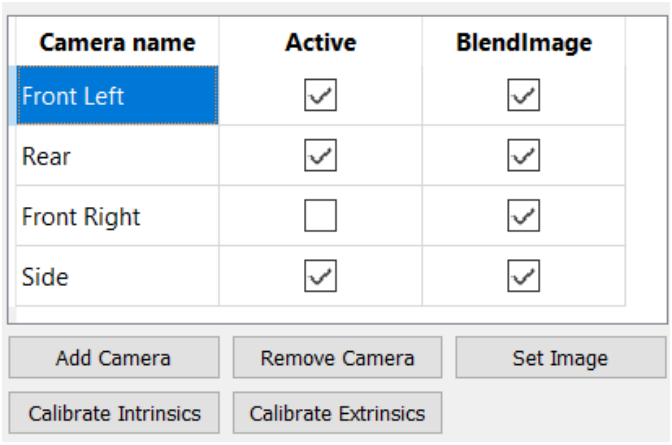


Figure 5.2: The camera list shows all cameras of the rig and allows the user to select a camera for manipulation.

5.3.2 Camera Context Section

The lower left section of the main window (green) changes its contents based on the currently selected camera. It contains three tabs, each providing a different set of properties.

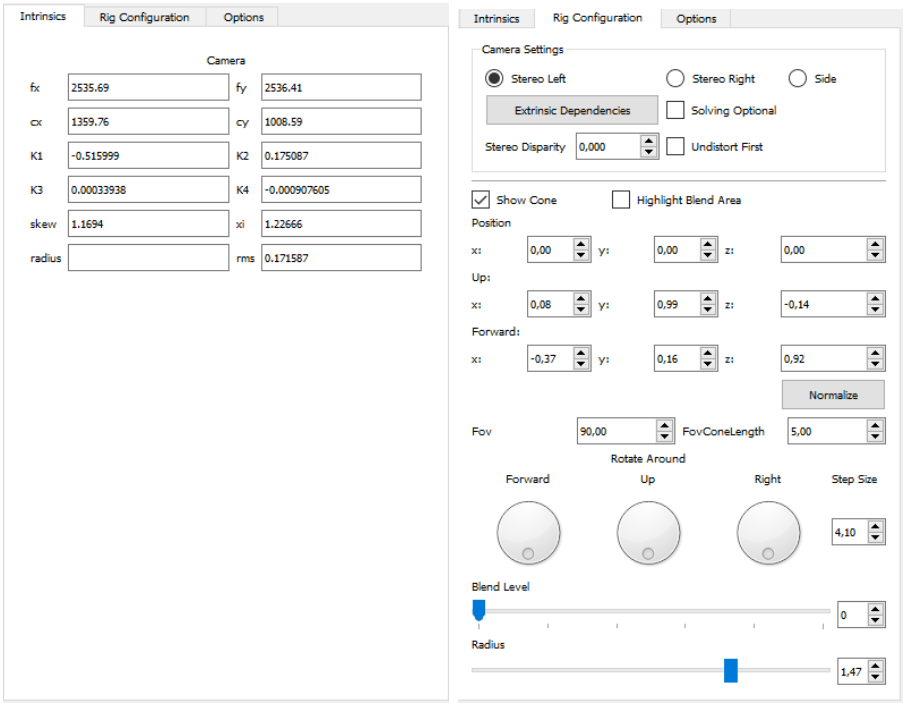
Intrinsic Properties

The first tab (see Figure 5.3 a)) shows the camera intrinsics, mostly for informational purposes, but it also allows for direct manipulation if needed. These parameters contain the focal length, image center, distortion correction parameters and, additionally, the mean reprojection error found during the calibration process. Modifying any of these parameters will update the camera and the image in the viewer is updated in response.

Camera Settings

The second tab (see Figure 5.3 b)) contains camera settings as well as blending, registration and positioning parameters. Each camera must be either a stereo or a side-camera, which influences how they contribute to the panorama. The "Extrinsic Dependencies" button makes it possible to to set the camera hierarchy. Checking "Undistort First" will reproject images from the circular images to cubemaps before feature point extraction

and matching. The orientation and position of the cameras can be set through either the numeric fields or the knobs. The "Blend Level" slider manipulates how the image recorded by the camera is blended into neighbouring cameras and the radius is used to set the size of the circular region in the fisheye-image.



(a) (b)

Figure 5.3: The camera configuration panels of the main GUI. (a) Intrinsic parameters.s (b) Extrinsic and other camera properties. The values of the currently selected camera can be changed by manipulating the corresponding fields.

Options

The last tab (see Figure 5.4) contains options to control the viewer. Most importantly, the moving least squares approach can be turned on or off and panorama images can be exported. Additionally, a raster can be enabled to help in aligning images, or points can be marked in an image, which are then afixed to the camera and move with it.

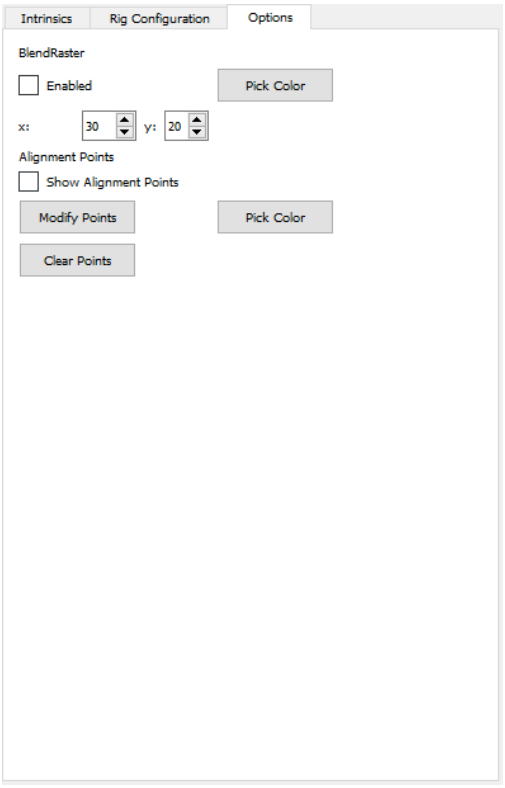


Figure 5.4: The options tab contains further settings for the viewer and allows the user to export images.

5.3.3 Viewer

The viewer (as shown in Figure 5.5) takes up the largest part of the GUI and provides multiple modes. The content of the viewer is generated on the system's GPU in order to speed up the process.

Direct Show

The default mode "Direct Show" will simply display the image set for the currently selected camera. It can be used to inspect the raw image and will also display the radius.

Equirectangular

The second mode displays the currently loaded image projected to an equirectangular surface. The unified projection model is used to reproject the image recorded by the camera based on the properties found by calibration. This mode will only show an image if the camera was calibrated before and allows for a quick visual inspection of the calibration quality.

Rectified View

In a similar fashion to the equirectangular mode of the viewer, the rectified mode will display the image projected to a 90°quadratic surface. In contrast to the panoramic projection, straight lines will not bend in this mode and bad calibrations should be easy to spot.

Panoramic View

The heart of the viewer is the panoramic view, which will reproject all cameras based on their properties onto an equirectangular surface. Some additional helpers can be activated in this mode. This includes, but is not limited to, the option to zoom, show an alignment grid, switch between left and right image of the stereoscopic panorama and average all images. If a panorama generation template is exported from the GUI, it will reconstruct the panorama if the same images are used.

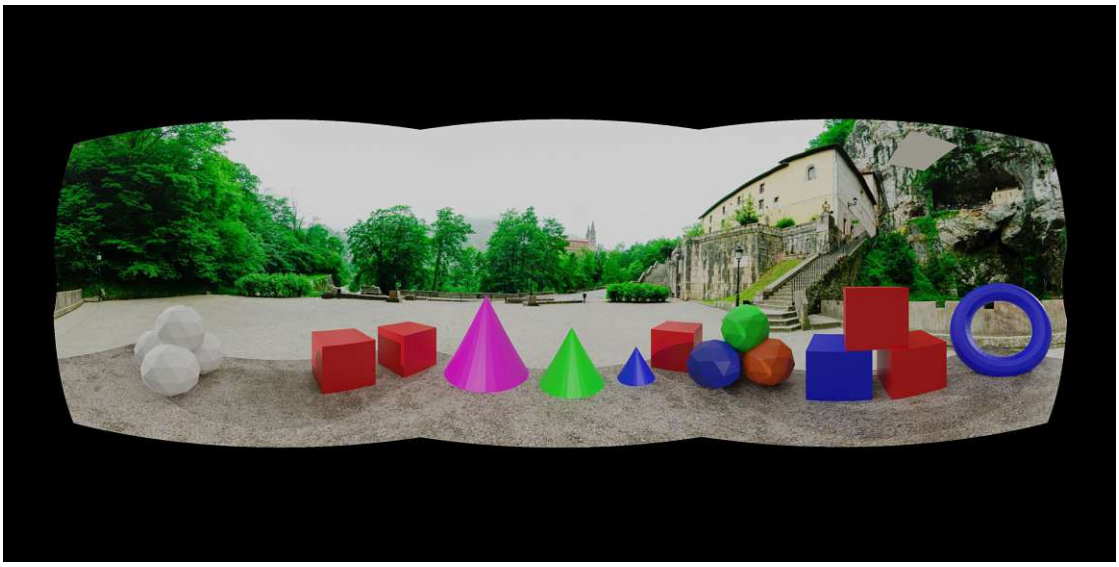


Figure 5.5: The viewer displaying a panorama.

5.3.4 Intrinsics Calibration Window

Camera calibration is handled in the intrisics calibration window, which can be accessed by clicking the "Calibrate Intrinsics" button after selecting a camera in the camera list. The window with some calibration images loaded can be seen in Figure 5.6. A number of images can be added by clicking the "Add Images" button, which will open a file selection dialogue. The window features a simple viewer, mirroring the direct show functionality of the main window, and is meant to provide a way of quickly inspecting the loaded images. The calibration process works on circle boards, the properties of which can be configured in the lower left section. In order to optimize the blob detector, which is used to find

the circles, the minimum and maximum size of the circular blobs must be defined. If the camera heavily distorts the image, the minimal circularity of the circles can be adjusted. The calibration process can then be started by pressing the "Calibrate Camera" button. A progress bar will show the current progress, while the text-output widget will print additional information and inform the user if the calibration was successful. If the calibration was successful, the camera will now have its intrinsic properties set and the window can be closed.

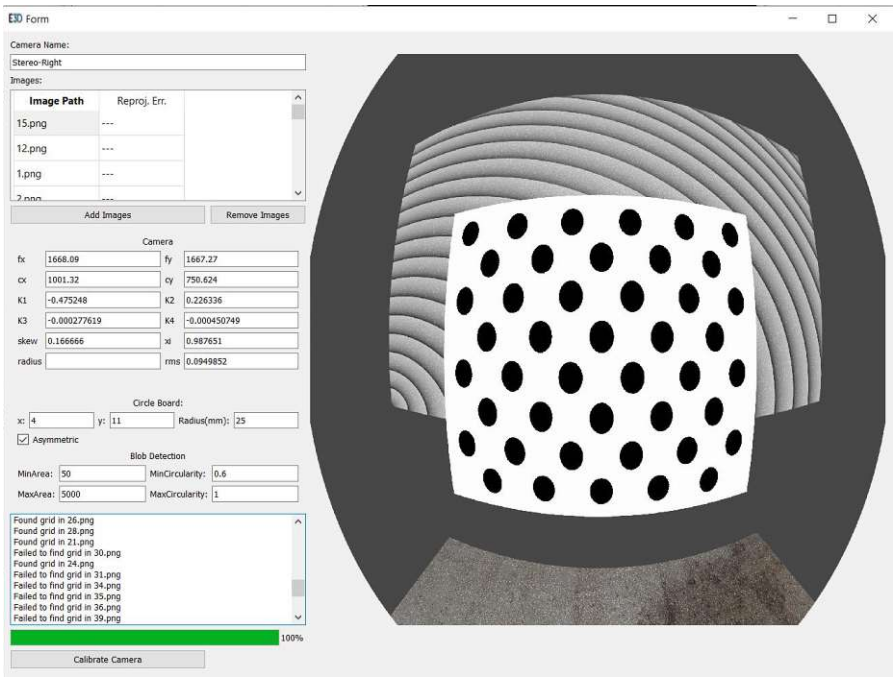


Figure 5.6: The intrinsic calibration window. The top left shows a list of the currently loaded calibration images. The viewer on the right can be used to inspect the loaded images.

5.3.5 Registration Pop-up

Clicking the "Registration" button will open a pop-up window (see Figure 5.7) where parameters can be adjusted before running the algorithm. Most importantly, the amount of extracted features can be set as well as how unique they must be when matching them. The registration process can then be started by pressing the button at the bottom right. Like in the intrinsic calibration window, a text-box and progress bar inform the user of the current progress. The GUI will be locked while the algorithm is processing and upon finishing, the panorama and all cameras will be updated immediately.

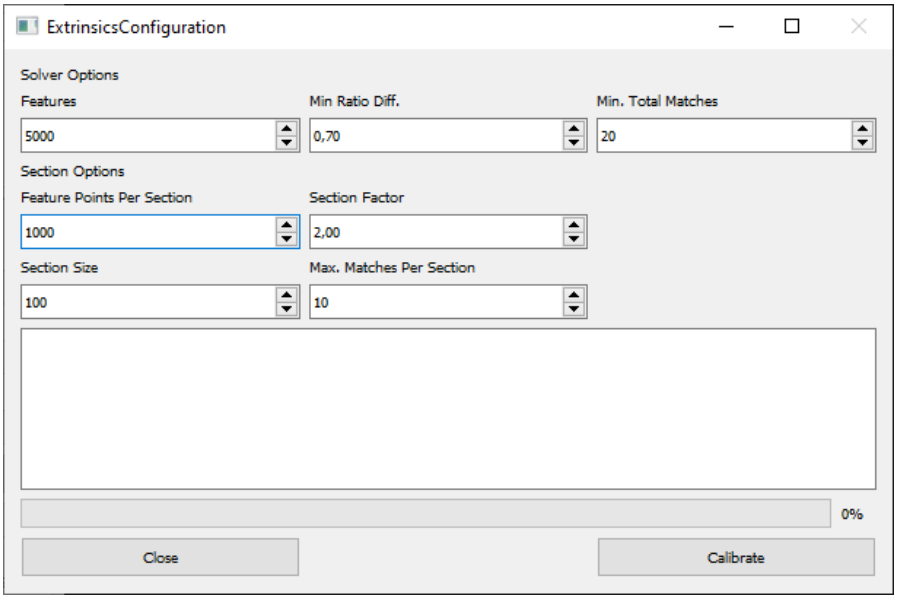


Figure 5.7: The registration pop-up allows the user to set various options for the registration step of the algorithm.

5.3.6 Panorama Generation Template Export Pop-up

The "Panorama Generation Template Export" pop-up (shown in Figure 5.8) can be accessed through the dropdown menu at the top of the GUI under "File", "Template Export". It allows the user to set the size of the equirectangular surface and the path the panorama generation template should be exported to. If the checkbox "Show Results" is active, both images of the stereo-panorama will be opened in order to enable the inspection of the results. As before, a progress-bar will show the status of the export-process.

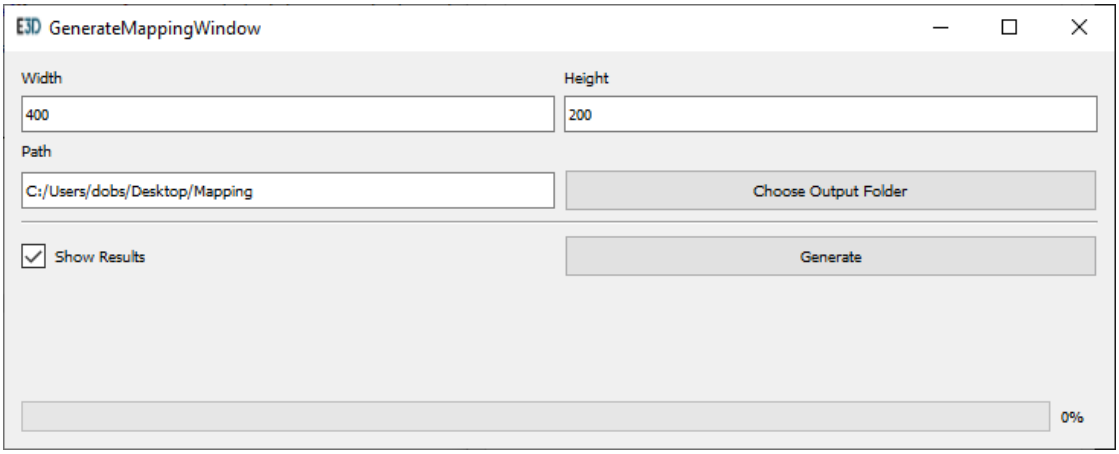


Figure 5.8: The panorama generation template export pop-up lets the user define the path of where the mapping should be stored.

CHAPTER 6

Evaluation and Results

In order to produce test data and a verifyable ground truth, two test scenes were set up in Blender [Com18]. The results of the parameter generation are analyzed under the criteria of accuracy, visual quality and computing performance. To determine the accuracy of the autocalibration, the specific parameters of the cameras in the simulated data are compared to results found by the extrinsic calibration algorithm.

6.1 Synthetic Test Setup

6.1.1 Scenes and Configurations

The algorithm was tested against two different scenes. Each scene consists of a ground-plane with a texture applied to it at a few meters vertical distance to the camera rig. Objects were placed at a distance where they would produce noticeable parallax. The scenes differ in how the objects were placed and the available texture in the background. Scene 1 (see Figure 6.1 and Figure 6.2) has the objects placed around the camera rig, while in Scene 2 (see Figure 6.3) the objects are placed in straight lines moving away from the rig. Three camera configurations were tested in both scenes with objects and the ground plane present, and then with neither the ground plane nor the objects, in order to measure the influence of parallax effects. Additionally, each testrun was executed once with and without the optional cube-reprojection in order to examine the influence of distortion when matching features. Before each test run, all adjustable camera parameters were randomly perturbed. In total, this amounts to 16 different test cases of which the results were averaged over 100 test runs.
The tests were carried out in three different configurations regarding the camera orientations and their field of view. In configuration 1 and 2, the side-cameras were rotated 90° to the left or to the right, respectively. In configuration 3, the side-cameras were

rotated 120° to the left or right and then elevated 30° towards the top. The field of view is successively increased from configuration 1 to 3, starting at 120° , then 150° to 180°. The three field of view setups are called Low FOV, Medium FOV and High FOV from here on. It should be noted that a higher field of view coincides with more distortion, as a larger section is projected into the same image size (in pixels). The three configurations are listed in Table 6.1. Raw images and their undistorted counterpart, depicting the field of view for each configuration are shown in Figures 6.4, 6.5 and 6.5.

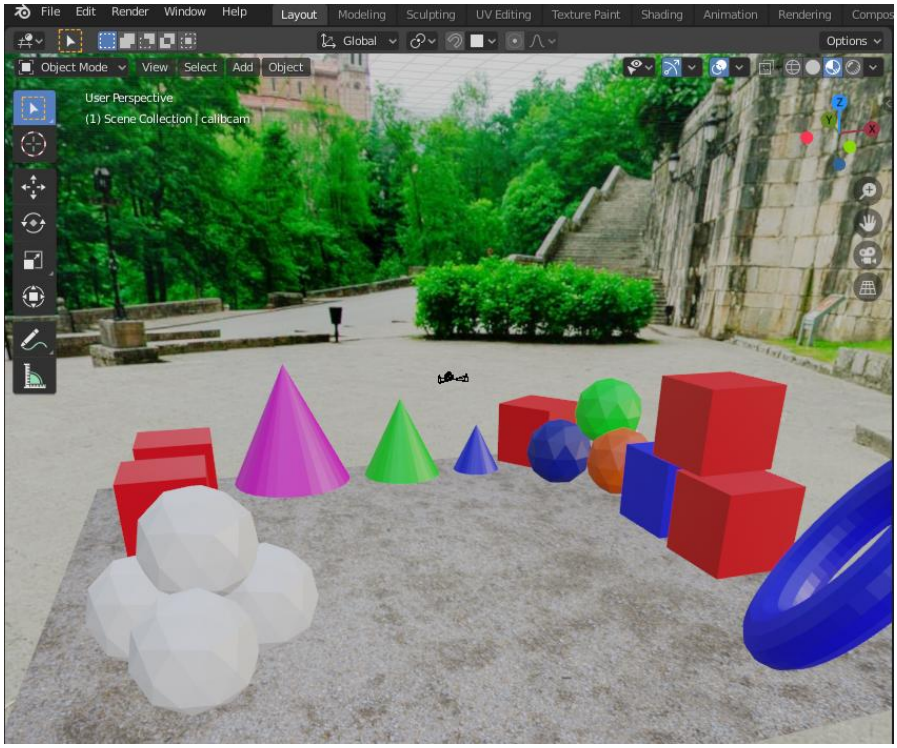


Figure 6.1: The test scene modeled in Blender. The background is a sphere with a panoramic image[1] attached to it, while the foreground consists of a textured ground and various objects.

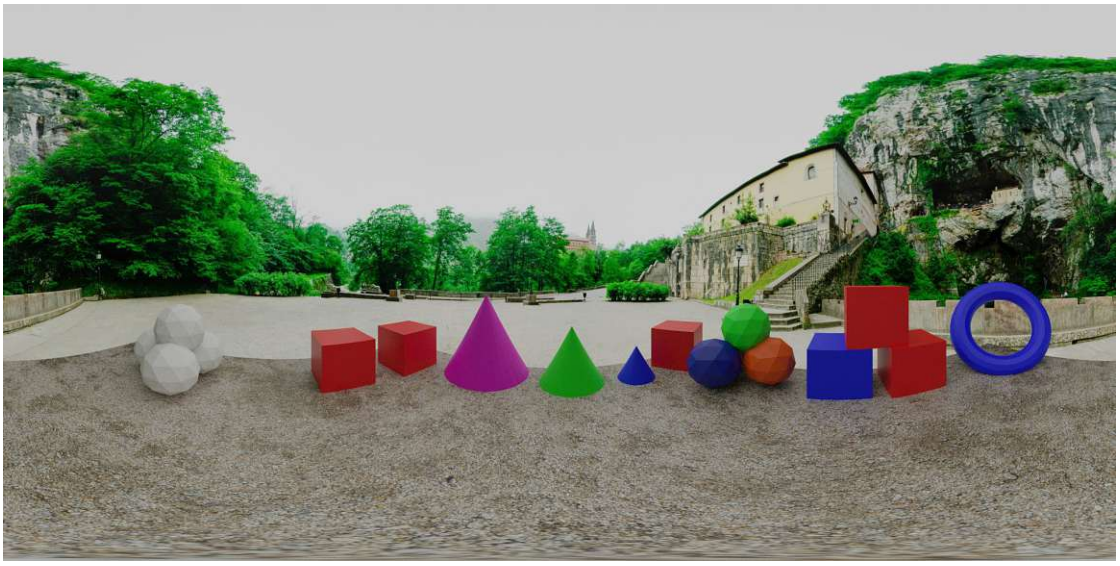


Figure 6.2: Scene 1 displayed as an equirectangular panorama, rendered in Blender.

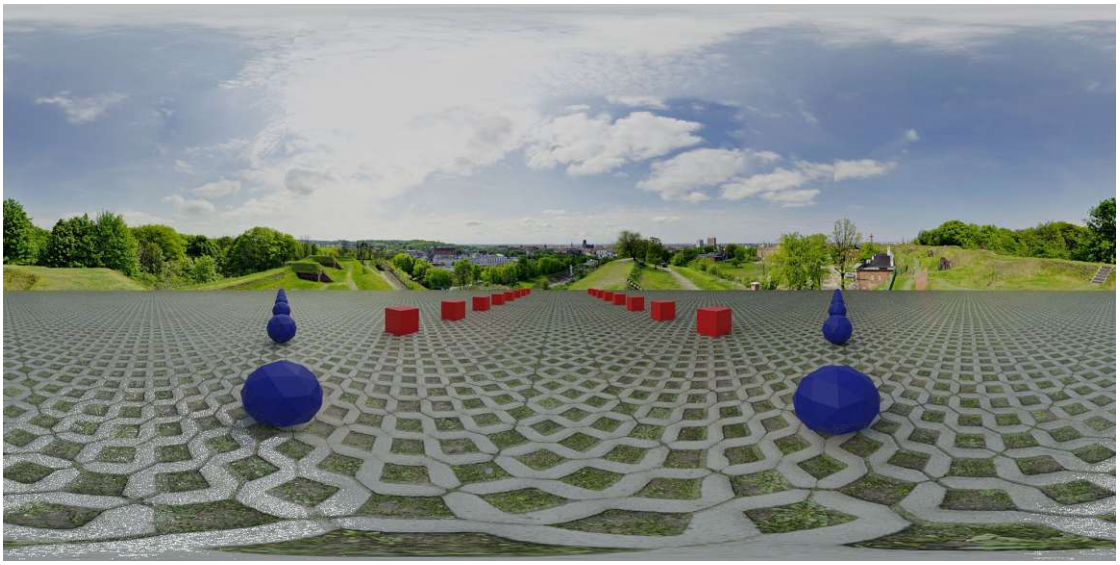


Figure 6.3: Scene 2 displayed as an equirectangular panorama, rendered in Blender.

[1]The background images are taken from Texturify `https://texturify.com/`

Table 6.1: The three test configurations and their parameters. The angles noted in the table describe how far the side cameras are rotated away from the front facing cameras. The field of view for all cameras is different for each configuration. It is smallest in configuration 1 and largest in configuration 3.

Configuration	Horizontal Angle	Vertical Angle	Field of View
1 (Low FOV)	90°	0°	120°
2 (Medium FOV)	90°	0°	150°
3 (High FOV)	120°	30°	180°

Figure 6.4: Configuration 1, source (left) and undistorted (right) image.

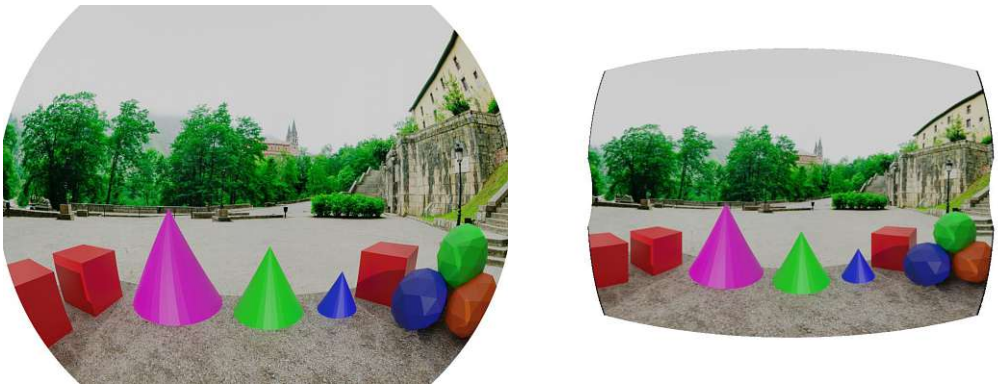


Figure 6.5: Configuration 2, source (left) and undistorted (right) image.

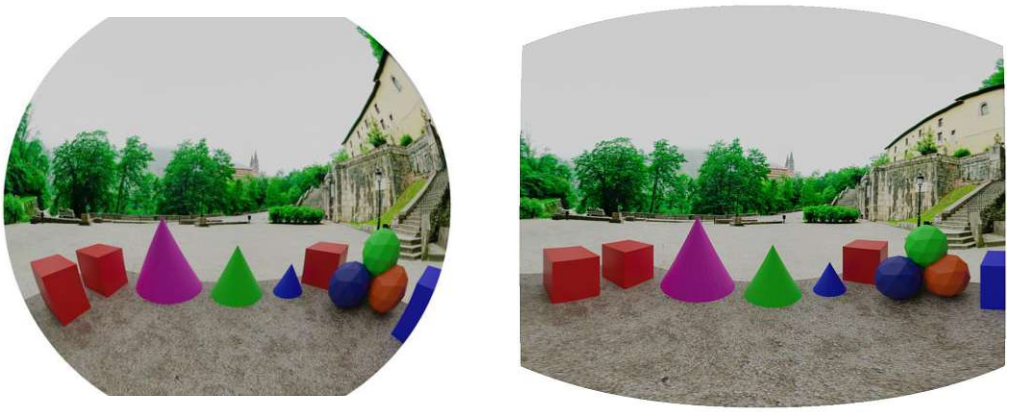
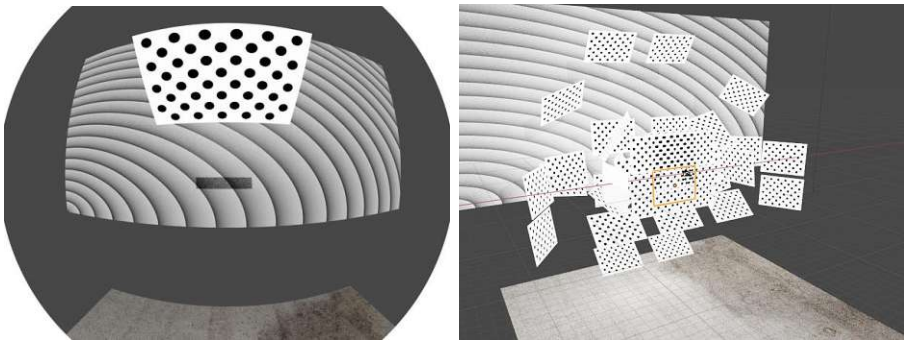


Figure 6.6: Configuration 3, source (left) and undistorted (right) image.

6.1.2 Calibration

To calibrate the virtual cameras, a circle grid was constructed in Blender and positioned at different angles and distances to the camera and images rendered. Figure 6.7 shows a single calibration image as well as the scene which was constructed to render muliple images. With this setup, it was possible to test varying camera types and configurations without the need for a physical set of lenses. Generating the set of calibration images for a new camera configuration was fully automated in Blender.



(a) (b)

Figure 6.7: Images representing the calibration process of the virtual cameras. (a) shows one calibration image of around 40 usually taken per camera. (b) shows the calibration scene simultaneously including all calibration patterns.

6.2 Feature Matching Results

This section presents the results of the feature extraction and matching stages. The goal of this stage is to maximize the extraction of distinct features, reduce the amount of

impossible features (features in non-overlapping areas) and remove bad or redundant feature matches. In all configurations 15000 features were extracted per image with 32 pyramid levels.

6.2.1 Region Mask

Before any features are extracted for the registration or alignment stage of the algorithm, a feature mask (see Figure 6.8) is generated to reduce the area which features are taken from. While this thesis expects images to be aligned manually before executing this step, a RANSAC approach could also be used for automation instead [Ho17].

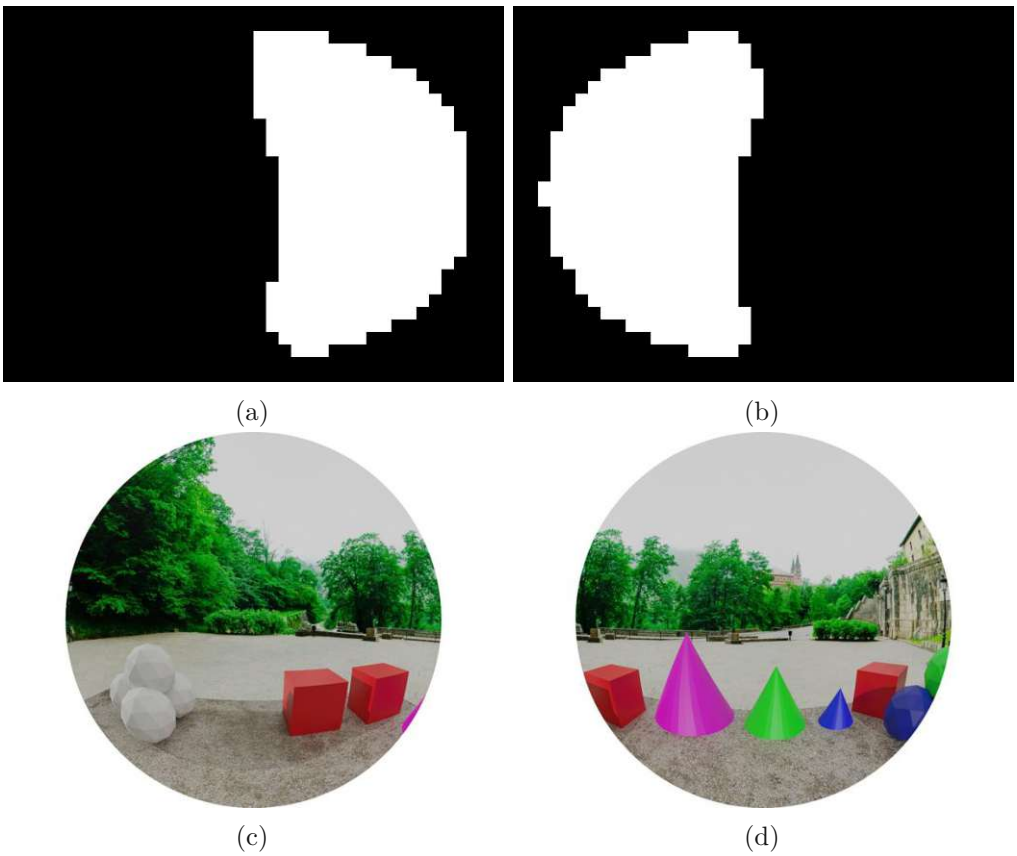
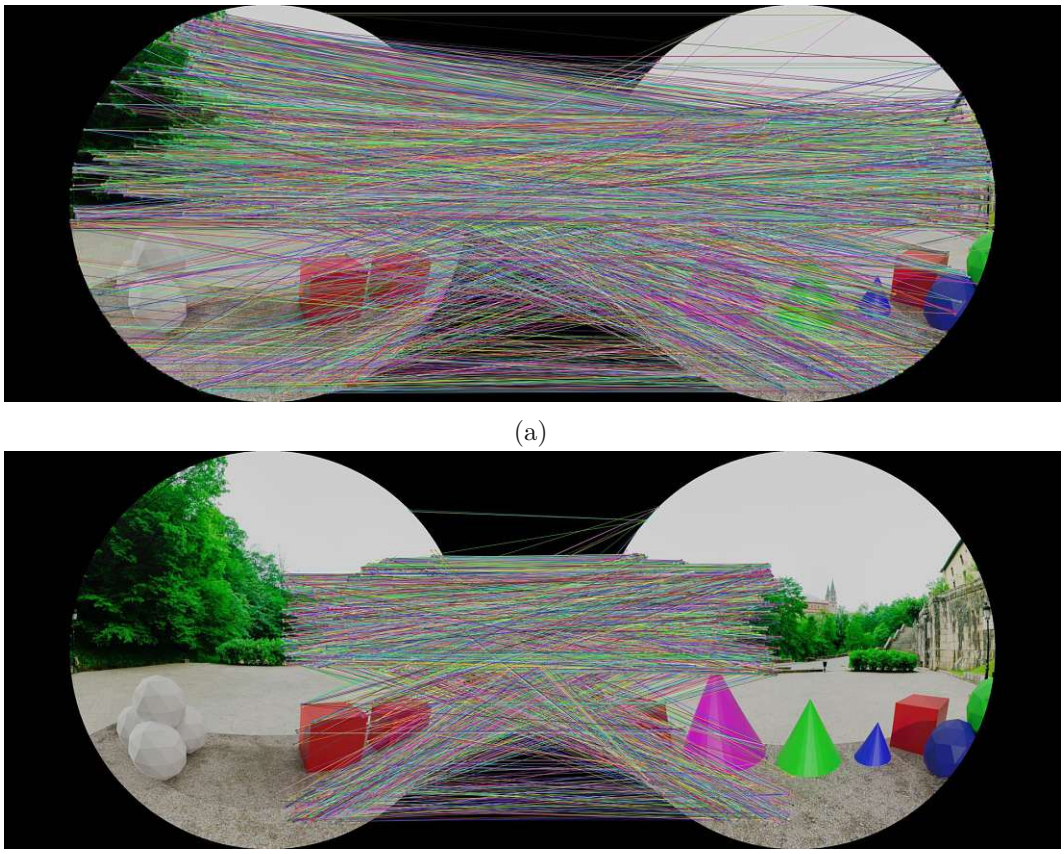


Figure 6.8: Images taken from the region mask generation. Images (a) and (b) show the masks which are combined with images (c) and (d) when extracting feature points from the images.

The feature extractor will generate features in any salient area. Saliency means that the feature detector will find many feature points in an area (in the case of ORB, this means corner-rich areas). By excluding parts where there is no overlap, the density of features

in areas with overlap will increase, as the ORB algorithm extracts a fixed amount of features. This then generates more features which possibly match. In turn, this means that the more overlap there is between two images, the less improvement can be achieved. In our tests this led to a 100% increase in usable feature matches in configuration 1, which had the smallest overlap. With configuration 2, there was still an increase of around 50% of usable features, while for configuration 3 only around 15% more features were found. As can be seen in Figure 6.9, there is a large amount of impossible matches present without using a mask. Simply preselecting the areas in which the features are detected removes many unwanted features, as matches cannot be found in non-overlapping areas. Depending on the setup, the gain in useful feature-matches can be significant. In our tests of configuration 1, this removed around 50% of unwanted features, which in turn means that 50% more features were now extracted from overlapping areas.



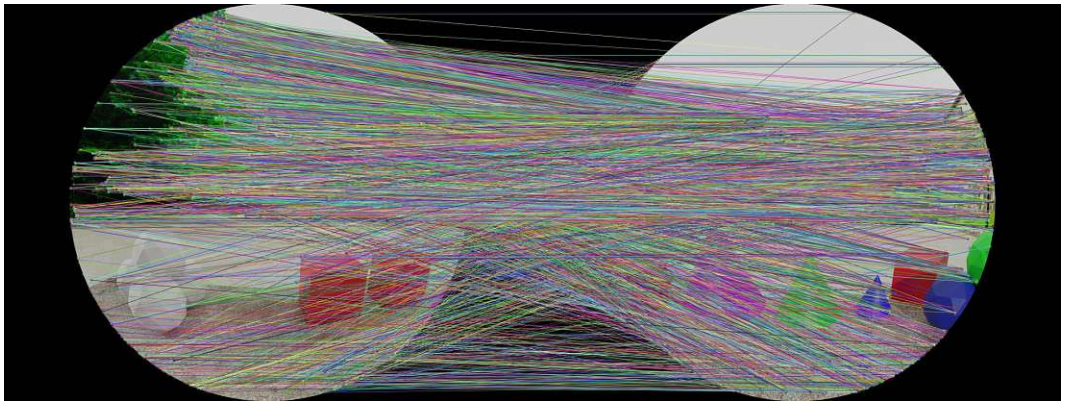
(a)

(b)

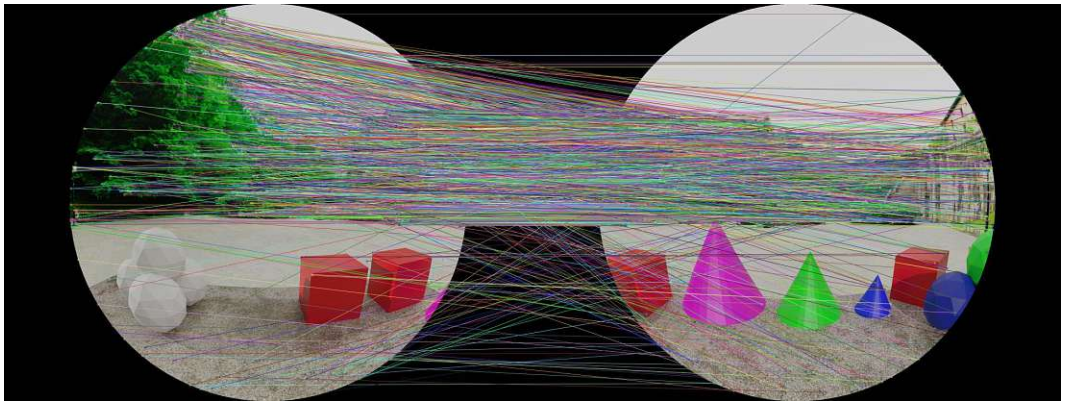
Figure 6.9: Feature matching with and without using region masks. (a) shows the result of extracting and matching feature points without using a mask, while (b) shows the result when using a mask.

6.2.2 Ratio Test

After extracting and matching the features, many bad or impossible matches are part of the result. Bad matches are removed by first applying a ratio test [Low04] with a similarity cut-off at 0.85. For configuration 1 this led to a reduction in bad matches from around 14000 to 4000 matches as demonstrated in Figure 6.10.



(a)



(b)

Figure 6.10: Feature matches are reduced by applying the ratio test. (a) shows the resulting feature matches before applying the ratio test and (b) shows the result after the ratio test was applied. The similarity of the features on the ground area leads to them being more likely eliminated.

Although many of the false matches are thereby removed, some clear outliers are still part of the result. For example, there are many feature matches between parts of the images in Figure 6.10(b) which are only present in one of the images. The algorithm expects the cameras to be correctly aligned to each other within around 15° and any match outside of this error margin is simply removed. As a result, Figure 6.11 shows

the feature matches after applying the region masks, the ratio test and removing any impossible outliers based on their rotational differences.

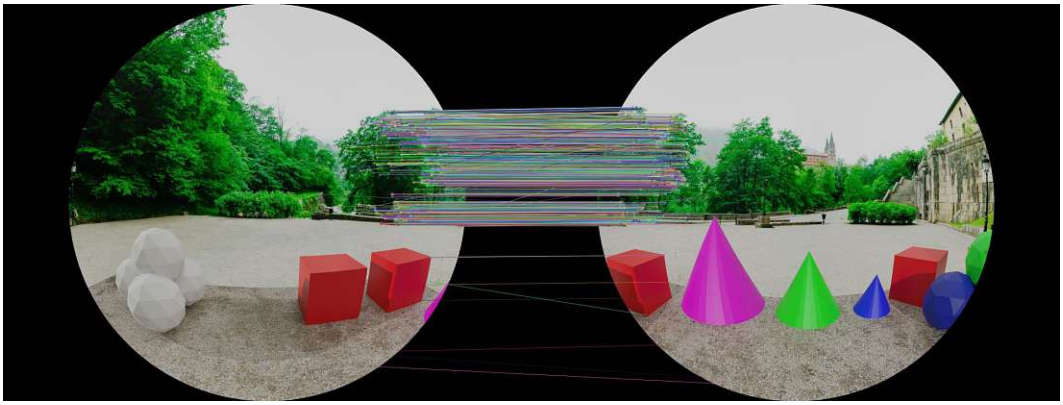


Figure 6.11: The remaining feature matches after removing outliers.

6.2.3 Grid Reduction

Salient areas (trees) contain many feature matches, while others contain less (textureless objects) or none (grey sky). To reduce the bias towards areas with many feature matches and increase the influence of less saturated areas, features are reduced in a grid-like fashion. As can be seen in Figure 6.12, the image is first divided into cells of 100 by 100 pixels each and then all but the 3 best matches (according to similarity) are removed from the result. If the 3 best matches are less than 10 pixels apart, a different set of matches is selected. Figure 6.13 shows the result after applying the reduction to the result shown in Figure 6.11. Not only does this remove bias, but it also reduces the complexity of the bundle adjustment later in the optimization stage. While this already provided a good amount of usable matches, further strategies were employed in order to increase the amount of good feature matches.

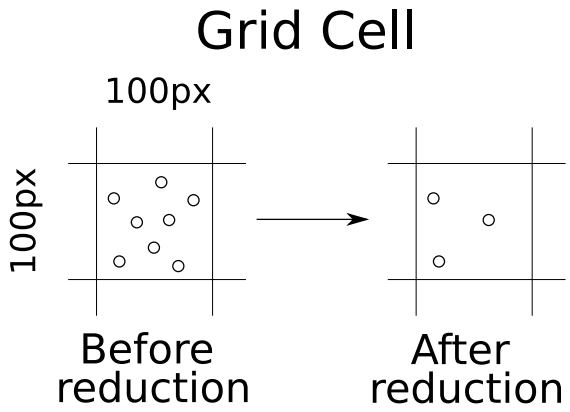


Figure 6.12: Example of a cell with the dimensions of 100 by 100 pixels containing many features. After reduction, each cell is limited to a predefined maximum of features in order to reduce bias towards some areas of the panorama.

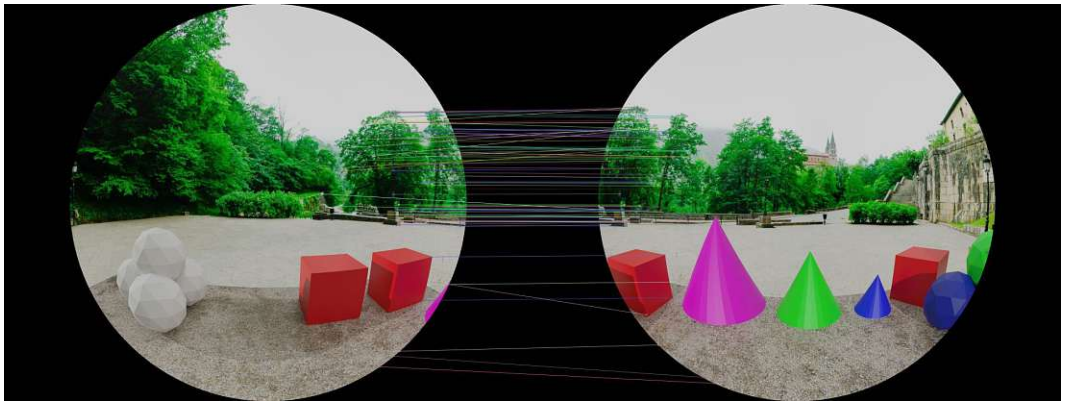


Figure 6.13: The result after applying a grid-like reduction

6.2.4 Cubemap Reprojection

Another step to improve the comparability of feature descriptors is to remove as much distortion as possible before extracting them. Reprojecting the image to a cubemap improves the feature descriptors twofold. On one hand, each of the cubemap sides is a 90° pinhole image where the radial distortion is removed. On the other, the reprojection makes use of the camera parameters, which takes the lens and camera imperfections into account. As ORB features are not designed to specifically handle the type of radial distortion introduced by fisheye-lenses [CLÖ+12], a reprojection which includes undistorting the image to a cubemap should improve the matching results. Figure 6.14 shows the result of transforming a circular image to a cubemap. Figure 6.15 shows the amount of usable feature matches without reprojection. No grid-like reduction of features

was applied, apart from the ratio test and the outlier removal and the camera parameters were randomly perturbed prior to feature extraction and matching. In configurations 1 and 2, there is only a little amount of distortion towards the rim, as the field of view is relatively low. More overlapping area also produces more feature matches in configuration 2. A sharp decline in feature matches can be found in configuration 3. Even though the image covers more overlapping area than configuration 1, the stronger distortion reduces the amount of usable feature points. More of Scene 2 is covered by blue sky than forest background, which explains the further drop in feature matches between the two scenes under configuration 3. Figure 6.16 shows the amount of feature matches after first projecting the images to cubemaps. The sharp drop in matches with configuration 3 is no longer an issue. Configuration 2, which features the largest overlap and relatively low distortion, again produces the best outcome. When strong distortion from a high field of view is present in the overlapping area, the expensive reprojection provides a good option for improving the results.

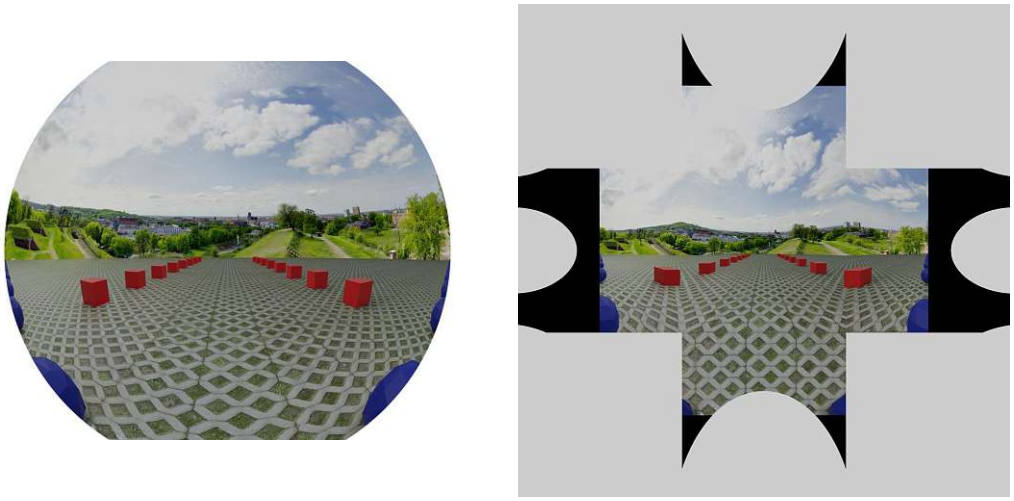


Figure 6.14: A circular image is reprojected to a cubemap in oder to remove some of the distortion.

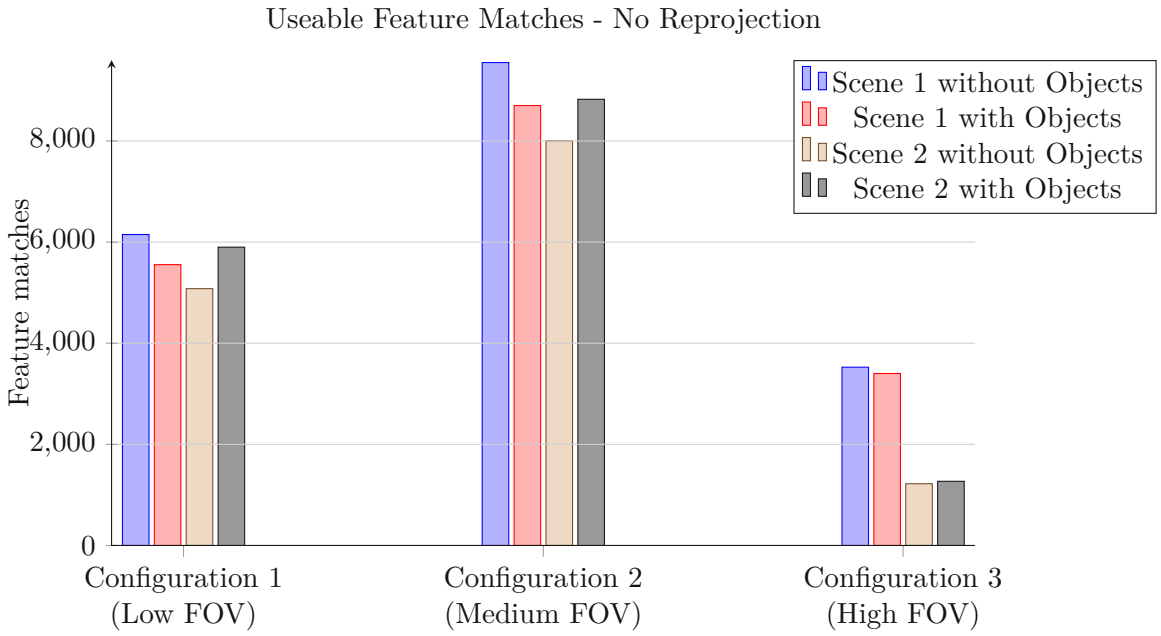


Figure 6.15: The amount of feature matches in each of the configurations after applying the ratio test and outlier removal.

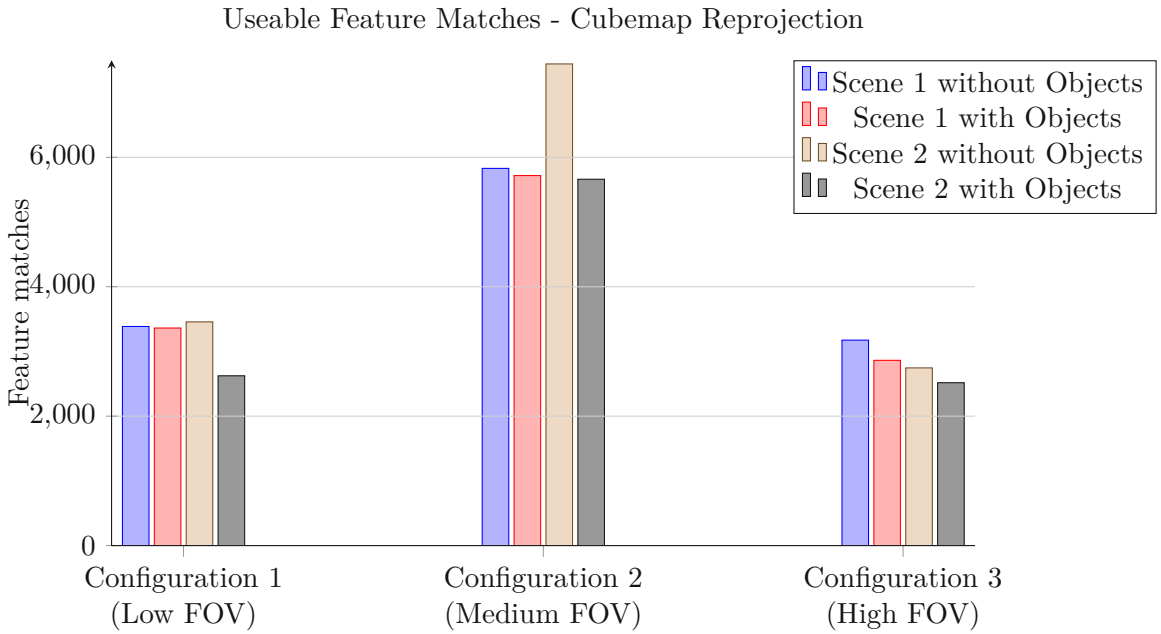


Figure 6.16: The amount of feature matches in each of the configurations after applying the ratio test and outlier removal, when first projecting the circular images to cubemaps.

6.3 Registration

The registration tests were run again with the same Low, Medium and High FOV configurations as mentioned earlier. Each test run consisted of estimating the parameters for three overlapping cameras. One of the cameras was looking forward and was fixed, which means its parameters were not updated (it is simply the camera to which the others are aligned). The parameters for the other two cameras, simply termed "left" and "right" according to the direction they are pointed at in comparison to the fixed camera, are then estimated. In order to produce a meaningful representation of the algorithm's accuracy, the forward and up vectors after registration were compared to the ground truth. A lower value indicates higher accuracy of the algorithm, but the visual quality of the final panorama also heavily depends on the other camera parameters. Comparing the estimated forward and up vectors to the ground truth therefore allows for the evaluation of only the registration step.

Two approaches for registration were implemented and tested. The first of the two is based on projecting a ray from a feature match and then optimizing the camera properties (rotation, calibration) as well as the ray direction. Reprojecting the ray to each of the cameras and measuring the distance from the original feature position then serves as the error metric. In contrast, the second method projects both feature positions into the equirectangular panorama and uses the distance between them as the error metric.
The two approaches were then tested under various conditions, see Figure 6.17 for a visual representation. There are 2 scenes in which the 3 camera setups (Low FOV, Medium FOV and High FOV) were tested. Each combination of scene and camera setup were tested under 16 different settings (from 4 binary options). The options are:

- Include optimizing camera distortion parameters or not
- Are there objects at a parallax present in the scene or not
- Are circular images reprojected to a cubemap before feature extraction and matching or not
- Is the Ray optimization or the Equirectangular optimization used

With 2 scenes, 3 camera setups and 16 combinations of settings, this results in a total of 96 test setups. All results were averaged over 10 runs. Before each run, all of the camera parameters were perturbed, by modifying each parameter by a random value.
In order to evaluate the results, the entire process has been automated. First the program loads each configuration and produces a comma separated file containing the results. A further script then reads these files and compiles the data into a single database to be examined.

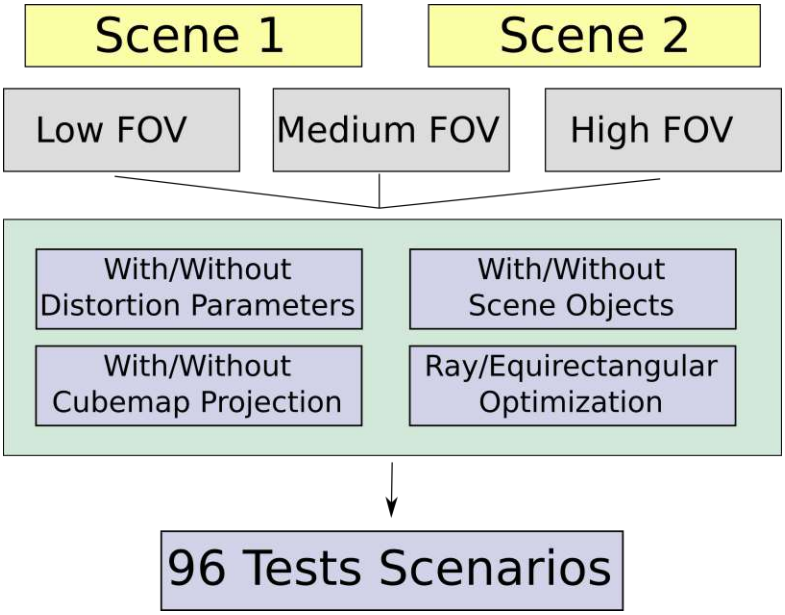


Figure 6.17: The test setup contains 2 scenes, 3 configurations and various other settings, resulting in a total of 96 test scenarios.

6.3.1 Results

The results of the algorithm were satisfactory in all configurations except for some outliers. In almost all cases, the up and forward vectors differed by less than half a degree from the ground truth. In the better configurations, the difference was below 0.2 degrees. In the following tables, there are two values given for accuracy. "Left Accuracy" is calculated by comparing the estimated forward and up vectors of the left camera to its ground truth. This results in 2 values, each representing a rotational difference in degrees. They are summed up and averaged over 10 test runs. The "Right Accuracy" is the same value for the right camera. A value of 1 implies that a camera is, on average, off by one degree of the ground truth.

Scenes

The accuracy of the registration was similar in both scenes and under all other options (see Table 6.2). Slightly more feature points were found in Scene 2, mainly due to the higher saliency of the background in this setup. No notable differences resulted from the slightly different scene setups in regard to the results. Both scenes resulted in a similar amount of matches and accuracy.

Table 6.2: The feature matches and accuracy compared by the scene they were taken from. There is no notable difference in the amount of matches or the accuracy between scenes.

	Feature Matches	Left Accuracy	Right Accuracy
Scene 1	221	0.308	0.315
Scene 2	226	0.310	0.330

Configuration & Field of View

The difference in field of view and hence overlap had a strong influence on the feature matches used (see Table 6.3). The second configuration produced almost 40% more features due to the higher overlap between cameras, which in turn led to the best results for accuracy. The higher distortion in Configuration 3 led to a lower overall accuracy (a higher value means more difference to the ground truth).

Table 6.3: The feature matches and accuracy compared by the camera configuration used during registration. The best results were generated from Configuration 2 in which the overlap was the largest and the distortion relatively small.

	Feature Matches	Left Accuracy	Right Accuracy
Configuration 1	167	0.338	0.364
Configuration 2	305	0.166	0.231
Configuration 3	198	0.424	0.373

Influence of Parallax

The influence of the parallax was tested by comparing the results of the scenes with and without objects and a floor present (see Table 6.4). Due to the objects having been rendered without texture, they were not considered salient by the feature extractor, which resulted in fewer feature matches. Overall, the accuracy dropped slightly with objects present.

Table 6.4: The feature matches and accuracy when objects producing a parallax were present compared to when there were no objects present in the scene. Due to the textureless objects, fewer feature matches were extracted and the accuracy dropped slightly when objects were present.

	Feature Matches	Left Accuracy	Right Accuracy
With Objects	203	0.301	0.296
Without Objects	243	0.317	0.349

Cube Reprojection

Using cube reprojection resulted in more usable feature matches (see Table 6.5), mostly in combination with Configuration 3 and its high distortion. Overall, more feature matches

were discovered which had no effect on accuracy.

Table 6.5: The feature matches and accuracy when the images were first reprojected to cubemaps. Although more feature matches were produced, no change in accuracy was observed.

	Feature Matches	Left Accuracy	Right Accuracy
With Cube Reprojection	265	0.311	0.314
Without Cube Reprojection	182	0.307	0.332

Ray- vs. Equi-optimization

A direct comparison between the two approaches showed that both, ray- and equi-optimization, work similarly well (see Table 6.6). As both approaches optimize a similar error, this is in line with our expectations.

Table 6.6: The feature matches and accuracy compared by the optimization strategy employed. Both ray- and equi-optimization produce similar results.

	Feature Matches	Left Accuracy	Right Accuracy
Equi-optimization	223	0.307	0.323
Ray-optimization	223	0.312	0.322

Property Optimizing

Including the optimization of the camera (see Table 6.7), we observed that distortion properties only worked well when the features were spread across the entire images, which was only the case in Configuration 3, which had an overlap on both sides of the left and right cameras. If the feature matches are only present on one side of the image, the algorithm optimizes the error while ignoring the distortions produced in the rest of the image. As a result, including the optimization of the distortion parameters is only useful when features are present across the entire image, as otherwise the quality of the panorama is reduced.

Table 6.7: The feature matches and accuracy compared by the use of parameter optimization. The improvement was dependent on the spread of feature matches and only improved in Configuration 3, where the feature matches were spread evenly.

	Feature Matches	Left Accuracy	Right Accuracy
Optimizing distortion	223	0.355	0.440
Not optimizing distortion	223	0.263	0.205

Summary

We found that the scenery had little influence on the result, while objects producing parallax reduced the accuracy slightly. A larger amount of overlap produced a significantly

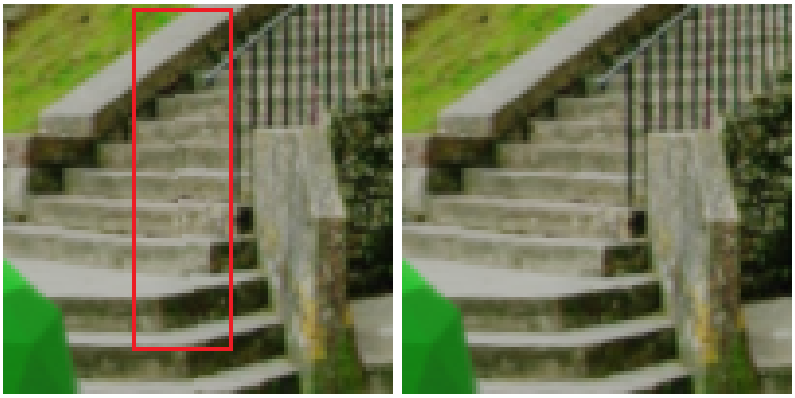
better result, if the overlap did not come at the cost of higher distortion. Cube reprojection was only useful when working with strongly distorted images and there was little difference in the ray- and equi-optimization approaches.

6.4 Visual Quality

The most important aspect with regards to visual quality is the presence of a visible seam at camera overlap regions. The main culprits responsible for visible seams are parallax from objects, misalignment and miscalibration. In most cases, it is impossible to produce a seamless panorama from alignment only and further techniques are needed. After the registration step, the result will in most cases have a small seam present. A strategy is then necessary in order to remove the visible seam and produce a more visually continuous panorama image.

6.4.1 Moving Least Squares

The strategy used to resolve seams between two images is to deform the image in order to make the feature points overlap in the panorama. Figure 6.18 and Figure 6.19 demonstrate the effect of this approach.



(a) (b)

Figure 6.18: A small area of the panorama is enlarged in order to show the seam before and after applying the moving least squares approach. (a) shows a small seam of a few pixels is visible at the stairs after running the alignment algorithm. (b) shows the same alignment after applying the moving least squares deformation, effectively removing the seam.

(a) (b)

Figure 6.19: Another example of a seam taken from Scene 2. The moving least squares approach eliminates the seam entirely. (a) shows a very noticeable seam, resulting from misalignment. After applying the moving least squares approach in (b), it is no longer noticeable.

This approach requires features present around the seam, as it will not produce results otherwise. Figure 6.20 demonstrates the algorithm applied to the full panorama.

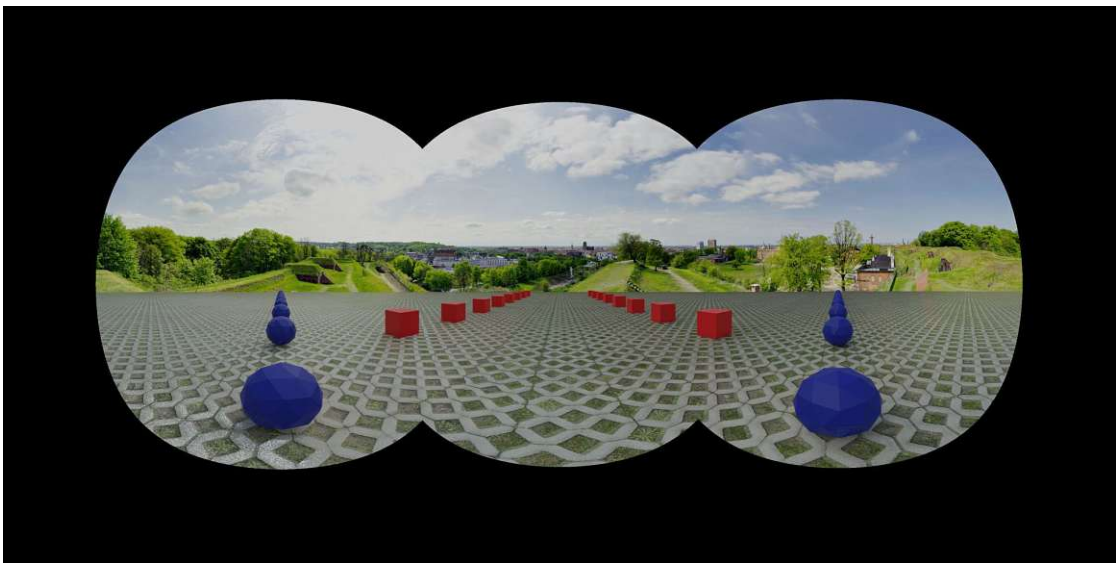


Figure 6.20: Result of applying the moving least squares approach to the full panorama.

6.5 Performance

The test system was an AMD Ryzen 7 1700X 8-Core CPU with a Geforce GTX 1080. The two parts which were evaluated in terms of performance are the extrinsic calibration and rendering time of a panorama. Calibrating the extrinsic parameters requires first the extraction of the overlap region, then the extraction of features from said region, the matching of those features, the removal of outliers and then solving a linear system of equations in order to estimate the camera parameters. The results in Table 6.8 were produced by solving for two side and one stereo-camera without shared overlap between the side-cameras. As a result, all steps except the solving were executed once for each side-camera in relation to the stereo-camera.

Table 6.8: Average performance of extrinsic calibration in seconds.

No. Features	5000	10000	15000
Finding Overlap	Average 0.05s		
Feature Extraction	3.111s	3.262s	3.344s
Outlier Removal	0.006s	0.015s	0.018s
Solving	0.009s	0.011s	0.013s
Total	3.126s	3.238s	3.344s

The panorama is rendered almost entirely on the GPU. The most important factors influencing the computing time are the size of the panorama, the area of the panorama covered by cameras and the amount of control points for the moving least squares approach. The amount of control points for the moving least squares approach was limited to 30 in total or 15 per camera pair, in order to produce consistent results. The results can be found in Table 6.9.

Table 6.9: Average performance of panorama generation in milliseconds.

	Camera Setting 1	Camera Setting 3
2000 x 1000	30ms	41ms
2000 x 1000 MLS	67ms	87ms
4000 x 2000	120ms	137ms
4000 x 2000 MLS	232ms	237ms

The area of the panorama covered by camera in case of Configuration 1 was 51% and 81% for Configuration 3. The results of Configuration 2 were almost identical to Configuration 1.

CHAPTER 7

Conclusion & Future Work

This chapter will give a summary and critical review of the development process and suggestions as to how the imlemented system could be further improved.

7.1 Conclusion

The aim of this thesis was to create a system which enables the real-time configuration and manipulation of a multi-camera, high field of view panorama with as much automation as possible. The development progressed in stages and began with implementing the GUI, which was later expanded when needed. Integrating the intrinsic calibration was rather straight-forward, as the algorithm was already implemented in the OpenCV library and little work was necessary to process the images in parallell. One of the more challenging and often revisited stages was the viewer, as it combines many of the concepts and models presented in the thesis. After the basic implementation of the viewer was completed, which allowed to project multiple images into the panorama, the extrinsic calibration was up next. While the algorithm was developed in a brief period of time, various issues arose from bad feature point matches and no directly available ground truth besides physically measuring the rig. The lack of specialized equipment to either mount the cameras within a considerable error range or find the camera parameters, led to the designing of the synthetic test setup. Being able to produce various test scenarios and immediately find bad feature matches was a huge improvement to the development process. This allowed us to align the images to a satisfactory degree and move on to dealing with seams, which are a result of the parallax between cameras. The moving least squares approach was added, which provided a satisfactory result in terms of visual continuity. Many setbacks plagued the development, especially during the feature matching stage of the project. Most feature extractors do not work properly for fisheye-lenses and either need preprocessing or produce low-quality results. We found that the extractors which do exist for fisheye-lenses often have no properly available implementation or are not open

source.
The results of the overall pipeline are visually pleasing and can often produce panoramas with no noticeable seam in the case of still images. Due to the implementation on the GPU, working with the GUI is possible in real time on consumer hardware.

7.2 Future Work

There are three major avenues for how the current system could be improved: Either through extension of the implemented algorithms, an improvement of the accuracy or a speedup through optimization. We review the three options in the following.

The system currently works with 2D points to find the relative rotation between the cameras. If the algorithm was extended to work with 3D points, the distance between the cameras as well as their relation to the points in world-space could be used as context information when rendering the panorama. This could be used to modify the radius of the sphere at certain positions, before the images are projected onto it and before applying further techniques like moving least squares. The resulting system would then more closely resemble a typical bundle adjustment SLAM algorithm. Currently, there is also the need for an approximate alignment of the cameras before the algorithm can be executed, which could be removed by adding an additional automated alignment step at the start of the pipeline.

The accuracy of the algorithm could also be improved by using feature points designed specifically for fisheye-lenses. Feature detectors for regular images do not work well with high field of view fisheye-lenses, especially in the regions close to the outer rim of the image. This effect worsens with increasing field of view. Preprocessing by reprojecting the entire image to a cubemap is a computationally expensive step and a proper feature detector could provide much faster and more accurate results. In turn, this would also provide better features to be used in the extrinsic calibration part of the pipeline.

Finally, there are several parts in the implementation which would benefit from optimization. The rendering of the panorama is slowed down mostly by the moving least squares approach. At each fragment, or pixel, the entire equation is calculated to warp the image, which adds substantial computation time. Instead, the moving least squares approach could be applied in a grid-like fashion, only precalculating the corners of the grid and then interpolating all pixels inbetween. If the extrinsic calibration were to be used in real time, the feature extraction would have to be drastically improved, although solving the system derived from the features is already fast enough.

Bibliography

[AAMDG12] S. Ait-Aoudia, R. Mahiou, H. Djebli, and H. Guerrout. Satellite and aerial image mosaicing - A comparative insight. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 652–657, 2012.

[Ana89] P. Anandan. A computational framework and an algorithm for the measurement of visual motion. *International Journal of Computer Vision*, 2(3):283–310, 1989.

[AOV12] A. Alahi, R. Ortiz, and P. Vandergheynst. FREAK: Fast retina keypoint. In *Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 510–517, 2012.

[BAHH92] J. R. Bergen, P. Anandan, Keith J. Hanna, and Rajesh Hingorani. *Hierarchical Model-Based Motion Estimation*. Springer, Berlin, Heidelberg, 1992.

[Bar19] J. T. Barron. A general and adaptive robust loss function. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4326–4334, 2019.

[Bet05] F. Bettonvil. Fisheye lenses. *Journal of the International Meteor Organization*, 33:9–14, 2005.

[BL07] M. Brown and D. G. Lowe. Automatic panoramic image stitching using invariant features. *International Journal of Computer Vision*, 74(1):59–73, 2007.

[Bra00] G. Bradski. The OpenCV Library, 2000. Accessed: 05.17.2021, Available: `https://opencv.org/`.

[CC13] S. Chupty and J. De Coninck. Head mounted displays, 2013. Accessed: 05.17.2021, Available: `http://www.student.kuleuven.be/~s0214423/CapSel1314/hmds004.pdf`.

[CLÖ+12] M. Calonder, V. Lepetit, M. Özuysal, T. Trzcinski, C. Strecha, and P. Fua. BRIEF: Computing a local binary descriptor very fast. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(7):1281–1298, 2012.

[Com18] Blender Online Community. Blender - a 3d modelling and rendering package, 2018. Accessed: 05.17.2021, Available: `http://www.blender.org`.

[CT00] S. Coorg and S. Teller. Spherical mosaics with quaternions and dense correlation. *International Journal of Computer Vision*, 37(3):259–273, 2000.

[FB81] M. A. Fischler and R. C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[Ho17] T. Ho. Dual-fisheye lens stitching for 360-degree imaging. *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 1–5, 2017.

[LK81] B. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 674–679, 1981.

[Low04] D. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[Mar04] C. B. Martin. Design issues of a hyperfield fisheye lens. In Jose M. Sasian, R. John Koshel, Paul K. Manhart, and Richard C. Juergens, editors, *Novel Optical Systems Design and Optimization VII*, volume 5524, pages 84 – 92. SPIE, 2004.

[Miy64] K. Miyamoto. Fish eye lens. *Journal of the Optical Society of America*, 54(8):1060, 1964.

[Mor78] J. J. Moré. The levenberg-marquardt algorithm: Implementation and theory. In G. A. Watson, editor, *Numerical Analysis*, pages 105–116, Berlin, Heidelberg, 1978. Springer.

[MR07] C. Mei and P. Rives. Single view point omnidirectional camera calibration from planar grids. In *International Conference on Robotics and Automation (ICRA)*, pages 3945–3950, 2007.

[OBS99] A. V. Oppenheim, J. R. Buck, and R. W. Schafer. *Discrete-Time Signal Processing; 2nd ed.* Prentice-Hall, Upper Saddle River, NJ, 1999.

[PAG11] T. Panarungsun, S. Auethavekiat, and D. Gansawat. Foreground rejection for parallax removal in video sequence stitching. In *International Symposium on Intelligent Signal Processing and Communications Systems (ISPACS)*, pages 1–6, 2011.

[PCT+17] T. M. Porcino, E. Clua, D. Trevisan, C. N. Vasconcelos, and L. Valente. Minimizing cyber sickness in head mounted display systems: Design guidelines and applications. In *International Conference on Serious Games and Applications for Health*, pages 1–6, 2017.

[RFP08] R. Raguram, J.-M. Frahm, and M. Pollefeys. A comparative analysis of ransac techniques leading to adaptive real-time random sample consensus. In D. Forsyth, P. Torr, and A. Zisserman, editors, *Computer Vision – ECCV 2008*, volume 5303, pages 500–513, Berlin, Heidelberg, 2008. Springer.

[RRKB11] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. ORB: An efficient alternative to SIFT or SURF. *International Conference on Computer Vision (ICCV)*, pages 2564–2571, 2011.

[SK] A. Sameer and M. Keir. Ceres solver. Accessed: 05.17.2021, Available: `http://ceres-solver.org`.

[SMW06] Scott Schaefer, Travis McPhail, and Joe Warren. Image deformation using moving least squares. *ACM SIGGRAPH 2006 Papers, SIGGRAPH '06*, pages 533–540, 2006.

[Sny93] P. Snyder. *Flattening the Earth: Two Thousand Years of Map Projections*. University of Chicago Press, 1993.

[SUS11] R. Szeliski, M. Uyttendaele, and D. Steedly. Fast poisson blending using multi-splines. In *International Conference on Computational Photography (ICCP)*, pages 1–8, 2011.

[Sze06] R. Szeliski. Image alignment and stitching: A tutorial. *Foundations and Trends in Computer Graphics and Vision*, 2(1):1–10, 2006.

[The21] The Qt Company. Qt : cross-platform application and ui framework, 2021. Accessed: 05.17.2021, Available: `http://qt.io/`.

[XP10] Y. Xiong and K. Pulli. Fast panorama stitching for high-quality panoramic images on mobile phones. *Transactions on Consumer Electronics*, 56(2):298–306, 2010.