A Composable and Reusable Photogrammetric Reconstruction Library

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Visual Computing

eingereicht von

Attila Szabo
Matrikelnummer 00925269

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Prof. Dr.Dr.h.c. Werner Purgathofer
Mitwirkung: Dipl.-Ing. Dr. Stefan Maierhofer

Wien, 18. Jänner 2018

Attila Szabo

Werner Purgathofer

FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

A Composable and Reusable Photogrammetric Reconstruction Library

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Visual Computing

by

Attila Szabo
Registration Number 00925269

to the Faculty of Informatics

at the TU Wien

Advisor: Prof. Dr.Dr.h.c. Werner Purgathofer
Assistance: Dipl.-Ing. Dr. Stefan Maierhofer

Vienna, 18th January, 2018

Attila Szabo

Werner Purgathofer

Erklärung zur Verfassung der Arbeit

Attila Szabo
Floridusgasse 58, 1210 Wien

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, 18. Jänner 2018

Attila Szabo

Acknowledgements

This diploma thesis wouldn't have been possible without the unrelenting spirit of scientific inquiry blossoming within the Aardvark Platform team, the opportunities to partake in industrial-scale real world projects down to the most byzantine of details provided by the VRVis Zentrum für Virtual Reality und Visualisierung Forschungs-GmbH, and the abundance of knowledge and scientific expertise maintained by the Vienna University of Technology. I am grateful for the chances I'm given, and I try to make the best of them.

Special thanks to Andreas Walch and Stefan Maierhofer for very closely scrutinising this diploma thesis, so that it may be the best it can be.

I want to thank Georg Haaser for his ungodly mathematical intuition. His involvement in this work is substantial.

Warm thanks to Harald Steinlechner for being the heart and soul of our projects.

My heartfelt thanks go out to everyone who was close to me in the past years. Thanks for engaging in inspiring discourse with me to my coworkers and professional associates Michi, Lisa, Bernhard, Lui, Maria, Maria, Daniel, Andi, Lukas, Thomas, the remaining SMAQ group, every Aardvark Platform contributor, the VRVis and the Computer Graphics institute. For their emotional support and encouragement throughout my studies, I would like to thank my family, especially my parents Jutta and Janos who also supported me financially, my sister Emmi, and my very good friends Martin, Matthias, Simon, Christoph, Tobias, Richi, Stefan, Fluttershy, the local running community for professing the joy of exercise, and my online gaming acquaintances.

Kurzfassung

Photogrammetrie beschreibt den Vorgang, Informationen aus Fotografien herauszumessen. Photogrammetrie wird heutzutage als flexiblere Alternative zu traditionellen Vermessungstechniken, wie etwa mithilfe von Laserscannern, angesehen, was hauptsächlich der digitalen Fotografie und der Verfügbarkeit erschwinglicher und leichter Konsumentenkameras zu verdanken ist. Diese Flexibilität kommt zum Beispiel auf Baustellen oder bei Drohnenflügen günstig. Allerdings ist es oft unverhältnismäßig schwierig, neuartige oder experimentelle Techniken der Photogrammetrie zu entwickeln, da verfügbare Software komplexe und schwer zu modifiziernde Implementierungen aufweist, oder der Programmcode bei kommerziellen Tools überhaupt nicht eingesehen werden kann.

In dieser Diplomarbeit präsentieren wir eine Programmbibliothek für Photogrammetrie, die Modularität und Wiederverwendbarkeit betont. Die Bibliothek zielt darauf ab, schnelles und gewisses Experimentieren in einem wissenschaftlichen Kontext zu unterstützen. Dazu leiten wir die notwendigen Komponenten her: Die Repräsentation von Eingangsdaten (Feature Extraction, Feature Matching), das Zurückrechnen von dreidimensionaler Struktur aus zweidimensionalen Beobachtungen (Pose Recovery), und das Berechnen eines global konsistenten Modells aus vielen teilweisen Beobachtungen (Structure From Motion, Bundle Adjustment). Wir gewinnen Einsichten aus dem Bereich der Funktionalen Programmierung um diese einzelnen Bausteine zu komplexeren Modulen zusammenzuführen, sowie die Erweiterbarkeit und Flexibilität auf mehreren Ebenen der Abstraktion zu gewährleisten. Diese Diplomarbeit beinhaltet begleitende Codestücke und Rekonstruktionsbeispiele, um Funktionalitäten zu illustrieren.

Abstract

Photogrammetry is the act of recovering information from photographs. Thanks to digital photography and cheap consumer cameras, the recent years have seen a rising interest in photogrammetry as a more flexible alternative to traditional surveying technologies, for example laser scanners. Specifically, the flexibility of photo cameras makes photogrammetry suitable for outdoor scenes unfavourable for heavy equipment, such as construction sites or drone flights. However, commercial photogrammetry software often makes experimentation with novel techniques difficult, since the subject is mathematically complex and code is often difficult to modify or closed source.

In this diploma thesis, we present a composable, reusable photogrammetry library which aims to facilitate confident experimentation in a scientific context. We derive important components needed for photogrammetry, including the representation of data (Feature Extraction and Feature Matching), obtaining three dimensional scene structure from two dimensional images (Pose Recovery) and computing a globally consistent model from arbitrarily many images (Structure From Motion and Bundle Adjustment). Adhering to Functional Programming paradigms, we compose these basic components to form more complex modules, and show how to build an extensible and flexible library API on multiple levels of abstraction. Accompanying code listings and examples are included, showcasing example reconstructions with our library.

Contents

Kurzfassung **ix**

Abstract **xi**

1 Introduction **1**
1.1 Photogrammetry 1
1.2 Recent Work 6
1.3 Aim of this Diploma Thesis 8

2 The Camera Model **13**
2.1 Pinhole Camera 13
2.2 Camera Model 15
2.3 Camera Parameters 17

3 Image Features **23**
3.1 Image Filters 24
3.2 Interest Points 25
3.3 Feature Extraction 26

4 Feature Matching **33**
4.1 Baseline 33
4.2 Camera Motion 35
4.3 Classification 36
4.4 Brute Force Matching 37
4.5 Motion Consistency 40
4.6 Motion Prediction 42
4.7 Composition 46

5 Multi View Geometry **49**
5.1 Homogeneous Coordinates 50
5.2 Epipolar geometry 52
5.3 Euclidean Reconstruction 55
5.4 Camera Calibration 58

6 Pose Recovery **61**
6.1 Eight Point Algorithm and Five Point Algorithm 63
6.2 Decomposition into R and t 64
6.3 Ambiguity Removal 65

7 Structure from Motion **69**
7.1 Tracks 70
7.2 On Contradicting Tracks 73
7.3 Match Graphs 73
7.4 Photo Networks 74
7.5 Application 80

8 Bundle Adjustment **83**
8.1 Global Optimisation 85
8.2 Nonlinear Least-Squares Solving 85
8.3 Application 87

9 A Composable and Reusable Photogrammetric Reconstruction Library **91**
9.1 Input Photos 93
9.2 Library API 93
9.3 Results Discussion 98
9.4 Present State 99

10 Conclusion and Outlook **101**

List of Figures **103**

Bibliography **105**

CHAPTER 1

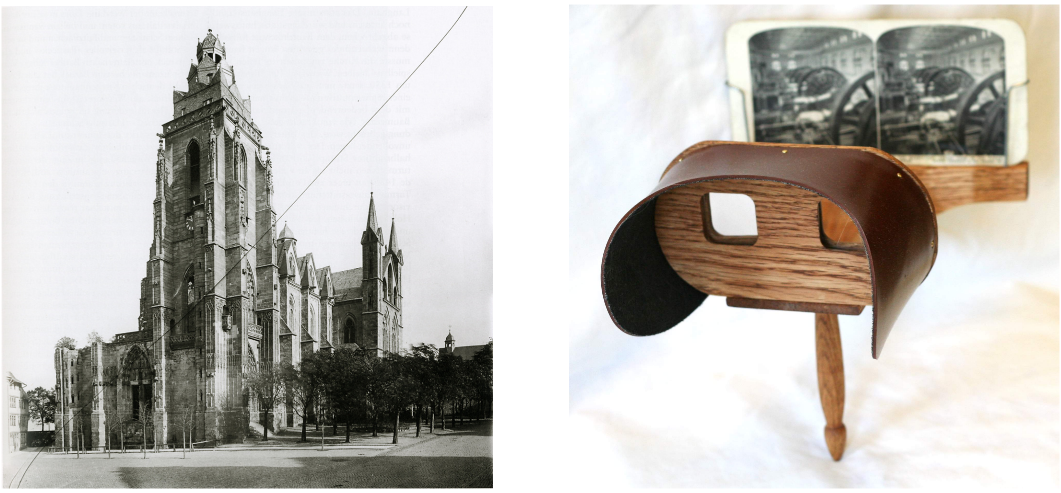
Introduction

1.1 Photogrammetry

The term photogrammetry refers to the process of taking measurements in photographs to recover information about an object.

Attempts to make measurements in photos have historically always accompanied the technological development of photography. For example, during the advent of photography in the 1850s, experiments with various chemical compounds have been made to permanently capture the image of a camera on a physical medium. British photographer Henry Fox Talbot described the different reactions of various chemicals to differing light wavelengths. He used photosensitive base plates of various chemical compositions to perform spectral analysis on light sources, and later theorised on applying the same method on telescopic photographies of the stars [Hen34]. Though imaging techniques were not yet evolved at that time, this is a first example of someone inferring material properties of an object through photographic processes.

In the 1870s, German architect Albrecht Meydenbauer developed methods to document buildings using photographs for preserving cultural heritage [Alb02]. His idea was that direct measurements on a facade could be replaced by indirect measurements in a photograph as long as enough information about the imaging process was known. His experimental first photogrammetric camera already exhibited all the required properties for a metric reconstruction: a wide-angle lens to fit an entire building into the image, a fixed image plate with a measured coordinate cross hair, as well as a mounting system that preserves a fixed focal length. An example photograph can be seen in Figure 1.1a. As long as parallel lines and vanishing points could be found in a photo, Meydenbauer successfully used this technique to measure sizes, angles and distances on buildings and plot them in a ground plan. His work coined the term photogrammetry, and, a more modern term, metric photogrammetry, the measurement of three-dimensional geometry in metric units.



(a) Photogrammetry in the 19th century. The surveyor Albrecht Meydenbauer photographed the building with a pre-calibrated camera and could take metric measurements directly in the photo. Source: [Mey]

(b) A stereoscope projects two photographs from slightly different viewpoints into the left and right eye. This creates a sense of three dimensional form. Source: [Hol]

Figure 1.1: Measurements in images during the advent of photography.

The human visual system understands three-dimensional structure using visual cues stemming from the binocular vision of the eyes. Both eyes have slightly different viewpoints from each other, and the different appearances of objects up-close result in a sense of depth. In the beginning of the 20th century, stereo photographies imitating binocular vision were used as a form of entertainment, as viewed through an apparatus known as Stereoscope [Sil93]. A sterescope is depicted in Figure 1.1b. The viewing apparatus consisted of two lenses projecting each of the stereo photos into the viewer's according eye, creating a sense of depth. Stereoscopy exemplifies the idea of observing something from two or more different viewpoints and inferring scene depth from the observation disparities. This concept is critical for most forms of photogrammetry, and is referred to as Two-View or Multiview Stereo.

Over the courses of the 20th and 21st centuries, photogrammetry has remained a popular field of research as imaging and sensor technologies advanced. Photogrammetry allows measuring without having to make physical contact with an object - the underlying property for the field of Remote Sensing, in which photos and sensor recordings from airplanes or satellites are used to reconstruct the ground surface, oceans, atmospheric or radiation informations [WAH12] [RR99]. The same property is also exploited in the medical community, where patient information is recovered from computer tomography and x-ray sensor information. More closely related to photography, established techniques include rapid measurement of a patient's physique from a photo using mirrors to create multiple viewpoints [TM00], and analysis of a person's locomotion from a photographic

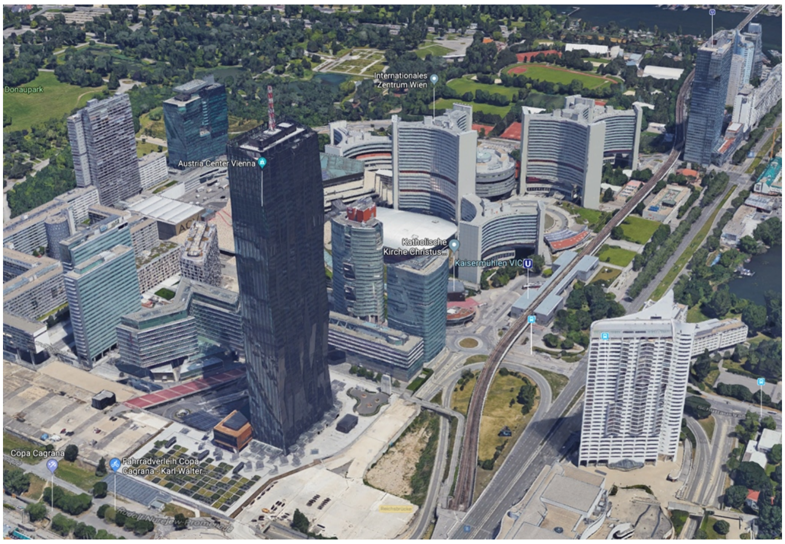


Figure 1.2: Large-scale photogrammetric reconstruction of buildings in Google Maps. Screenshot taken from Google Maps.

time series and/or movie [Cho09].

Evidently, the most driving factor for modern photogrammetry is the technological advancement of computer systems. Specifically, the advent of digital photography made imaging extremely cheap and widely available. Consumer-level integrated cameras are lightweight and small, while still producing high-quality photos to rival analogue technologies. Digital cameras can be made cheap and lithe enough to covertly fit on mobile phones and most other electronic devices, and robust and resistant enough to damage for deployment under severe circumstances, like on cars, underwater or in space. In surveying, commercially available drones fitted with digital cameras are employed in combination with GPS equipment to be able to create precise photo series from flights over terrain [RBN+11] or around large buildings [IKK+10]. An example of building reconstruction is shown in Figure 1.2. Various digital imaging equipment is also heavily employed in extraterrestrial surveying missions, such as NASA's Opportunity and Curiosity rovers capturing assorted photographic data from Mars for further photogrammetric processing [DXW+08], the efforts of which are currently ongoing.

Digital photography also makes physical storage of photos extremely cheap. In fact, widespread internet access and the availability of cloud storage and photography-tailored web applications allows one to collect practically infinitely many photos and share them with other people or the public. In 2011, Agarwal et al. exploited the massive number of publicly available photos of a tourist hotspot on Google Images and Flickr in their project 'Building rome in a day [AFS+11]. A screenshot of the project is shown in Figure 1.3. The challenge was to fully automate reconstruction steps that were historically done by hand, including the collection and sorting of photographs, identification of objects

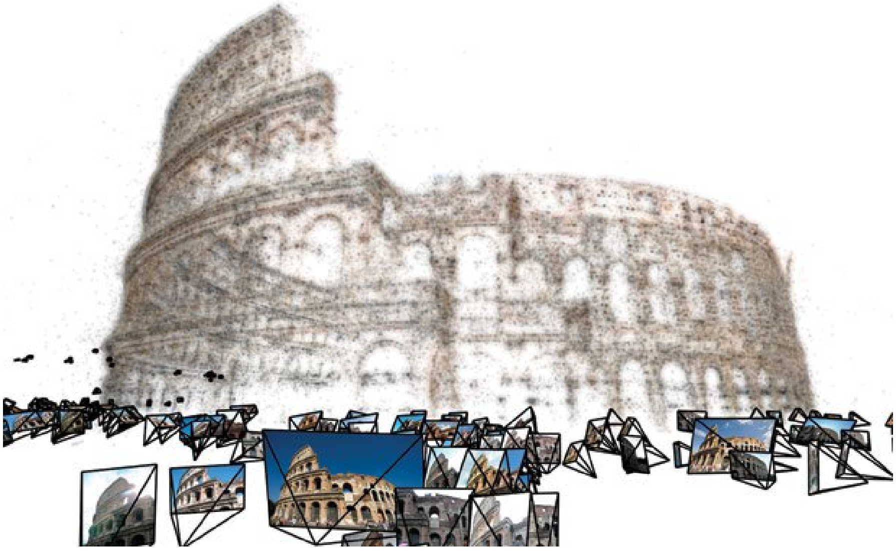


Figure 1.3: General-purpose photogrammetry used to reconstruct a part of Rome from public tourist photos. Image Source: [Sna]

in the images and taking of measurements. Although the particular goal of this project was the recovery of the 3D shape of architecturally complex buildings, the lack of prior information about the unsorted image set and the sheer heterogeneity of the cameras used makes it an important example of a general-purpose photogrammetry pipeline, a kind of photogrammetry that doesn't require specifically crafted and calibrated sensory equipment.

Nowadays, commercial photogrammetry applications are continuing to pervade most industries even remotely related to sensory data capturing and computer vision, especially where ordinary consumer cameras can be employed to profit off ever-decreasing cost and increasing speed and flexibility of the technology. A number of tools are available either as part of open source projects or to be licensed from commercial developers, notable examples including Agisoft PhotoScan [Agi], CapturingReality [Rea], and Autodesk ReCap [Aut]. Most of these tools are shipped alongside 3D modelling software, or at least 3D model viewers, allowing the user immediate processing of the reconstruction result, sparse or dense point clouds or 3D meshes, as an intuitive 'three-dimensional snapshot of reality'.

The bundle of photogrammetry and 3D editing software is usually tailored towards common use cases. For example, the Agisoft photogrammetry suite is often used in the terrestrial surveying and Geographic Information System (GIS) domains, especially in disciplines where flexibility and low cost are key, such as Unmanned Aerial Vehicle (UAV) based landscape surveying [RBN+11] or deformation monitoring at construction sites

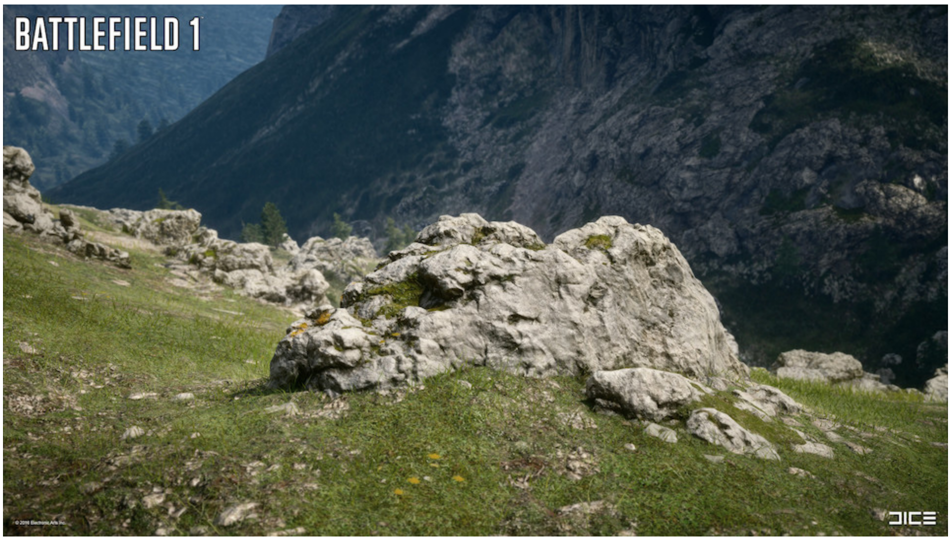


Figure 1.4: Natural scene in *Battlefield 1*. The assets were created using photogrammetric reconstruction. Image Source: [Sch]

[MH06]. Since the input data, photos of nature and human-made structures, can be interpreted by humans, the results lend themselves especially well for further processing in Building Information Modeling (BIM) [MR96], where the photos can be used to associate objects on maps and floor plans with semantic information. Research in this direction is currently ongoing with great interest [VSS14].

Further examples include, in the instances of Autodesk software and CapturingReality, creation of realistic content [S^+16] for movies and games [Rec], VFX art and interactive media [PHH15]. An example can be seen in Figure 1.4. Especially naturally rough and rugged structures with lots of distinguishing features, such as tree bark, rocks or mountains, are commonly digitised and artistically touched up with much success, creating media assets much more quickly and efficiently than by hand. In fact, the process has recently become established not only for final consumer assets, but also as support for preproduction and previsualisation. A similar ruggedness of structures also appears in archaeology, where photogrammetric scene modeling is employed as monitoring technique for necessarily destructive excavation and analysis [DSC^+03].

Almost all photogrammetry software is geared towards specific needs. This simple observation highlights the two conceptual challenges that fundamentally drive the entire science of photogrammetry.

Firstly, there is no perfect solution to handle all photogrammetry problems at the same time. A photogrammetry pipeline, as characterised by the calculations it implements, sets of free parameters, order of execution, preconditions, qualities, assumptions, restrictions, and so on, is always created to handle one reconstruction use case in particular, and the more it's specialised toward that one use case, the less applicable it becomes to entirely different scenarios. A reconstruction pipeline assuming ordered image sets is faster than reconstruction from unordered image collections, arbitrary cameras are handled differently from specialised and more accurate cameras, and photos of buildings are distinct from photos of trees or photos of human faces. Therefore, new photogrammetry pipelines will always need to be implemented as long as new use cases arise.

Secondly, implementing a general-purpose photogrammetry pipeline is relatively complicated. The math is rather involved: geometric constraints putting multiple hypothetical cameras in relation to each other from sets of observations require computations in higher dimensional space, and become ostensibly more ambiguous when prior information neither about the geometry nor about the cameras is available. In addition, reconstruction usually happens through processes of numeric solving and numeric optimisation of large systems of equations, calling for the utmost care and responsibility in implementation to avoid potentially bad convergence behaviour, imprecisions and extremely long runtimes. At least a modicum of programmer sophistication is required.

The implications make experimentation with photogrammetry in a scientific context considerably difficult. Implementing an entire photogrammetry pipeline from scratch only to investigate a novel use case or explore variations in methodologies and parameters isn't practically feasible. In addition, scientific evaluations often call for rapid prototyping, which is severely limited in confidence if the inner workings of photogrammetry software components are muddled by sketchy implementations or unknown in closed source software.

1.2 Recent Work

A number of scientific open source general-purpose implementations exist in the photogrammetry community. Many [RSN+14] tools cite the project Bundler Sfm by Snavely et al. [Sna06], which was first presented in 2006 and still serves as a reference implementation today. Bundler Sfm represents the contents of photos using so-called interest points, which are the locations of the most salient and visually striking features in a picture. The algorithm for finding these points, originally developed for the purpose of object recognition in photographs, is called Scale Invariant Feature Transform (SIFT) [Low99]. The underlying assumptions are that objects of interest can be adequately represented by enough such interest points, and that the 3D shape of the object can be reconstructed by looking at the motions of the interest points across multiple photos. The interest points also define Bundler Sfm's output, a point cloud representation of the object's surface.

VisualSFM by Wu [W+11] picks up this idea and implements the relatively time-consuming computations incrementally and on the GPU, allowing for significantly faster parallel solution of very large scale problems. This work embraces the potential payoff of using a huge number of photos for photogrammetric reconstruction, as the results become more precise the more evidence one can use to calculate them. The author reports a coherent reconstruction of Rome's inner city calculated from over 15000 photos in around one hour, which is orders of magnitude faster than a naive sequential implementation.

Further notable examples include Furukawa's Patch-based Multi View Stereo (PMVS) [FP10], which qualitatively expands the approach to produce dense 3D point clouds, allowing an object's complete and unbroken digitisation. Clustering Views for Multi View Stereo (CMVS) [Fur10], also by Furukawa, makes heterogeneous photo sets more manageable by introducing algorithms to identify cohesive clusters of images showing the same object, either automatically or guided by the user, and divide apart or join together such clusters during reconstruction depending on how much it improves the result. Very promising data representations different from interest points have also been proposed, for example in Salaün et al.'s LineSFM [SMM16], who use interest lines to successfully reconstruct office and building interiors with otherwise featureless white walls.

Most [Sze10] of modern photogrammetry cites the 2003 book 'Multiple View Geometry' by Hartley and Zisserman [HZ03], in which the mathematical background required for photogrammetric reconstruction is derived and concrete algorithms for individual reconstruction tasks are discussed. Much of the mathematical theory examined in this diploma thesis also appears in that book. Though they present the basic geometric and numeric understanding required to perform most photogrammetry tasks, the authors explicitly abstain from maintaining a complete, end-to-end photogrammetry pipeline in their book, emphasising that photogrammetry can be infinitely multifarious and there can be no one-size-fits-all solution.

A remarkable example of recent photogrammetric reconstruction is the 2011 project 'Building Rome in a Day' [AFS+11]. The authors, Agarwal, Furukawa, Snavely et al, fully embrace the emerging possibilities of modern technology, most of all the vast amount of public information as provided on the internet by users of social photo sharing sites. Hobby photographers tend to take pictures of the most interesting and salient features of an object, in this example the city of Rome and its buildings. The authors maintain to find over three million photos of Rome on Flickr, representing a complete photographic record of the city, the automatic processing of which is the main motivation of their work. The huge size of the input also poses the biggest challenge, requiring full automation of every reconstruction step. The authors employ computer vision, machine learning and information retrieval techniques on distributed systems using incremental updating to be able to scale to such orders of magnitude.

A large collection of more advanced photogrammetry algorithms tailored towards specialised use cases is given in 'Computer Vision: Algorithms and Applications' by Szeliski [Sze10]. The book concentrates on computer-based image understanding that goes beyond the mere localisation of interest points, such as image filtering and spectral transforms to express different aspects of an image, image segmentation and stitching to identify and localise expected contents in an image, photometry and computational photography to handle various kinds of sensors, 3D geometry representations, image recognition and so on. The algorithms in this book are often referred to in practical, production-quality software projects [MWA+13].

1.3 Aim of this Diploma Thesis

We observe the large number of very interesting photogrammetry approaches that could all ostensibly be employed to try solving a particular problem. However, there is no way of easily and quickly switching between various implementations, or even reusing existing implementations in explorative prototyping.

Almost all freely available photogrammetry libraries originated as research prototypes, and are fine-tuned to handle specific use cases. Accordingly, little attention is usually given to generalisation and reusability of software components, stability of the program and extensive documentation, since those aren't explicitly required for the scientific process. Further, most libraries use high-performance data structures that are optimised down to the lowest memory access level, oftentimes making alterations of the processed data difficult and structural optimisations, such as parallelisation, downright impossible. Central concepts, such as use of interest points or creation of dense point clouds, often govern the entire design of a photogrammetry pipeline, rarely making it possible to remove or replace such a component without a complete rewrite. Specialised implementations result in unwieldy 'black-box' programs, obfuscating or, in the case of closed source software, entirely omitting important implementation details.

In this diploma thesis, we present a composable, reusable photogrammetry library that emphasises clean, modular code. Programs are created through composition for rapid reasoning about, easy alteration and flexible reuse, and a functional programming style ensures logical soundness and robustness by design. Our library can be seen as a set of building blocks to create photogrammetry pipelines, aimed to alleviate the aforementioned problems and enable confident and rapid prototyping and experimentation with photogrammetry.

The following use cases are of particular interest to us:

- Given a photogrammetry pipeline describing images using specific features and descriptors, we would like to seamlessly substitute our own image description algorithm and easily switch between them. For example, we could use SIFT to identify objects in photos, but we also want to try out alternative interest point algorithms to see if they perform better, or we could use entirely different representations like interest lines instead. Our library achieves this by making no assumptions about image representation whatsoever. The data description itself is generic, and every function depending on it merely requires a minimal signature to be fulfilled, where the implementation is supplied by the library consumer. We provide a number of reference implementations for different interest points and object representation algorithms.
- A photogrammetry result's formation should be entirely transparent. For example, every point in a 3D point cloud should be fully accountable in the photogrammetry pipeline that produced it, from the localisation in the photos, over stable triangulation from recovered camera poses, to post processing in form of global optimisation.

This allows us to assess the quality of a result through statistical means, such as means and variances, by calculation and tracking in every reconstruction step. Our library supports this through a clean and minimal implementation following consistent and reasonable conventions, which sports no algorithmic peculiarities unless implemented by the consumer.

- Photogrammetric reconstruction inherently is an iterative process, which is why we would like to also support iteration on a pipeline level. For example, we might want to reconstruct from a set of, in terms of photogrammetry, well behaved photos first to obtain a stable initial guess, and then subsequently adjoin less stable, more ambiguous photos step by step to potentially increase detail. Our library is fundamentally built on composition of functionalities, enabling the consumer to compose, shuffle, repeat and switch between computation steps as desired.
- We would like to reorder computations, for example performing global optimisation either after every iteration or after all iterations, or both. Since all our functionalities are packaged in small modules free of interdependencies, any rearrangement compliant with the signatures is trivial.
- We want to swap out arbitrary computation steps with alternative implementations. Every implementation is valid as long as it fulfils the signature, but we provide additional support through readable and logical reference implementations in a syntactically clean language, in combination with this diploma thesis, which aims to serve as a technical documentation of the photogrammetry process.
- A given photogrammetry pipeline should support structural changes and optimisations. For example, long running computations should be easily parallelised. Another example would be the inclusion of user interaction, which could interrupt a calculation to adjust an intermediate result. Our library achieves this by applying pure functional programming and using immutable data structures, in this case meaning the following:
  - No algorithm or data structure implicitly assumes any precondition that isn't ensured on type level. No exceptions or memory management details are exposed by the library.
  - Every call to a function using the same parameters always returns the same result.
  - There is no mutation. Every computation creates a conceptual copy of the input. Every intermediate result can be accessed at all times. By extension, everything can be serialised easily.

We heavily utilise insights learned from functional programming for all the associated advantages. Functional programming fundamentally encourages immutability, composability and reusability, and the consumer needs not worry about maintaining program state as our library is stateless.

Ultimately, our library aims to facilitate rapid and confident experimentation with photogrammetry in a scientific context, while still allowing to produce production quality code. In this diploma thesis, we derive an abstract photogrammetry pipeline from first principles and provide reference implementations for each required computation step. The consumer is intended to see the reference implementation as a working basis, and should feel empowered to adapt it to specific needs. In the following, we discuss related work in the sections where appropriate, and illustrate findings through the examples of landscape surveying and building facade reconstruction.

The remainder of this diploma thesis is structured as such.

In Chapter 2, we begin by discussing the physical process of photography and the mathematical model of a camera, that is, the base input data on which we operate. We focus on the example of the common integrated consumer camera, which exhibits a number of physical properties, such as the imaging chip and lens optics, that we need to incorporate into our model in order to hypothesise about the 3D structure of the images it takes. We establish the notion and importance of the theoretically perfect Pinhole Camera, and the Internal and External Camera Parameters as a means of relationship to a real photo camera.

Chapter 3 concerns itself with data representation, which happens in an image by identifying a set of interesting items called Image Features. We discuss the mathematical processing of images using Image Filters, and exemplify their use in image understanding through the extraction of Interest Points, a number of implementations of which are employed to identify objects in images.

In Chapter 4, we discuss the central concept of Feature Matching, which is the identification of the same image feature across multiple photographs. Looking at a feature from multiple viewpoints is what allows us to reason about its three dimensional structure. Since photography inherently produces measurement errors due to the many real-world circumstances surrounding the act, a robust feature matching algorithm that can incorporate contextual information to produce as many correct matches as possible is paramount.

Chapter 5 introduces the geometry involved in the system of multiple cameras observing the same object. We discuss how arbitrary cameras are projected onto the pinhole camera, which yields us the Epipolar Geometry model, a system of properties that must hold true for multiple photographies of the same object. By estimating a geometric configuration that fulfils these properties, we obtain a local reconstruction of a part of a scene that is at most projectively ambiguous.

In Chapters 6 and 7, we combine many local reconstructions into one globally consistent 3D model, in a process referred to as Structure From Motion. We establish a traversal order over a set of photographs and iteratively recover a new geometric configuration that is consistent with all previously estimated ones, removing projective ambiguity and arriving at true Euclidean Reconstruction, including the set of camera locations for every image and the three dimensional triangulation of the image features used.

What follows in Chapter 8 is a global optimisation step that minimises the reconstruction errors accumulated through measurement inaccuracies in a least-squares sense, referred to as Bundle Adjustment. We touch on the strategies and implementations used to describe nonlinear projective geometry, and discuss the improvements yielded by, and limitations of, this approach.

Finally, Chapter 9 wraps up all functionality in a composable, reusable photogrammetry library. We discuss the function signatures required to introduce the minimal amount of information for each function to fulfil its purpose, the signatures also serving as sleek connection points for the purpose of composition. We outline the implementation for each of the concepts in the previous chapters and present exemplary applications thereof.

We round off the diploma thesis with a selection of results achieved with our reference implementations, and an outlook on the possible research and experimentation immediately supported by our library's flexibility.

Our implementation is built on Microsoft .NET and the visual computing platform Aardvark [Aara], and written in the language F#, which is also used in this diploma thesis.

CHAPTER 2

The Camera Model

Our goal is to reconstruct information about an object from a set of photographs. Usually, this information pertains to the three-dimensional structure of the object, meaning the outline, the surface, the material or the geometry comprising the object. Evidently, whatever qualities and properties our photographs exhibit are of fundamental consequence for every conceivable information or result that we infer.

Let us first consider the act of representing an object through a set of photographs, and along the way re-trace the general thought process and high-level structure leading to the basic photogrammetry process.

2.1 Pinhole Camera

The process of photography is capturing the appearance of a scene as viewed from one location. More precisely, the appearance of a scene is its two-dimensional projection onto a flat surface (sensor) placed in space at a particular position and orientation. This projection is called the *image* of the scene.

An apparatus that captures images is called a *camera*. Cameras can come in many different forms and functionalities, often characterised by their construction and complexity, the functionalities and qualities of their sensors, their cost, size, weight and ruggedness, the quality and content of their images, and so on. Examples for cameras are mobile phones, film cameras or the human eye.

The minimal possible camera is obtained from the *Pinhole Camera* model. The pinhole camera is shown in Figure 2.1.

The pinhole camera has a single point through which all light rays pass. This is called the *Focal Point* or *Camera Center*.

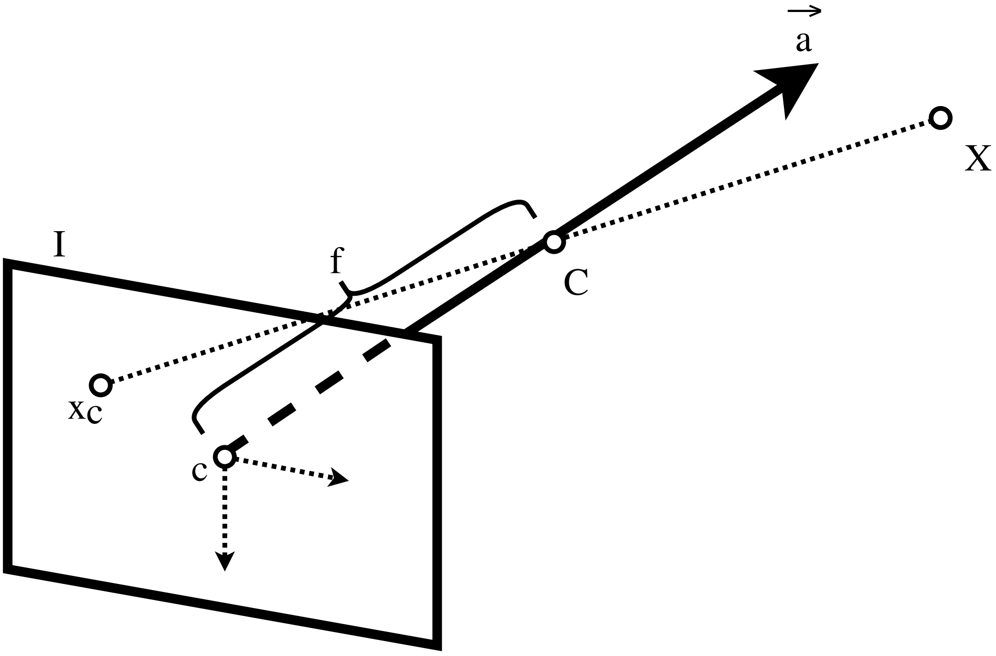


Figure 2.1: The pinhole camera model. The focal point C lies on the camera axis $\vec{a}$ at the distance of focal length f from the principal point c. Every point X is mapped to its image x_c on image plane I with the line going through C.

The camera is assigned a principal direction called the *Camera Axis*. This defines the focal point's relative *forward* and *backward* directions.

Behind the focal point lies the *Image Plane*. The incoming light rays are projected onto the image plane. The image of whatever lies in front of the focal point becomes visible on the image plane.

The distance between focal point and image plane is called the *Focal Length*. The bigger the focal length, the larger in size the image appears. This effect can be used for image magnification and is colloquially called 'zooming'. The focal length is inversely proportional to the camera's *Field of View*.

If the camera axis is perpendicular to the image plane, the image has no *skew*.

The point of intersection between camera axis and image plane is called the *Principal Point*. Interpreting the principal point as centre of the image allows localising all other image points relative to it.

A pinhole camera can be readily constructed by hand using a cardboard box. We make a small hole in one side of the closed box with a very thin needle. Then we create an opening on an adjacent side large enough to see through, and drape a cloth over it to shield it from any unwanted light. Looking through the opening into the dark inside, we

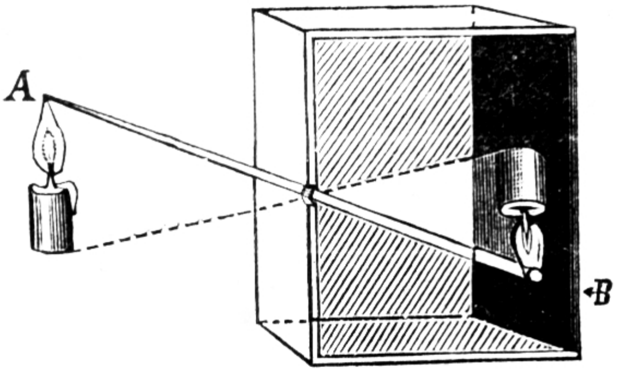


Figure 2.2: A Camera obscura. Source: [191]

will be able to see an image of the outside world on the side opposed to the pinhole. The pinhole is the focal point and the cardboard wall is the image plane of our handmade camera. This apparatus is called 'Camera Obscura', illustrated in Figure 2.2.

This construction is easy, but effectively demonstrates the considerable shortcomings of this minimal camera model, as imposed by the vast variety of physical properties appearing in the real world. The image is fairly dark because the tiny pinhole lets through only very little light. The image gets darker with increasing distance from the principal point, since the thickness of the cardboard opening absorbs light rays coming in at steeper angles. The image is also blurry due to both diffraction effects as well as the pinhole being larger than an impossibly perfect point. And so on.

2.2 Camera Model

Various imaging techniques and challenges arising from practical requirements are the prime reasons for the very high level of sophistication of modern consumer cameras. In fact, no camera can perform all conceivable imaging tasks perfectly, which is why many specialised gadgets and constructions exist, such as zoom objectives, wide-angle lenses, sunlight filters, anti-reflectance surface coatings, night vision cameras, et cetera.

Nevertheless, reducing complex cameras to the minimal pinhole camera model is very desirable for performing large-scale computations. In fact, the reduction is mathematically unavoidable for non-trivial cases.

Most modern cameras employ a lens system comprised of multiple lenses mounted within an objective. Such a lens system is illustrated in Figure 2.3.

The frontmost lens usually is convex, parabolic or spherical, such as to collect as much of the incoming light as possible regardless of incidence angle. A system of several lenses usually follows, purposed to make the rays move the distance and hit the sensor in parallel so as to lose as little intensity as possible to the surface of the casing.

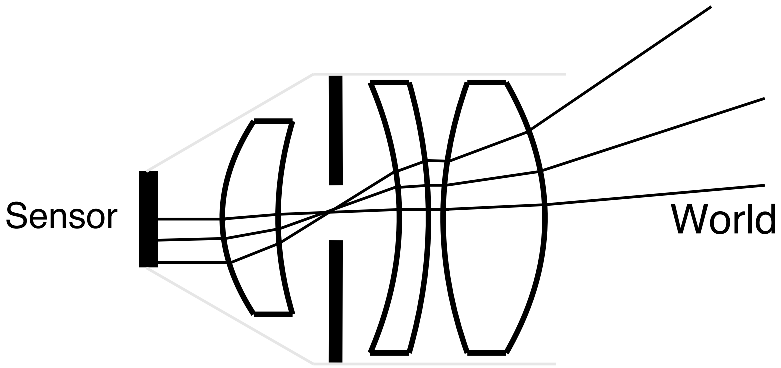


Figure 2.3: Symbolic structure of a camera objective. The lens system collects as much light from the world as possible, and makes the rays pass through the aperture to hit the sensor in parallel.

The lens system directs the light rays through the focal point at extremely high precision, avoiding blurriness resulting from angular deviation of light rays. The lenses of zoom objectives are also mounted on retractable rails, making it possible to adjust the focal length by hand.

The focal point is located at the centre of a ring-shaped shutter comprised of a number of rapidly moving fins. This is called the *aperture*. The opening diameter of the aperture, referred to by the parameter *f-stop*, can be adjusted. A larger aperture lets through more light, but introduces more aberrant light rays into the sensor and reduces the range at which objects appear sharp. The sharpest point is the *Focus*, and the blurring effect around it is called *Depth of Field*. While this effect is technically speaking an imaging artefact, it's often used as a stylistic device in artistic photography.

The brightness of a photo is controlled in two ways, by adjusting the light sensitivity of the sensor, the parameter for which is called *ISO Value*, and through the aperture's *length of exposure*. These two parameters complement each other: too high an ISO value makes images much too bright, while too long an exposure blurs the image from the movement of the camera if held in hand.

A wide variety of additional measures are commercially available to further alter image quality. Notably, these are physical filters adjusting the incoming light in various ways, special lens coatings reducing undesirable effects of sunlight or lens flares caused by interreflection of light within the objective, a swathe of digital processing techniques from artistic filters to HDR photography, and so on.

Almost all of these, however, only correlate to the perceived quality or impression of an image, but have no influence on the geometric properties of an object's appearance. In fact, the geometry of an image depends in the largest part on the physical path the light travels until it hits the sensor. This trait is conceptually the same for all commercially prevalent cameras. Accordingly, it, and the interrelated deviation in image geometry, is characterised by a stable set of parameters, called the *Camera Parameters*.

2.3 Camera Parameters

Camera Parameters describe a camera's geometric characteristics when taking an image. The significance for photogrammetry is twofold.

Firstly, camera parameters fully define the projection function that maps every point in 3D Euclidean space onto its 2D likeness in an image.

Secondly, we deliberate on disparities and similarities found in multiple images of the same object using their corresponding camera parameters.

We differentiate between two types of camera parameters, *External* and *Internal*.

2.3.1 External Camera Parameters

External parameters construe the camera's localisation within the three dimensional world.

- **Location**: Three components `X,Y,Z` representing the position of the camera centre in the 3D world. The measure is relative to a particular reference coordinate system defined beforehand.
- **Orientation**: The direction in which the camera looks, specifically defined by the orientation of its camera axis and the perpendicular upward direction. It also refers to the previously chosen coordinate system.

2.3.2 Internal Camera Parameters

The path light travels within the body of the camera until it is measured by the sensor is inferred from the internal parameters.

- **Focal Length**: Most cameras' focal length can be adjusted manually, hence the knowledge of this parameter's significance.

  The focal length is the distance between focal point and image plane. The larger the focal length, the more the image is magnified and the smaller the camera's opening angle.

  Images are rarely quadratic. Most sensors have higher extents or resolutions in one dimension, engendering different focal lengths in the horizontal and vertical directions in 2D image space. The focal length is thus measured in two parameters `fx, fy`.

- **Principal Point**: The 2D coordinate `cx, cy` of the intersection between optical axis and image plane within an image is called the principal point. All other image points' 2D locations are in relation to this point.

  This should usually be the exact center of the image. However, in practice, we sometimes want to use only a cropped part of an image containing the object of interest. The principal point lies somewhere else in such a case.

  What's more, the principal point is usually very slightly offset from the true optical axis, stemming from mechanical inaccuracies and the numeric integration on the sensor. This minor deviation may cause up to several pixels in shift on observed objects, depending on their distance to the camera. Therefore, knowledge of this parameter is undoubtedly paramount for a precise result.

- **Skew**: Some cameras exhibit shearing in their images due to some digitisation processes. This is represented by the optical axis skew parameter `s`. The value is simply zero for most cameras, unless the image has been cropped or otherwise processed.

- **Lens Distortion**: Photography employs systems of optical lenses sporting curved surfaces. These various kinds of surface curvatures precede a whole slew of related imaging artefacts.

  Lens distortion effects may be utilised on purpose. For example, wide-angle lenses bear very strong curvature, bending the image significantly along the outer edge such as to achieve an extreme field of view.

  Mechanical imprecisions introduce lens distortion artefacts in practically all consumer cameras. Notably, especially cheap and compact devices such as mobile phone cameras evidence these effects often. Moreover, the fact that we take rectangular images using round lenses causes the aberrations to be the strongest in the corners and along the edges of a photo.

  Lens distortion is generally a property of the camera objective, and remains the same until the objective is modified or exchanged.

  It usually suffices to represent all lens distortion effects using *Brown's distortion model* [WCH+92]. This function models common radial distortion effects apparent in photography, as illustrated in Figure 2.4. It is specified by three radial distortion coefficients `K1, K2, K3`, and two tangential coefficients `P1, P2`. In the **x** dimension, the relation between distorted image point x', true image point x and principal point c is given by

  $$x' = x + (x - c_x)(K_1 r^2 + K_2 r^4 + K_3 r^6) + \left[P_1(r^2 + 2(x - c_x)) + 2P_2(x - c_x)(y - c_y)\right]$$

  where $r = \sqrt{(x - c_x)^2 + (y - c_y)^2}$. The formula is analogous in the **y** dimension. It approximates an image point's deviation from the image centre using the first few powers in distance of the exact physical distortion model.

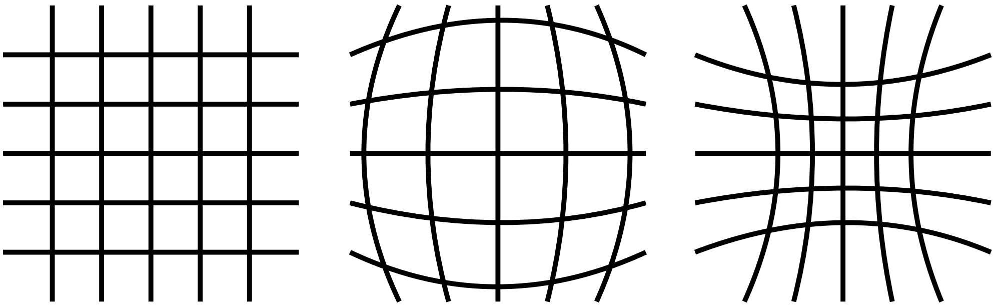


Figure 2.4: Common lens distortion effects in consumer cameras. *Left*: no distortion, *centre*: barrel distortion, *right*: pincushion distortion.

2.3.3 Photo Undistortion

Lens Distortion causes images to bulge outwards from or inwards to the centre. Straight lines appear curved. Based on the strength of the effect, the curvature may cause divergence of up to several tens of pixels in a typical photo. The importance of this property for photogrammetry is clear.

Brown's distortion formula allows us to incorporate lens distortion effects into our camera model. However, actually computing the effects on the fly would introduce numerical instability and is rarely practicable.

In practice, we want to do it the other way around. After obtaining the distortion parameters, the camera distortion is inverted and the inverse distortion is applied to every pixel of the image. This process precisely rectifies lens distortion artefacts in a photo. Straight lines appear straight and objects look as if the lens had been perfectly flat.

We call this *Image Undistortion*, and a rectified photo is an *Undistorted Image*. An example for photo undistortion can be seen in Figure 2.5.

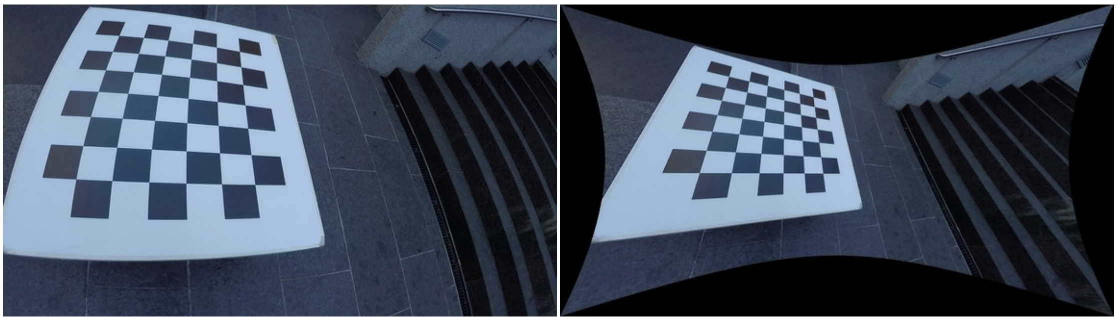


Figure 2.5: Photo undistortion for a fisheye lens. *Left*: original photo. *Right*: undistorted image, in which straight lines appear straight. Source: The PennCOSYVIO Data Set [Pfr16].

2.3.4 Camera Matrix

Camera parameters allow us to define the projection function, which is the projective mapping of every point in the three dimensional world onto the two dimensional image coordinates as observed by one particular camera in one particular location. If we can saturate a camera's projection function with all camera parameters, we call the camera *calibrated*.

We refer to the combination of one camera at one specific location and orientation, commensurably a full set of camera parameters, as *Shot*.

Image undistortion eliminates the parameters related to lens distortion from the projective equation. The projection function for an undistorted image can be written as a linear equation. We call the matrix representing this function the *Camera Matrix*.

The camera matrix `P` looks like in the following Equation 2.1. Multiplication of this matrix with the homogeneous coordinates of a 3D point followed by perspective division (see Chapter 5) results in the point's 2D image.

$$\mathbf{P} = \underbrace{\begin{pmatrix} 1 & 0 & c_x \\ 0 & 1 & c_y \\ 0 & 0 & 1 \end{pmatrix}}_{\text{principal point}} \times \underbrace{\begin{pmatrix} f_x & 0 & 0 \\ 0 & f_y & 0 \\ 0 & 0 & 1 \end{pmatrix}}_{\text{focal length}} \times \underbrace{\begin{pmatrix} 1 & s & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}}_{\text{shear}} \times \underbrace{\mathbf{t}}_{\text{3D translation}} \times \underbrace{\mathbf{R}}_{\text{3D rotation}} \tag{2.1}$$

We usually decompose the camera matrix into two parts, the internal (principal point, focal length and shear) and external (3D translation, 3D rotation) camera matrices. To improve numeric stability, we usually impose some restrictions when taking pictures for photogrammetry, ensuring that the internal matrix remains mostly constant and only the external part varies. These restrictions are discussed in Chapter 7.

The pinhole camera has an internal matrix equal to identity. The focal length is one unit long, and the image plane is a square of two units in side length. The principal point is at the exact centre. There is no skew. Correspondingly, the field of view is 90 degrees.

The reference camera can only be moved around and changed in orientation, but has no other degrees of freedom. Every camera can be reduced to a pinhole camera by multiplying the inverse of its internal matrix onto its projection function. The advantages of identifying the pinhole camera are twofold.

Firstly, the pinhole camera has only the aforementioned six degrees of freedom. Keeping the set of variables small yields significant improvements in convergence of numerical optimisations.

Secondly, being able to reduce all cameras to a reference camera also allows us to translate cameras between each other. We may hypothesise about the appearance of one camera's image in a different camera.

This is the exact condition permitting many heterogeneous cameras to coexist within the same frame of reference. We gain the ability to reconstruct objects in a very general fashion, given only a large enough set of photos and knowing a sufficient number of camera parameters for each.

In terms of code, we represent a camera with the type given in the following Code Listing 2.1

Listing 2.1: Camera type and basic functions.

```
type Camera =
    {
        position    : V3d
        forward     : V3d
        up          : V3d
        right       : V3d
        focal       : V2d
        pp          : V2d
        skew        : float
    }

let Identity =
    {
        position= V3d.OOO
        forward = -V3d.OOI
        up      = V3d.OIO
        right   = V3d.IOO
        focal   = V2d.II
        pp      = V2d.OO
        skew    = 0.0
    }
```

where `V2d` and `V3d` are two and three dimensional vectors, and transformations are applied with

```
let translated trans cam =
    { cam with position = cam.position + trans }

let rotated rot cam =
    { cam with
        forward = rot * cam.forward
        up = rot * cam.up
        right = rot * cam.right }
```

Filling the parameters into the camera matrix lets us project a 3D point onto the camera's image plane

```
let matrix cam = ... //equation (2.1)

let project x cam =
    let view = (cam |> matrix) * x
    view.XY / view.Z
```

This function shall serve as a brief introduction to the F# syntax. The syntax of the expression above is read like this:

The function `project` with arguments `x` and `cam` is defined as follows. The camera `cam` is piped as argument into the function `matrix`, and the resulting camera matrix is multiplied onto the point `x`. The resulting value is bound to the name `view`. The overall result is obtained by dividing the `XY` components of `view` by the `Z` component.

CHAPTER 3

Image Features

Reconstructing objects from photographs necessitates an appropriate representation of objects' appearances.

Looking at an image and identifying the objects it depicts is easy for us humans. The human conscience is capable of recognising things by context, which is to say, semantic information derived from the form and qualities of an object beyond its mere appearance. Even further, we can readily pinpoint the same objects in multiple different photos. A walk around an object poses no problem to us even though the appearance might come out extremely different at opposing viewpoints.

Yet how would one program a machine to exhibit a similar degree of understanding? The task is far from trivial. Although computational complexity is always going to be part of the challenge, even asking the right questions is a major investigation on its own. How do we encapsulate an object's appearance in a generally applicable set of parameters? What makes something visually outstanding? How do we handle imaging artefacts? Additionally, the appearances of objects perceivably change due to practical circumstances such as time of day, weather, illumination conditions, partial occlusion, passage of time, and so on.

A digital photograph is represented by a matrix of pixels. Any object we want to recognise must inevitably appear in subsets of these pixels. Comparing pixels with each other directly would be a trivial experiment. A red circle would remain a red circle in all images, right?

In the general case, however, this approach falls flat pretty quickly. Direct pixel comparison can suffice for pre-specified objects in stable conditions, such as special markers in a laboratory. But in a natural environment, if we find a number of brown pixels, we won't be able to tell whether they belong to the earth, a tree bark or a wooden door.

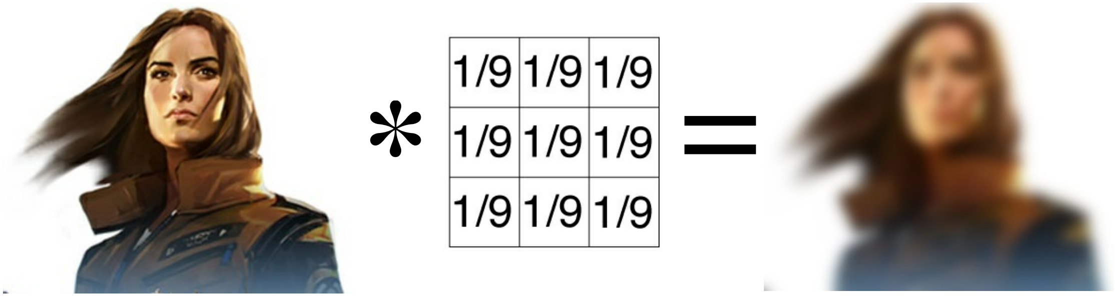


Figure 3.1: An example for a blur filter. Each pixel is averaged with its neighbours, using weightings given in the filter kernel.

3.1 Image Filters

The idea becomes apparent to observe the surroundings of a pixel in addition to the pixel itself. This harkens back to the idea of context in the understanding of semantics. Variations in the pixel neighbourhood of a certain shape and size gives testimony to the local context in the area, for instance if the pixel is part of an edge, a specific shape or a certain texture.

This is a fundamental technique used in Computer Vision in determining the contents of an image. Iteration over every pixel and aggregation of the local pixel neighbourhood is referred to as *Image Filtering*. The aggregation weighting function is called the *Filter*, and the output is the *Filtered Image*.

For example, choosing a patch of tree bark as filter will produce the highest values for pixels which are part of the tree. We can localise trees in a photo in such a way.

Image Filtering is a very powerful and versatile model of computation since almost all image operations can be expressed in terms of filters, including the ones we use in photogrammetry. Pre-applying generic filters to image sets often makes sense to get rid of image noise or enhance specific features. Specifically, this includes common imaging artefacts:

- Image noise from low-light conditions or high sensor light sensitivity can be reduced using gaussian filters. An example for such a blur filter can be seen in Figure 3.1.
- Blurry images resulting from camera movement or long exposure times can be enhanced using sharpening filters.
- Different dynamic ranges due to varying exposure times and f-stops are rectified using histogram equalisation.
- Shifts in hue and intensity that come with special optical filters and lens coatings have according normalisation filters.
- Image undistortion can also be interpreted as image filtering.

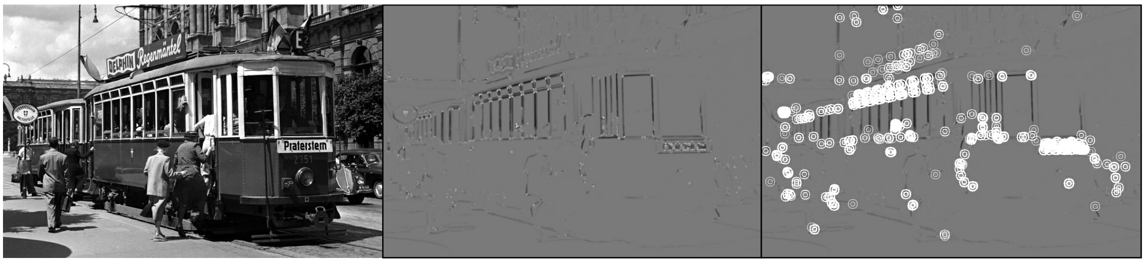


Figure 3.2: Corner detection. *Left*: Original image, *centre*: filtered image using corner detector, *right*: Corners (white circles) are the filter's strongest responses.

3.2 Interest Points

Objects typically consist of large amounts of locally meaningful pixels situated within local regions, which we will refer to as Points. Intuitively however, most of these points describe the same thing, especially when they lie right next to each other. Every point on a flat, featureless surface looks the same and can't be distinguished from the others. Consequently, not every point in an image is useful for photogrammetry.

In fact, photogrammetry can be viewed as an Image Classification or Machine Learning problem. Those disciplines teach that indiscriminate inclusion of information into a learning system actually worsens classification performance rather than improving it, introducing uncertainty and muddying outcome class boundaries. More often than not, the challenge is finding the minimal possible set of features to represent the most significant information present in an image.

We call this smallest set of most salient image points *Keypoints* or *Interest Points.* Interest points belong to the category of *Image Features.*

Keypoints are typically object corners, or points with similarly high visual impact. They are capable of adequately representing the form of an object to a certain degree, depending on how well the form is visible through structure and detail. For example, most algorithms will densely cover a photo of a ragged mountain with keypoints, but will find zero keypoints on a white wall. An example for keypoint detection using image filters is shown in Figure 3.2.

Notably, when the camera moves, the keypoints in an image exhibit an according motion in the other direction. Accordingly, the projective transformation of an object resulting from a change in perspective can be ascertained by analysing the collective motion of a set of independent points across several shots.

We refer to the identification of the same keypoint in multiple images as *Feature Matching* (discussed in Chapter 4).

Both the extraction and the matching of image features within a set of photos are paramount for photogrammetry, hence the special attention given to both topics.

Note that we limit ourselves to keypoints here since they perform adequately well in the general case. Also, fairly sophisticated implementations exist. However, more specific problems may suggest using entirely different features, such as lines in aerial photos or planar polygons on the subject of houses, potentially increasing the quality of the result. We further discuss the possibility of different features and the ease of resultant extension conferred by our library's composability and general applicability in Chapter 9.

3.3 Feature Extraction

Interest point features serve two distinct purposes.

Firstly, they localise a point of interest within an image, as purveyed by the point's visual fidelity and salience. A point is generally found at corners and edges of objects, in regions of geometric intersection and similar structural complexity, as well as fine embellishments and details.

Usually a keypoint is also associated with a size to indicate the feature's scale relative to the size of the image. This way, the thousands of features in a close-up of a house retain an association to the fewer keypoints in a photo from a distance.

Secondly, keypoints provide distinct descriptors for identification based on its visual appearance. The descriptor is derived from a keypoint's local surroundings' striking characteristics, and serves to cluster points together that look similar.

Most keypoint features sport a fixed length byte array representing a well defined set of pixel values sampled in a specific way. This is called the *descriptor*. In addition, a measure of scale and principal orientation is usually provided to contextualise a possible affine transition from camera movement.

Further desirable properties are robustness with respect to global illumination change, as well as tolerance of local geometric distortions and blurriness.

Several state of the art implementations experience widespread use and are of interest to us.

3.3.1 SIFT

Scale Invariant Feature Transform (SIFT) is a patented feature detector and descriptor framework first published by David Lowe in 1999 [Low99]. Due to its speed and robustness, SIFT is often viewed as a reference implementation of keypoint localisation and has inspired many algorithms over the years (for example the one described in Section 3.3.2.

Keypoint localisation in SIFT works using a Difference of Gaussians (DoG) approach. The image is blurred at varying strengths, efficiently achieved through convolution with Gaussian filters of different variances. The Gaussian kernel suppresses information that is higher in spatial frequency than its own extent, and the difference between two filtered images retains only the frequencies in between. If a point appears in many frequencies, it must have some degree of visual salience and is declared a keypoint.

After enough keypoints have been found, they are assigned SIFT descriptors by inspection of their local pixel neighbourhoods. Keypoints may be shifted towards regions of high contrast to align them with structures in the image. The local neighbourhoods are processed at multiple sizes and rotations to assign them a principal scale and orientation, in addition to a histogram ’fingerprint’ of the neighbouring pixels’ intensities. In combination, this descriptor makes it possible to assign a distinct identifier to a keypoint, regardless how far away and at what angle the photo was taken.

The original SIFT publication goes on to describe an entire framework for using these features in object recognition and image understanding tasks. However, much of the recognition goes to the features themselves, which are often used as a first stepping stone in computer vision tasks, for example by the OpenCV implementation [Oped].

SIFT features are patented. An example from the OpenCV implementation can be seen in Figure 3.3, *top right*.

3.3.2 SURF

Inspired by SIFT, Speeded Up Robust Features (SURF) is a patented feature detector and descriptor, first presented by Bay et al. in 2006 [BETVG08], and is claimed to be up to several times faster and somewhat more robust.

Keypoint detection in SURF follows a similar principle as SIFT, albeit with a more efficient implementation. By using square gaussians together with integral images for spectral analysis, the image can be processed very quickly using only few integer operations. Keypoint localisation happens efficiently by examining the Hessian, the second order derivative and measure of curvature, in an image region.

Keypoints are described using Haar wavelets, a family of orthonormal functions suitable as basis for wavelet transforms, which are the simplest possible wavelets and have an accordingly simple computational representation. By expressing the image signal in spectral space, principal orientation and scale can be easily obtained from the shape of the wavelet transform.

SURF is also implemented in OpenCV [Opee]. Again, SURF is patented. An example from the OpenCV implementation is shown in Figure 3.3, *bottom right*.

3.3.3 ORB

Oriented FAST and rotated BRIEF (ORB) is a keypoint localisation and description algorithm presented by Rublee et al. in 2011 [RRKB11]. It specialises on extremely fast performance suitable for application in mobile phones and low-power situations.

To localise keypoints, it employs the FAST corner detector [RD06]. The local neighbourhoods of pixels are represented as an ordered sequence of values, which are checked for certain patterns that must exist for the pixel to be considered a corner. Then, all candidates are checked for patterns which can not exist in corners to eliminate false positives. The patterns are established through machine learning.

Using these patterns, ORB orients the usually fixed-direction BRIEF descriptors [CLSF10], which are a byte string of pixel intensities and intensity differences in the local neighbourhood. This approach is reported to produce similar results as SIFT and SURF, but much more quickly.

The OpenCV implementation can be found at [Opeh]. ORB is free, and we prefer to use it in lieu of SIFT and SURF in our implementation. An example is shown in Figure 3.3, *middle left*.

3.3.4 BRISK

Binary Robust Invariant Scalabe Keypoints (BRISK) was published by Leutenegger et al. in 2011 [LCS11] abandons the search for specifically defined corner points in favor of a texton approach, that is, letting keypoints describe themselves through the 'textureness' [MBSL99] of their appearance.

After detecting keypoints in a fashion similar to FAST, a regular sampling technique is applied to extract a specifically layouted bit string representing the inner structures of enclosed shapes, through which repeating patterns and uniform textures become apparent. Scale invariance is achieved through multi-scale analysis.

An implementation of BRISK is part of OpenCV [Opeb], and an example is shown in Figure 3.3, *middle right*. BRISK is free.

3.3.5 AKAZE

In their work published relatively recently, in 2013, Alcantarilla et al. present their algorithm called AKAZE [AS11] for detecting and describing keypoints. AKAZE aims to outperform SIFT and SURF in terms of accuracy by performing spectral analysis in a nonlinear scale space that is adaptive to the actual contents of the image, which is considerably more sophisticated than simply blurring the image the same amount everywhere.

The nonlinear scale space is constructed using an approach based on Fast Explicit Diffusion [GWB10], a diffusion simulation technique that can be applied to image convolution in cases of inhomogeneity or discontinuity. Embedded into a multi-scale scheme, image smoothing is performed, through analysis of the Hessians, only inside the bounds of shapes in order to preserve hard contours for the subsequent feature detection.

AKAZE's feature descriptors extend this focus on edge preservation. They adapt a texton approach based on binary feature descriptors to respect boundaries by having individual samples work on area averages instead of single points, which introduces gradient information into the descriptor in addition to the intensities. Scale and orientation of a feature is carried over from the nonlinear scale space representation.

AKAZE is reported to outperform other keypoint algorithms for pictures of buildings, edges and human made structures. An implementation is included in OpenCV [Opea]. An example can be seen in Figure 3.3, *bottom left*. AKAZE is also free.

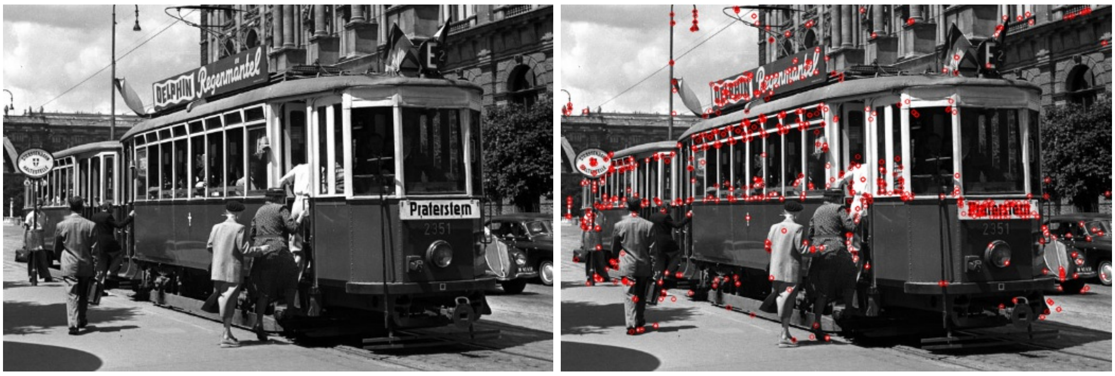
3.3.6 On Other Features

The implementation developed in the scope of this diploma thesis doesn't include any structurally different features, such as lines or shapes. However, our photogrammetry library is not bound to any specific feature detector. In fact, a library consumer must only supply different implementations for the types of features, shown in the following Section 3.3.7, suitable feature matching, Chapter 4, and projection into a camera, Section 5.4. Due to our library's compositional nature, these implementations are brought into effect through simple function substitution, as shown in Chapter 9.

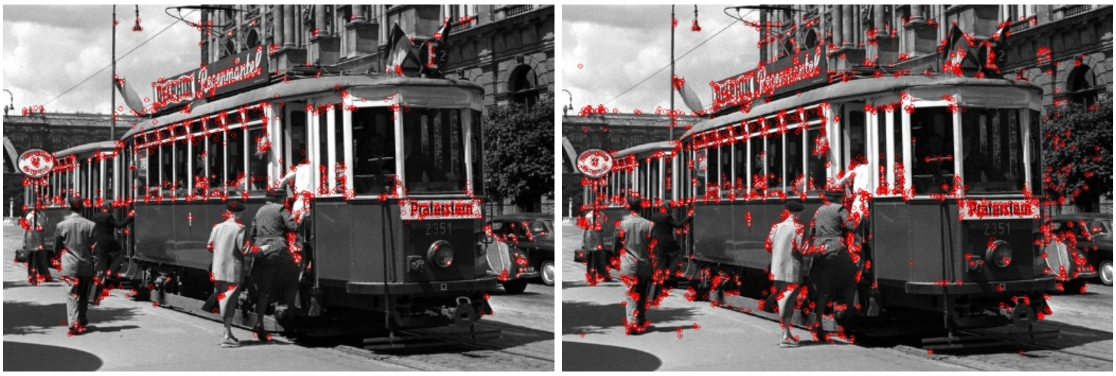
3.3.7 Application

Our photogrammetry library allows the use of arbitrary image features to represent photos. Specifically, different image feature algorithms can be swapped out easily, and features can be combined or used in sequence to exploit emergent synergies between the various specialisations. Our library merely requires the consumer to supply a type signature, which can be referred to in subsequent steps of a pipeline.

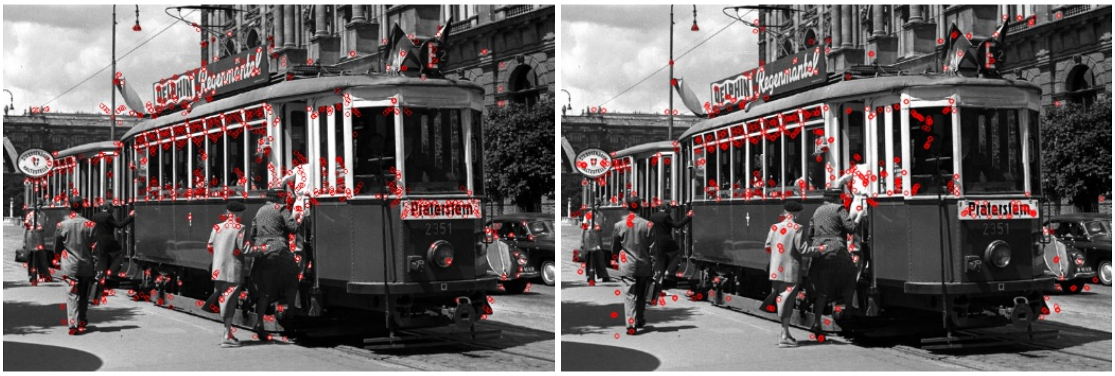
In our implementation, we use a common type signature for the aforementioned image features consisting of a location and scale in normalised device coordinates, orientation in degrees relative to a principal direction, and a feature descriptor as a byte array. We use the features' respective OpenCV implementations for the actual algorithms.The type signature we use can be seen in the following Code Listing 3.1



Original Image SIFT



ORB BRISK



AKAZE SURF

Figure 3.3: Different feature extraction algorithms applied to the same image. Features are visualised as red circles. The parameters were chosen manually to produce about 500 keypoints.

Listing 3.1: Type representing features.

```
type Feature<'a,'b> =
    {
        location : 'a
        description : 'b
    }

//compute the distance between two descriptions
val featureDistance : 'b -> 'b -> float

type OpenCVDescriptor =
    {
        angle : float
        scale : float
        intensity : float
        descriptor : float[]
    }

type OpenCVFeature = Feature<V2d, OpenCVDescriptor>
```

Feature detection is done by a function of type `PixImage -> list<Feature>`, where `PixImage` is an image file loaded from disk,

```
type FeatureKind =
| ORB  of int
| BRISK
| AKAZE
| SIFT of SiftParameters
| SURF of SurfParameters

let detect kind img =
    match kind with
    | ORB maxcount -> OpenCV.detect(img,"orb",maxcount,...)
    | SURF -> cv2.nonfree.SURF(...)
    ...

let detectOrb = detect (ORB 500)
let detectAkaze = detect AKAZE
let detectBrisk = detect BRISK
```

CHAPTER 4

Feature Matching

Given two sets of image features extracted from two images, the goal is to find feature pairs spanning both sets which depict the same three dimensional world point.

Identifying the 'same' point in both images is a term laden heavily with meaning. The human observer may easily identify parts of images as the same thing, even though they might appear vastly different. For example, photos from different sides of a house will obviously depict the same house for us. We don't care wether the change in perspective distorts the picture, or the location of the sun muddles the colours. Corners and edges outlining the geometry are still related in a logical and coherent fashion.

On the other hand, looking at cropped portions around some image features, tiny snippets of an image instead of the whole, makes piecing together correspondences difficult even for a human. An individual image point looks way too similar to the thousands of others, and could be either one of many candidates depending on how we look at it. An example can be seen in Figure 4.1.

We observe that feature matching is a global operation requiring information across the whole image, rather than an isolated per-feature process. The idea of embracing context holds firm.

4.1 Baseline

The *baseline* is the distance between camera centres.

It is proportional to the depth information we can recover from their images.

A small baseline means that the posture of the camera barely changed between shots. Consequently, the images look almost the same, exhibiting only small changes in perspective. Feature matches are extremely high in number, but each feature match only contributes a very small amount of novel information.

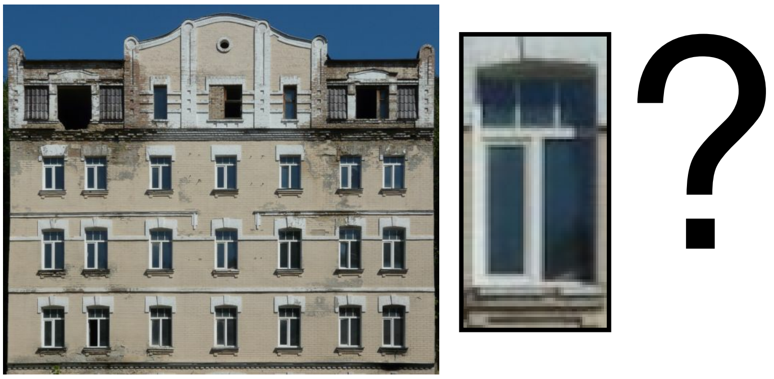


Figure 4.1: Ambiguity in feature matching. The local image patch on the right could be one of many in the left photo. Image Source: [Chu13]

Narrow baseline reconstruction is typically employed when a massive number of images is available, such as movies and dense image sequences, as well as public photo collections on the internet.

Typically, such reconstruction can afford to discard features and images that appear unstable due to the sheer number of available alternatives. Oftentimes, a stable initial solution is constructed using the most reliable data points, and subsequently the solution is refined and made more dense by introduction of the remaining information.

The large amount of information poses its own unique challenges. Systemic inaccuracies can accumulate to extreme degrees over a large number of numeric iterations, which can be likewise difficult to detect and eliminate due to the potentially very long calculation time.

Furthermore, a number of pictures in a huge photo set may contain only redundant information. Nothing new can be inferred from (partial) copies of existing images. Failure to cull such superfluous data points merely repeats known observations, increasing numeric and computational complexity without any real gain.

Wide baseline reconstruction happens when the distance between cameras is large.

Feature matches typically become very few because the stark change in viewpoint completely alters the features' visual appearances. Coincidently, each feature match has a very high value in terms of information gain, since every picture introduces almost exclusively novel observations.

The width of baseline is a relative term. The baseline must always be narrow enough to actually see the object we want to reconstruct in sufficient detail, which may be different from object to object. For example, lots of photos taken in tiny steps are required to reconstruct the corner of a house, all the while we can take wide strides along the facade.

Extremely wide baselines usually occur when only few photos are available spanning a lot of space, and re-taking additional photos is difficult or impossible. Adorning shots

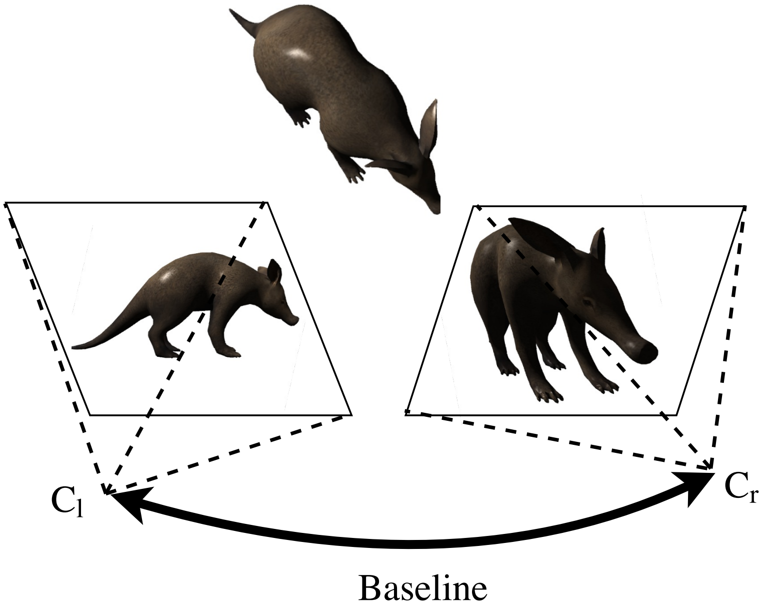


Figure 4.2: The baseline is the difference between camera centres C_l and C_r.

with additional knowledge typically improves such cases. For example, geo-referenced cameras can provide initial estimates for camera posture, while special markers with distinct geometries can establish three dimensional topology.

4.2 Camera Motion

A camera's difference in location and orientation between shots is called *Camera Motion*. The inverse movement can be observed on matching features across photos, in terms of pixel coordinates. We refer to that as *Feature Motion*.

When taking pictures specifically for photogrammetry, the manual choice of camera motion is a sensible task. The photographer must move in a way to capture as much of the desired information as possible.

This is rarely a task that can be executed mechanically, requiring a certain degree of experience and finesse on the photographer's part. It usually takes a few failed attempts to get a feel for the appropriate camera motion to observe an object adequately.

In practice, the photographer tries to think in terms of depth. A narrow baseline is required where the depth range is large, meaning that both near and far away objects are on the image. Typical scenarios are wall corners and streets, where we look down a projective surface. An example is shown in Figure 4.3. Conversely, a wide baseline is sufficient when the range in depth is small, such as looking straight onto a wall or a fence.

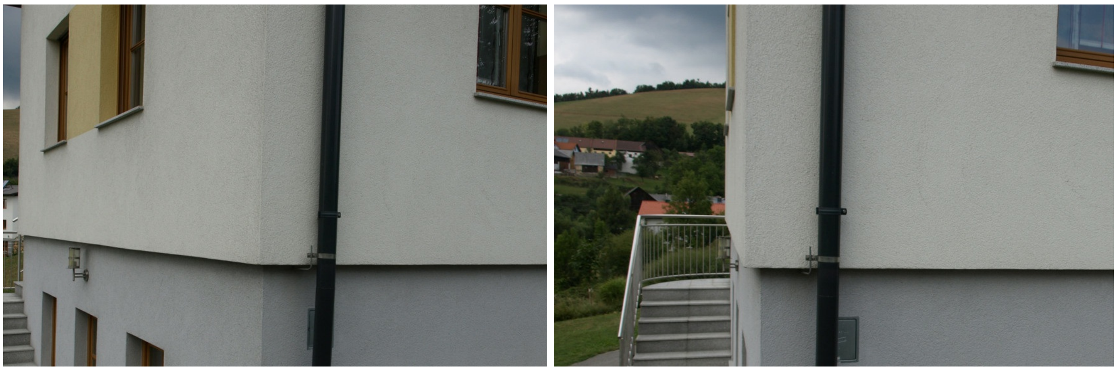


Figure 4.3: A small baseline is necessary around house corners because a mere small change in viewpoint makes for a large change in perspective.

4.3 Classification

Feature matching is a classification problem: Find the matching pairs within all possible pairings. Every matching technique should aim to minimize the two classes of error, false negatives and false positives.

False negatives are matching features that are not recognised as such.

Usually, visual similarity is a strong indicator for correspondence of two features. In fact, some form of basic descriptor correlation is usually required in all feature matchers preceding further analysis.

However, projective geometry and light capturing properties inherent to photography are often the biggest challenge in identifying similarities. Projections from different viewpoints induce affine distortions on a feature, which the feature descriptor should be able to express. Similarly, feature descriptors are usually robust against linear illumination change, for example when one shot looks away from the sun and the other shot looks into it. Especially problematic are cases where very few photos are available, for example historical photographs. Obvious correspondences might be missed, or the differences may be too stark to reconstruct anything useful at all.

On the flip side, not all features that really correspond can be used for reconstruction. For example, a person walking through a shot will probably move their hand. A hand is rich in distinct features and yields many matches. However, the motion has deformed the person's shape. Reconstructing rigid geometry from a scene containing something non-rigid will be difficult to impossible. Other pertinent sources of non-rigidity include trees swaying in the wind, clouds, people and vehicles, and moving specular highlights on shiny surfaces. A good feature matcher should recognise such features as not matching, despite their visual likeness.

False positives are also called *Mismatches.* They are features designated correspondent which don't actually represent the same thing in the real world.

Mismatches are the biggest detriment to convergence of a photogrammetry system since they actually introduce wrong information into the process. Finding a model that explains all available data in the best way possible becomes increasingly difficult when the data can't be explained, being inconsistent with reality.

Mismatches are often considered a necessary evil. Extracting as many features as possible from a photo is often desirable in order to represent an object very densely. No feature matcher is perfect, so including a few mismatches is the price to pay for increasing the number of correct matches. In fact, much of photogrammetry research focuses on increasing mismatch tolerance and improving convergence for imprecise information [Bau00] [HES15]. The underlying assumptions require a large enough amount of correct information to outweigh the effects of the erroneous observations. Oftentimes, some form of internal cohesion or parameter agreement is hypothesised in order to assess a solution's stability.

In truth, it's not solely the reconstruction algorithm's responsibility to resist the effects of mismatches, and a reconstruction pipeline shouldn't operate unaware of the matcher. A good feature matcher should be able to reduce the occurrence of mismatches, scaling with the amount of knowledge introduced.

Photogrammetric reconstruction imposes a number of restrictions pertaining to physical plausibility. It follows to understand these properties for the purpose of feature matching. In Sections 4.5 and 4.6, we aim to adjust the definition of a feature match away from a mere equivalence in physical manifestation, more toward its serviceableness in reconstruction.

4.4 Brute Force Matching

We require some sort of visual similarity between features in order to consider them matching. This is trivially the case a lot of the time.

The *Brute Force Matching* approach simply considers every possible pair of features and selects only pairs that look similar.

4.4.1 Similarity

Feature descriptors come with their own definition of similarity. Whatever information is contained in a descriptor, such as intensity values, gradients or frequency transforms, directly influences its mathematical interpretation. However, the most common feature descriptors are designed to support traditional distance measures.

Floating-point vector descriptors, such as SIFT and AKAZE, contain sets of measurements from surrounding pixels' intensities, or changes thereof. The *Feature Similarity* is the Euclidean distance between feature vectors.

Binary descriptors are fixed size bit strings that encode information in a structural way. An example is ORB, which tends to exhibit meaning in terms of textures and patterns. Similarity is defined as the Hamming distance between descriptors.

Matching features by similarity alone may suffice for very specific images, where enough distinct features are guaranteed to be available. Due to the speed and simple implementation, it's usually worth trying out.

4.4.2 Distinctiveness

In practice, especially in natural and architectural photos, similarity alone is not enough. A high number of mismatches is caused mainly by repeating patterns. For example, buildings often have rows of similar windows or long stretches of repeating ornaments. There are many features due to the high visual fidelity of the elements. However, many features repeat almost identically across the entire object (Figure 4.1).

To complement feature similarity, we introduce the measure of feature *Distinctiveness*. It's the measure of how unique a feature appears in an image, and establishes a quantity of confidence in a potential match in the other image. For clarity, we refer to the two images involved in feature matching as 'left' and 'right' image.

We implement similarity and distinctiveness as follows. For a feature in the left image, consider all potentially matching features in the right image. Those are, in the instance of the features we use, all features that pass the very liberal similarity threshold of `0.9` in normalised descriptor Euclidean distance. We order this set of candidates from most to least similar. The first feature in the ordered set becomes the true match if the second feature is only `t` times as similar than the first feature, where `t` is the *distinctiveness threshold*. An example for this configuration can be seen in Figure 4.4.

This ensures that a matching feature must look significantly more similar than all the other candidates. Thus, the implementation for the brute force matcher looks like in the following Code Listing 4.1. For simplicity, we assume the brute force matcher is always the first step in a more complex matcher.

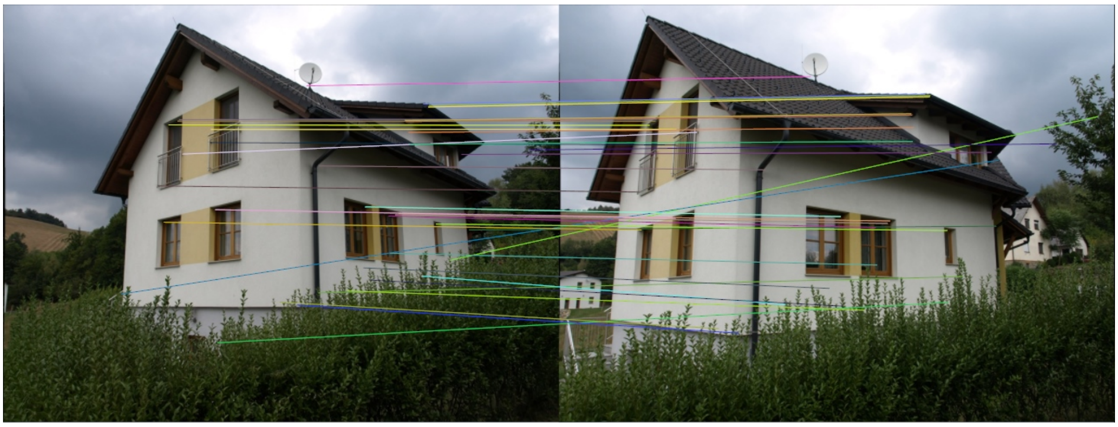


Figure 4.4: Brute force matcher. Matches are few in number, and mismatches often come from vegetation in the image. Matches are visualised as coloured lines.

Listing 4.1: Brute Force matcher.

```
let bruteforce (threshold : float) (l : list<Feature>) (r : list<Feature>) =
    let candidates =
        List.cross l r
        |> List.filter ( fun (lf,rf) -> lf.Distance rf < 0.5 )
        |> List.sortBy ( fun (lf,rf) -> lf.Distance rf )

    candidates
        |> List.filter ( fun (lf,rf) ->
            let ms =
                candidates |> List.filter (fun (l,_) -> l = lf)

            let (_,first) = ms.Head
            let (_,second) = ms.Tail.Head

            featureDistance first lf > threshold * (featureDistance second lf)
            )
```

Our final implementation also performs matching in the other direction, meaning that for all features in the right image selected as match, the matching feature in the left image must also be the best possible candidate. This is to ensure symmetry of matches. We omit this implementation detail from the code.

In our experiments, the brute force matcher on its own performs adequately with a threshold value of around `t = 0.6`. Lower values mean that only even more striking matches remain, although their numbers quickly become too few to be workable.

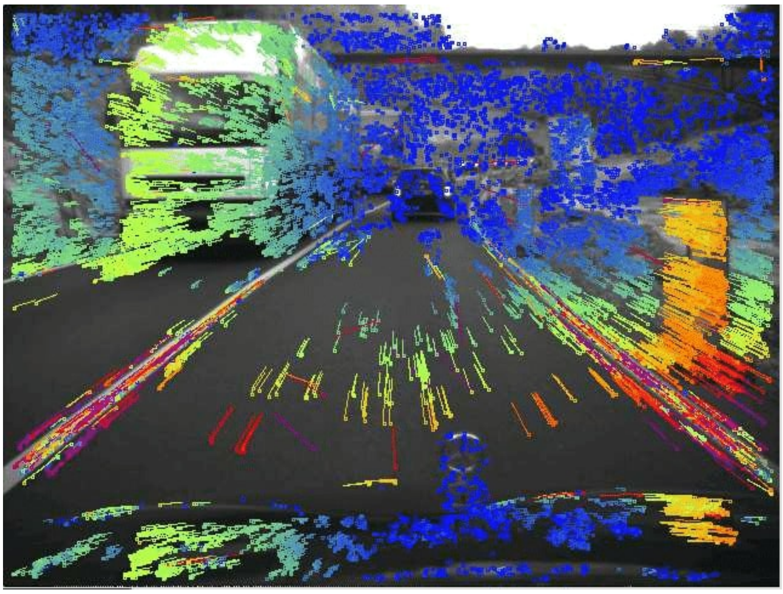


Figure 4.5: Motion of image regions in a road scene video, visualised as coloured lines. This concept is used heavily in video compression. Source: [Ste04]

4.5 Motion Consistency

We observe that feature matches are necessarily an expression of a physical relation beyond individual points merely looking similar. Critically, this includes the understanding of *motion*.

When taking photographs from different angles, the camera moves from one view point to the next. Features in the images exhibit a similar motion, inverse to the camera motion. Though the motion may be distorted by perspective, it must still be coherent and physically plausible. For example, if an object moves from one side to the other, all features on the object must exhibit the same coherent motion - no single point can break out and move in the other direction, because the other points wouldn't physically support such behaviour. An example of feature motion is shown in Figure 4.5.

This idea of a global view on feature motions was explored by Lin et al. in two publications, 'Bilateral functions for global motion modeling' in 2014 [LCL+14] and 'RepMatch' in 2016 [LLJ+16]. Analysis of features in an isolated fashion, such as with the brute force matcher, is seen as a piecewise recovery of a motion model that can be modeled globally. In fact, the global understanding of motion across two images is what allows model hypotheses supported by real observations in the first place, referring back to the idea of embracing context in understanding image content.

We implement this technique in the *Motion Consistency* matcher. The piecewise motion observations from individual matches are modeled as one smooth global motion hypothesis by placing a Gaussian distribution on every piece of data and learning aggregation parameters from the remaining observations' covariances. Concretely, this works as follows.

A feature match m in the pair of left and right image is comprised of left position x, the motion vector v which is the difference of the right and left positions, and the orientation change o.

$$m_i = [x_i; v_i; o_i]$$

The contribution of one match relative to its divergence from all N matches is given by the probability function $prob$

$$prob(m) = \sum_{i=1}^{N} w_i * exp^{-\frac{\|m-m_i\|^2}{\sigma}}$$

where the exponent assigns a high cost to observations that are wildly different from most others, w is the weighting parameter for an observation's contribution to the overall probability function, and σ is a free parameter representing an observation's range of influence.

To obtain the weighting parameters w_i, we minimise the divergences and according covariances over matches, using the Huber Loss function $huber$ to penalise outliers

$$huber(x) = \begin{cases} x^2 & \text{if } |x| < 1 \\ 2|x| - 1 & \text{else} \end{cases}$$

$$\underset{w}{\operatorname{argmin}} \sum_{i=1}^{N} huber(1 - prob(m_i)) + \lambda w^T G w$$

by choosing appropriate w, where $G(i,j) = exp^{-\frac{\|m_i - m_j\|^2}{\sigma}}$ and λ is the free parameter for punishing high variance solutions

The original authors use numeric methods to train w with some stable subset of all matches using RANSAC.

In our implementation, we go a different route instead. We choose to solve the equation analytically for all existing matches in order to keep the parameter set minimal. The minimum of the N-dimensional derivative is given by, in matrix form,

$$A(k,i) = \sum_{j=0}^{N} G(j,k) * G(i,j) + \lambda * G(i,k)$$

$$r(i) = \sum_{j=0}^{N} G(i,j)$$

$$A * w = r$$

which we solve for w by matrix inversion through QR factorisation. The code for the motion consistency matcher is thus in the following Code Listing 4.2, filtering matches by a consistency threshold parameter `pC`

Listing 4.2: Motion Consistency matcher.

```
let consistency (lambda : float) (sigma : float) (pC : float)
          (ms : list<Feature * Feature>) =
    //computes the consistency function
    let prob = probability lambda sigma ms

    ms |> List.filter (fun m -> prob m > pC)
```

We observe that this method improves the quality of feature matches substantially. Every feature match is supported by a large number of other feature matches that exhibit a similar motion, which is similar to the physical behaviour of rigid objects. Conversely, isolated matches sporting entirely different motion from the rest of the image, indicative of a mismatch more than anything else, are eliminated.

In our experiments, sensible values for the free parameters are about the same as the ones suggested in the original publication, which are about `lambda = 25.0` and `sigma = 0.6`, and a consistency threshold of `pC = 0.6`. In combination with the brute force matcher, we can use a much more lax distinctiveness parameter, like `t = 0.9`. An example result of this matcher is shown in Figure 4.6.

The motion consistency matcher can resolve ambiguity from repetitive structures by choosing the candidates that likely correspond to a real motion - we only need to ensure that observations of the real motion outnumber any contradicting motions.

4.6 Motion Prediction

A different application of the same technique is to not only establish a measure of confidence in a hypothesised motion, but to actually predict an affine two dimensional motion and to see how much an observation agrees with this prediction [LCL+14]. The underlying approach is similar to motion consistency, except that instead of the one parameter of motion differences, we are instead modeling the six parameters of affine two dimensional motion for each match explicitly.

The motion prediction function f in a dimension component d is therefore

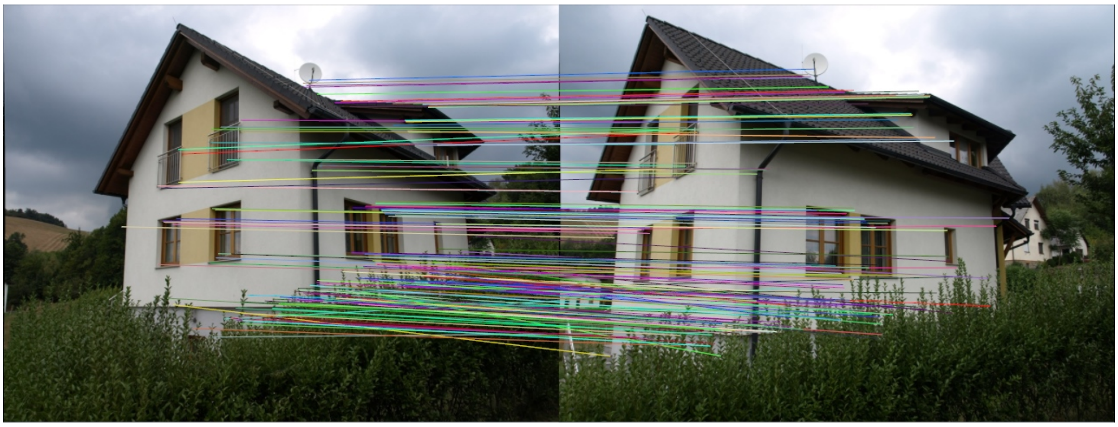


Figure 4.6: Motion consistency matcher. Matches are visualised as coloured lines.

$$f_d(m) = H_d + \sum_{j=1}^{N} w_d(j) g(m - m_j)$$

$$g(z) = exp^{-\frac{|z|^2}{\sigma^2}}$$

and the prediction for the motion in x and y dimensions, for a feature m with the coordinates x and y

$$pred_x(m) = f_1(m)x + f_2(m)y + f_3(m)$$

$$pred_y(m) = f_4(m)x + f_5(m)y + f_6(m)$$

and the accordance with a predicted motion, where feature m has the velocities u and v

$$pred(m) = (pred_x(m) - (x + u))^2 + (pred_y(m) - (y + v))^2$$

The motion model is characterised by the two affine parameters H_d and w_d, which appear three times each per dimension. The original authors train these parameters based on gradient descent and RANSAC for some stable subset of observations numerically.

Again, we choose to instead substitute our own solution. In order to avoid the additional parameters introduced by RANSAC and keep the computation performant, in our implementation we derive an analytical solution for the entire set of observations. This differs from the original authors' approach in that we use every available observation for parameter estimation. To make sure this set is stable enough, we compose this matcher with the simpler brute force matcher.

For readability, we assign letters to the affine parameters in each dimension. The cost function to train the parameters on a set of observations in the **x** dimension is

$$E(H, I, J, l, m, n) = \frac{1}{2} \sum_i \left[x_i + u_i - Hx_i - x_i \sum_j l_j g(i,j) - Iy_i - y_i \sum_j m_j g(i,j) - J - \sum_j n_j g(i,j) \right]^2 + \tag{4.1}$$

$$L = \lambda(\psi_l + \psi_m + \psi n), \psi_d = w_d^T G w_d \tag{4.2}$$

The derivatives, written in matrix form, for the $3N$ parameters depending on the match k, are

$$\begin{aligned} \frac{dE}{dl_k} = H \sum_i x_i^2 g(i,k) + I \sum_i x_i y_i g(i,k) + J \sum_i x_i g(i,k) + \sum_i l_i * [2\lambda g(i,k) + \sum_j x_j^2 g(i,j) g(j,k)] \\ + \sum_i m_i \sum_j x_j y_j g(i,j) g(j,k) + \sum_i n_i \sum_j x_j g(i,j) g(j,k) = - \sum_i (-x_i^2 - u_i x_i) g(i,k) \end{aligned} \tag{4.3}$$

$$\begin{aligned} \frac{dE}{dm_k} = H \sum_i x_i y_i g(i,k) + I \sum_i y_i^2 g(i,k) + J \sum_i y_i g(i,k) + \sum_i l_i \sum_j x_j y_j g(i,j) g(j,k) + \\ \sum_i m_i * [2\lambda g(i,k) + \sum_j y_j^2 g(i,j) g(j,k)] + \sum_i n_i \sum_j y_j g(i,j) g(j,k) = - \sum_i (-x_i y_i - u_i y_i) g(i,k) \end{aligned} \tag{4.4}$$

$$\begin{aligned} \frac{dE}{dn_k} = H \sum_i x_i g(i,k) + I \sum_i y_i g(i,k) + J \sum_i g(i,k) + \sum_i l_i \sum_j x_j g(i,j) g(j,k) + \\ \sum_i m_i \sum_j y_j g(i,j) g(j,k) + \sum_i n_i * [2\lambda g(i,k) + \sum_j g(i,j) g(j,k)] = - \sum_i (-x_i - u_i) g(i,k) \end{aligned} \tag{4.5}$$

and for the three remaining parameters

$$\begin{aligned} \frac{dE}{dH} = H \sum_i x_i^2 + I \sum_i x_i y_i + J \sum_i x_i + \sum_i l_i \sum_j x_j^2 g(i,j) + \sum_i m_i \sum_j x_j y_j g(i,j) + \\ \sum_i n_i \sum_j x_j g(i,j) = - \sum_i (-x_i^2 - u_i x_i) \end{aligned} \tag{4.6}$$

$$\begin{aligned} \frac{dE}{dI} = H \sum_i x_i y_i + I \sum_i y_i^2 + J \sum_i y_i + \sum_i l_i \sum_j x_j y_j g(i,j) + \sum_i m_i \sum_j y_j^2 g(i,j) + \\ \sum_i n_i \sum_j y_j g(i,j) = - \sum_i (-x_i y_i - u_i y_i) \end{aligned} \tag{4.7}$$

$$\begin{aligned}\frac{dE}{dJ} = H\sum_i x_i + I\sum_i y_i + JN + \sum_i l_i \sum_j x_j g(i,j) + \sum_i m_i \sum_j y_j g(i,j) + \\ \sum_i n_i \sum_j g(i,j) = -\sum_i (-x_i - u_i)\end{aligned} \tag{4.8}$$

The equations for the **y** dimension are almost the same, except the residual parts get replaced by

$$\frac{dE}{dl_k} = -\sum_i (-x_i y_i - v_i x_i) g(i,k) \tag{4.9}$$

$$\frac{dE}{dm_k} = -\sum_i (-y_i^2 - u_i y_i) g(i,k) \tag{4.10}$$

$$\frac{dE}{dn_k} = -\sum_i (-y_i - v_i) g(i,k) \tag{4.11}$$

$$\frac{dE}{dH} = -\sum_i (-x_i y_i - v_i x_i) \tag{4.12}$$

$$\frac{dE}{dI} = -\sum_i (-y_i^2 - v_i y_i) \tag{4.13}$$

$$\frac{dE}{dJ} = -\sum_i (-y_i - v_i) \tag{4.14}$$

These equations are saturated with all matches and solved by matrix inversion (QR decomposition), resulting in the optimal parameters for the motion model. The motion consistency matcher is implemented as in the following Code Listing 4.3

Listing 4.3: Motion Prediction matcher.

```
let prediction (lambda : float) (sigma : float) (pP : float)
            (ms : list<Feature * Feature>) =
    //computes the motion prediction function
    let pred = prediction lamba sigma ms

    ms |> List.filter (fun m -> pred m < pP)
```

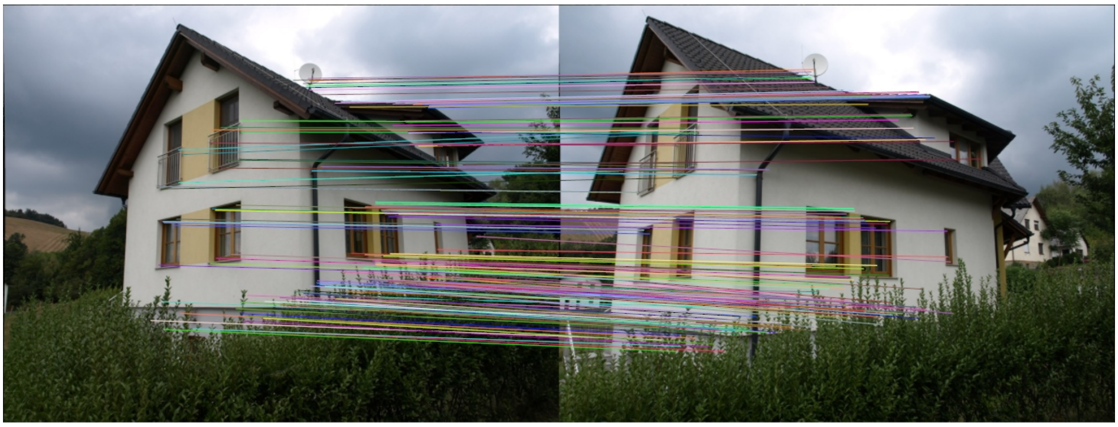


Figure 4.7: Motion prediction matcher. Matches are visualised as coloured lines.

In our experiments, we used the same values for `lambda` and `sigma` as with the motion consistency matcher. A sensible value for the prediction threshold is `pP = 0.01`, which is also a value suggested by the original authors of the publication. An exemplary use of the prediction matcher can be seen in Figure 4.7.

This matcher considers matches primarily in the domain of motion instead of the spatial domain, which means that it can find significantly more matches within regions where perspective distortion is strong. On the other hand, the computation is somewhat more complicated and usually takes several times longer than the consistency matcher.

4.7 Composition

The matchers discussed so far complement each other in terms of strengths and weaknesses. The brute force matcher doesn't take much context into account, but is able to identify matches without risk of confusion. The motion consistency matcher prefers matches that appear in cohesive groups, but needs a distinct input. The motion prediction matcher finds matches that show plausible motion across the entire image as long as the input features behave reasonably in local. Therefore, these three matchers lend themselves to being run in sequence.

We implemented our matchers as a filter function on a set of matches, meaning that every intermediate result is a valid final result, and we also took care of keeping appropriately ordered signatures. Thus, we achieve sequential application by simple function composition. Such an example can be seen in the following Code Listing 4.4

Listing 4.4: Composite matchers through function composition.

```
let motionMatcher t lambda sigma pC pP =
    bruteforce t
        >> consistency lambda sigma pC
        >> prediction lambda sigma pP
```

Some example feature matchers are given in Code Listing 4.5. Note that every matcher has the same consistent function signature `list<Feature> -> list<Feature> -> list<Feature * Feature>`, taking features from the left image, features from the right image, and returning a set of feature pairs, which are the matching features across both images. The matchers are visualised in Figures 4.8, 4.9 and 4.10.

Listing 4.5: Example feature matchers incorporating the motion model.

```
let veryStable      = motionMatcher 0.82 35.0 0.4 0.6 0.005
let middleGround    = motionMatcher 0.9 25.0 0.4 0.5 0.01
let manyMatches     = motionMatcher 0.92 15.0 0.5 0.5 0.01
```

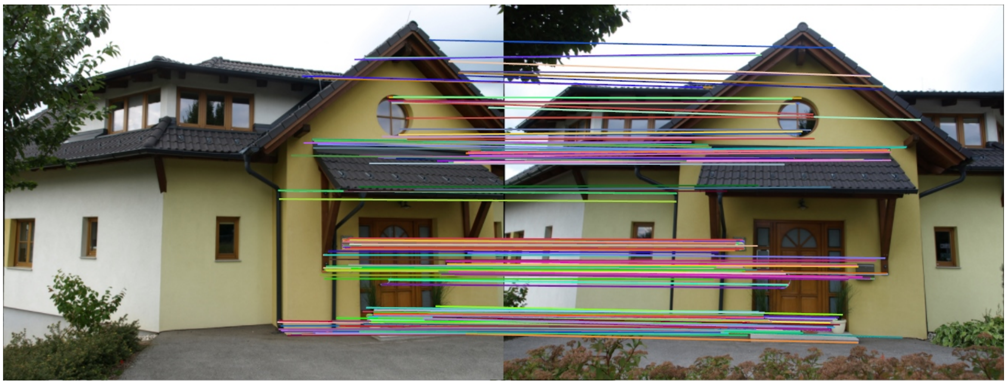


Figure 4.8: `veryStable` matcher producing about 300 matches.

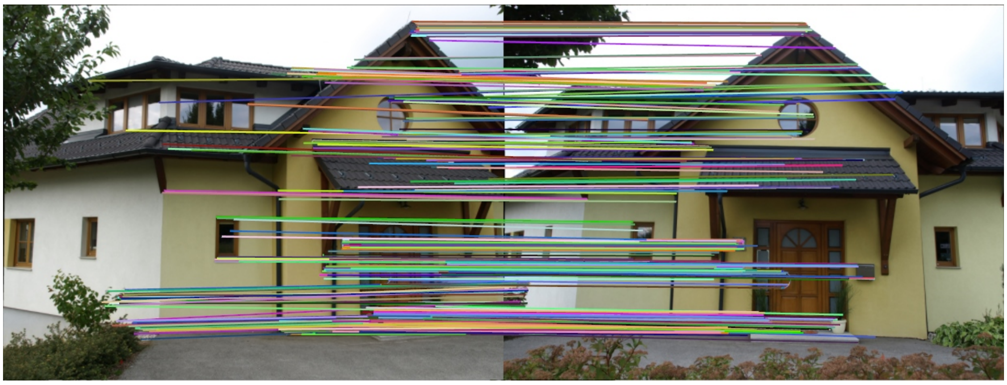


Figure 4.9: `middleGround` matcher producing about 500 matches.

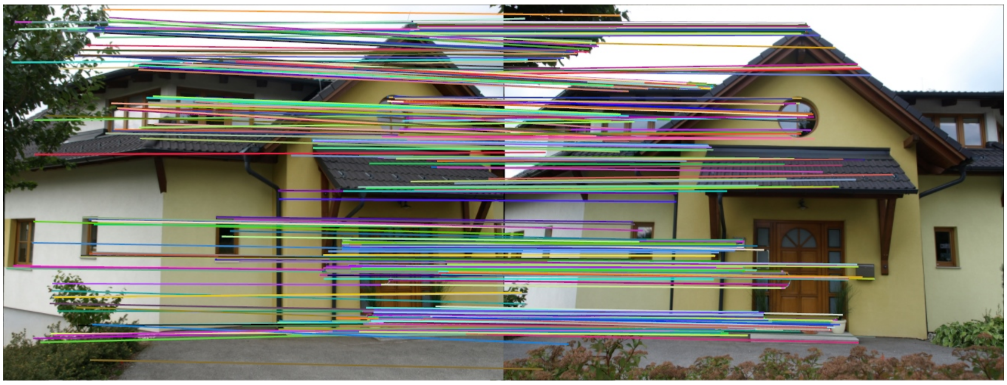


Figure 4.10: `manyMatches` matcher producing about 700 matches.

CHAPTER 5

Multi View Geometry

Photography is an instance of a projective transform. The human visual system is capable of understanding the distortions present in a perspective image, from which our conscience intuits object distance and scene depth relations.

Circles appear as distorted ellipses, parallel lines vanish at the horizon, closer objects obscure those further away. Visual effects heralded by perspective projection, so called *Depth Cues*, allow us to infer scene structure. The photo in Figure 5.1 shows some of these depth cues.

However, on its own, this so-called *Projective Geometry* has no Euclidean connotation, as a particular projective image could have been produced by infinitely many different projections. This property is called *Homogeneity*. The geometry prefers no particular coordinate system, it's valid for them all.

We want to estimate the 3D geometry of an object as well as the external camera parameters, locations and orientations, of several shots. Both refer to the same frame of coordinates, which is the Euclidean coordinate space. This coordinate frame can't express some properties of projective space, such as the intersection of parallel lines, and needs to treat them differently. We call such a space a *Heterogeneous Coordinate Space*.

They key to transitioning between homogeneous coordinates, which describe our photos, and heterogeneous coordinates, which describe the 3D world, is the *Projection Function*. In the case of a photo camera, this projection function is largely defined by the internal camera parameters.

We now discuss the coordinate frames involved in photography, projective, affine and Euclidean space, and how to traverse between those spaces with the information usually available.

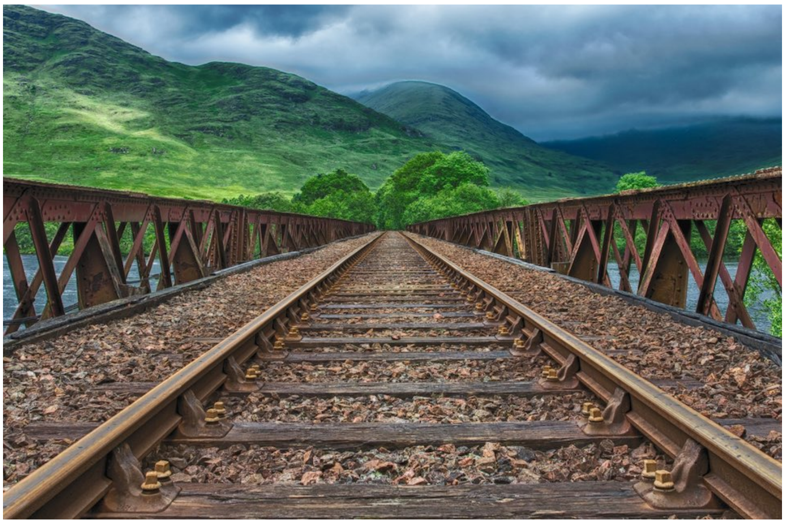


Figure 5.1: Perspective used as artistic tool in a photo. Human perception infers depth from perspective distortions.

5.1 Homogeneous Coordinates

Classical Euclidean geometry treats every object the same. A geometric object always retains its defining properties, such as lengths, angles, proportions, and so on. Similarly, a geometric transform is applied to every point of an object equitably. We say, an object is mapped onto its image under the transform. This mapping destroys or loses no information.

Euclidean Coordinates describe the locations of points. The value of a coordinate is given relative to a reference point, the *Coordinate Frame.* The values of coordinates change as different coordinate frames are chosen. Geometric transforms can be interpreted as movement of the coordinate frame, the 'world', around the object, but they can also be interpreted as the movement of the object within a fixed frame.

Euclidean coordinates are insufficient to describe the real world, however. To illustrate this point, consider the *intersection of two 2D lines*, which is the point where two lines meet in a 2D coordinate system.

When talking about intersections, we're required to make a glaring exception. Some lines don't have an intersection. We call those lines *parallel.* Colloquially, parallel lines are said to meet 'at infinity', a virtual point that is only used as a verbal aid (Figure 5.1). Euclidean transforms have to respect these exceptions however, which leads to mathematical grief.

In a 2D coordinate system $\mathbb{R}^2$, we identify a point as an ordered pair of values

$$(x, y) \in \mathbb{R}^2$$

We now declare the so-called *homogeneous coordinate* as

$$(x, y, 1) \in \mathbb{P}^2$$

which represents the same point in $\mathbb{R}^2$. We simply extend a coordinate $\in \mathbb{R}^2$ by a third value of 1. This new coordinate $\in \mathbb{P}^2$ represents the same point. Trivially follows the conclusion

$$(kx, ky, k) \in \mathbb{P}^2$$

all represent the same point $(x, y) \in \mathbb{R}^2$ through division by k.

From this follows another conclusion,

$$(x, y, 0) \in \mathbb{P}^2$$

does not have an associated point in $\mathbb{R}^2$, since division by the homogeneous coordinate would result in 'infinity'.

We call points $\in \mathbb{P}^n$ with a homogeneous coordinate of 0 *ideal points* or *vanishing points.*

Parallel lines intersect in a vanishing point.

By associating the Euclidean space $\mathbb{R}^n$ with the projective space $\mathbb{P}^n$, we are able to describe geometries, such as intersecting lines, in a truly homogeneous fashion. We do not have to make exceptions.

Photography is understood as a mapping from projective space to image space. Subsequently, observation of an image point allows us interpretation of its homogeneous equivalent

$$d * (x, y, 1) = project(x, y, z)$$

The projection of the 3D world is the homogeneous space for the 2D image plane, where the homogeneous coordinate d is called the *depth.*

The *project* function itself has projective properties, so we choose the representation of the 3D point $\in \mathbb{R}^3$ as its equivalent in homogeneous space $\in \mathbb{P}^3$. This allows us to express the operation in a single analytical function

$$project(x, y, z) = k * (K * R * t) * (x, y, z, 1) = P * X \tag{5.1}$$

where division by the *homogeneous coordinate k* is called the *perspective division.* The function is written in matrix form using K, R and t, which are exactly the camera parameters discussed in Section 2.3.

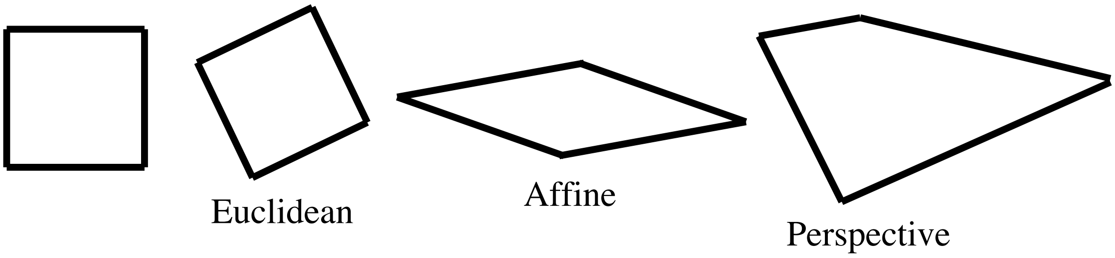


Figure 5.2: Geometric transforms visualised. Parallel lines intersect at infinity in Euclidean and Affine space, but the intersection of parallel lines can be localised in Projective space.

Familiar geometric concepts such as lengths and angles acquire meaning in this representation, allowing us to hypothesise on 3D structure from projective observations. Importantly, the analytical form also permits calculation of derivatives, enabling numeric optimisation.

5.2 Epipolar geometry

The camera model underlying the projection function is the pinhole camera (discussed in Section 2.1). It defines one point as the camera centre through which incoming light rays pass to intersect with the image plane, yielding image points.

Using the analytic description of the imaging process, we establish the same operation for not one, but a pair of cameras at the same time. A configuration of two cameras observing the same 3D point is called *Epipolar Geometry.*

Several important properties arise. For clarity, we refer to the two cameras as ’left’ and ’right’ camera. A diagram of epipolar geometry is shown in Figure 5.3.

The line connecting both camera centres is called the *Epipole* or *Epipolar Line.* The point of intersection between epipole and a camera’s image plane is the image of the other camera’s centre.

The triangle of both camera centres and the world point forms a plane called the *Epipolar Plane.* The epipole lies on that plane.The 3D point corresponds to an observation in both cameras.

But what if the 3D point itself is not known? This is the first fundamental assumption of photogrammetric reconstruction: Only observations are available as evidence of the underlying geometry, but the true 3D scene needs to be reconstructed.

An image point is actually the intersection of a 3D ray originating from the camera centre, called the *view ray*, with the image plane. The real 3D point that was observed could

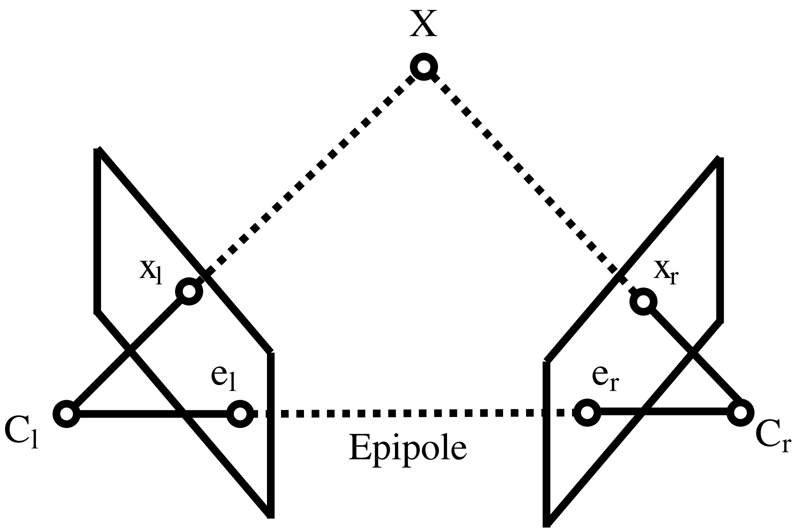


Figure 5.3: Epipolar geometry. The camera centre of the left camera C_l appears in the right image as e_r, the intersection between right image plane and the line between the two centres, which is called Epipole. The triangle of the two camera centres and the world point X forms the Epipolar Plane.

have been any point on this ray of light. This fact is intuitively clear: In projective space, the choice of homogeneous coordinate, the distance on the ray away from the centre, does not actually influence the Euclidean counterpart, up to free choice of coordinate frame. We call all of these points *projectively equivalent*, and say that projection partitions points into *projective equivalence classes.*

An observation's view ray originating from the left camera lies on the epipolar plane. In the right camera, the view ray appears as a 2D line, which is also the intersection of epipolar plane and image plane. On this line must lie the right camera's observation of the same point. The intersection of both view rays is the location of the observed 3D point, finding which we refer to as *Point Triangulation.*

The two matching observations are called *Point Correspondences.* We establish an important property of epipolar geometry, the *Correspondence Constraint.*

$$P_l X_i = x_{li} \wedge P_r X_i = x_{ri}, \forall i$$

Projection of point X_i results in the image point x_i in both cameras. For i many points, this constraint must hold for all points X_i at the same time.

Now, what if the camera matrix P wasn't known either, in addition to the unknown 3D point X? This is the second fundamental assumption and main motivation for photogrammetric reconstruction: Our observations evidence both an object and the camera, but our model must explain the evidence as plausibly as possible.

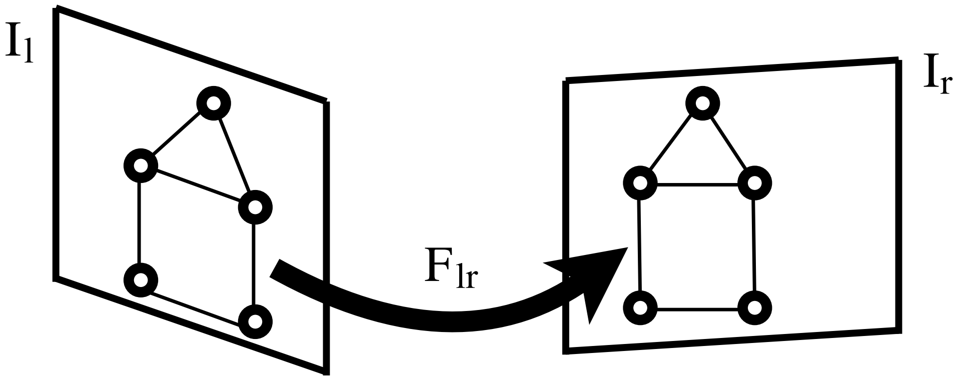


Figure 5.4: The Fundamental Matrix F_{lr} transfers corresponding points from left image I_l to right image I_r.

It's clear that a reconstruction can not be determined uniquely, but only relative to some previously established reference point. Any scale, orientation or position of the reconstructed scene is arbitrary. We call this a reconstruction *up to a similarity transformation of the world.*

Even further, we have established that changing the homogeneous coordinate of a projected point does not affect the image point, so

$$PX_i = PH^{-1}HX_i$$

applying a projective transform H on a point and its inverse on the camera results in the same projection in Euclidean space as the original. We call all equivalent choices for H the *projective equivalence classes of reconstructions*, and we say a reconstruction is up to *projective ambiguity.*

This ambiguous type of Euclidean reconstruction is only meaningful if viewed from either of the two cameras. However, we derive from it the basic operation for globally consistent photogrammetry.

Simple reordering of the correspondence constraint gives us

$$P_l X_i = x_{li} \wedge P_r X_i = x_{ri}$$

$$x_{li} P_l^{-1} = Xi = x_{ri} P_r^{-1}$$

$$x_{ri}^T * F * x_{li} = 0$$

where F is a 3x3 matrix of rank 2. We call F the *Fundamental Matrix.*

Note that we obtain an equation linear in F, meaning that all we need is a satisfactory number of observations x_{li} and x_{ri} to be able to compute its entries. The relation between observations and Fundamental matrix is shown in Figure 5.4. We then decompose F to obtain both cameras' camera matrices. They give us positions and orientations of the cameras relative to each other, which permits us to triangulate all image points and obtain 3D geometry. We have achieved reconstruction of a scene!

5.3 Euclidean Reconstruction

Although a projective reconstruction can be of conditional use, for example in point transfer, where observations are transferred directly from left to right image, our goal is to obtain a result of true Euclidean interpretation.

Projective reconstruction produces an equivalence class of reconstructions with respect to projective transforms. Let us consider the information required to single out exactly the projection consistent with the cameras', and thus obtain the Euclidean interpretation consistent with the real world.

Parallelism is not a concept of projective geometry. Parallel lines meet at infinity, but there is no distinction between points at infinity and the other ones because all points are equal. Thus, the free choice of coordinate frame does not preserve parallelism, nor critical accompanying properties, such as ratios of lengths.

We need to single out one feature of projective space to gain the concept of parallelism.

Consider the two dimensional projective space $\mathbb{P}^2$. We introduce one line and distinguish it as the *line at infinity.* Next, we imagine two lines that intersect exactly on the line at infinity. Since the lines 'meet at infinity', we call those lines parallel. Clearly, the resulting geometry, containing the notion of parallelism, can express lengths and ratios. If two lines between the points A,B and C,D are parallel, then the lines are equally long if the lines A,C and B,D are also parallel.

This model is consistent with the real world. The classic example is the image of a flat landscape with railway tracks, parallel lines, going off into the distance and meeting at the horizon, the infinity line. The landscape is an example of a two dimensional projective geometry embedded in three dimensional space.

The same concept extends to three dimensional projective space, except the infinity line becomes a *plane at infinity.*

The geometry of a projective space together with a distinguished plane/line is called *Affine Geometry.* A projective transform which maps between the distinguished planes/lines of different spaces is called *Affine Transform.*

The view rays of a camera which is not located on the plane at infinity, intersect the plane at infinity at exactly one point each. Thus, the plane at infinity maps one-to-one onto the image plane of the camera. Different cameras may issue different projections,

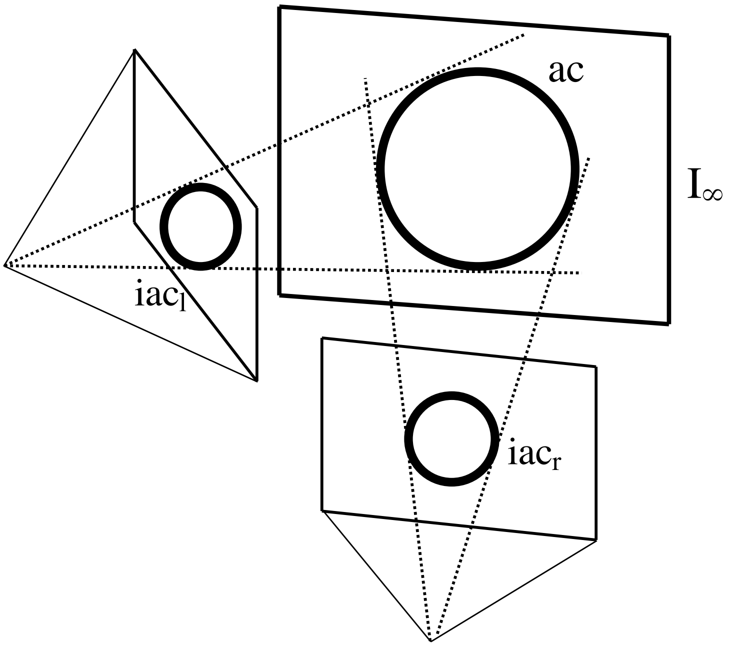


Figure 5.5: Symbolic depiction of the Image of the Absolute Conic (IAC). The IAC looks the same from every viewpoint and therefore puts different cameras in relation.

but the images of the plane at infinity hold the information necessary to put them in relation.

Consider a circle in 2D. Any stretching of space in one dimension, an affine transform, maps a circle onto an ellipse. Hence, a circle is not a distinguishable concept of affine space - circles and ellipses are the same.

$$(x - aw)^2 + (y - bw)^2 = r^2w^2$$

is the circle equation using homogeneous coordinates, where w is the homogeneous component. We observe that there exists a pair of points that lies on every circle,

$$(x, y, w) = (1, \pm i, 0)$$

These points are called *Circular Points.* Since the homogeneous coordinate is zero, the points lie on the line of infinity. Even though their coordinates are complex, they satisfy the real circle equation.

We also observe that the circular points remain the same regardless of affine transformation of the space around them. The real-world analogy here is the sensation of looking at the moon while walking down a road. The moon appears to remain at the same spot in the sky regardless of our movement due to its practically infinite distance.

The *Absolute Circle* defined by the circular points is actually at 'infinity', and its image appears at a fixed position even as the observer moves or rotates.

This concept generalises to three dimensions. The distinguishing entity becomes not a circle, but a general conic, which is fully defined by five circular points. It is called the *Absolute Conic*. It also lies on the plane at infinity, and since the plane at infinity maps one-to-one to the image plane, its image is a conic the same. We call it the *Image of the Absolute Conic*, or *IAC*. An illustration of the IAC can be seen in Figure 5.5.

If we know the IAC in an image, we can put its camera in relation to the world. We call a camera with a known IAC *calibrated*.

Note that this definition of camera calibration is no different from the one given in Section 5.4. The projective space induced by a camera depends only on the internal camera parameters, since the rotation and translation components, the external parameters, have no effect on the IAC.

The Absolute Conic acts as ruler for metric measurements in an image. Since the plane at infinity maps one-to-one to the image plane, we express projective distances and angles between points as their distance ratios relative to the IAC, also a feature of the plane at infinity.

Critically, in a calibrated camera, the true IAC is known, which means that the measurement of distances and angles is also valid in every other calibrated camera. More generally, it is valid under every transform that doesn't change the IAC, which are exactly the transforms that don't change lengths.

We call a transform that preserves lengths *Euclidean* or *Metric Transform*. A visualisation can be seen in Figure 5.2.

A two/three dimensional projective space together with a distinguished line/plane at infinity and two/five circular points is called *Euclidean Space*.

We call a reconstruction performed only with calibrated cameras *Euclidean Reconstruction*.

Obviously, it's important to understand and respect the steps from projective space over affine space to Euclidean space. Almost every reconstruction task includes some form of distance or angular measurement, be they projective (directly in the photo) or Euclidean (in 3D space), and many more complex tools use these notions as basic building blocks.

Camera calibration is the fundamental operation enabling metric measurements. The resulting Euclidean reconstruction contains 3D geometric objects 'how they really are', that is, how we would expect them to be represented in a Euclidean context, circles being circles and right angles being right angles. The results may be directly used, for example, in computer graphics renderings.

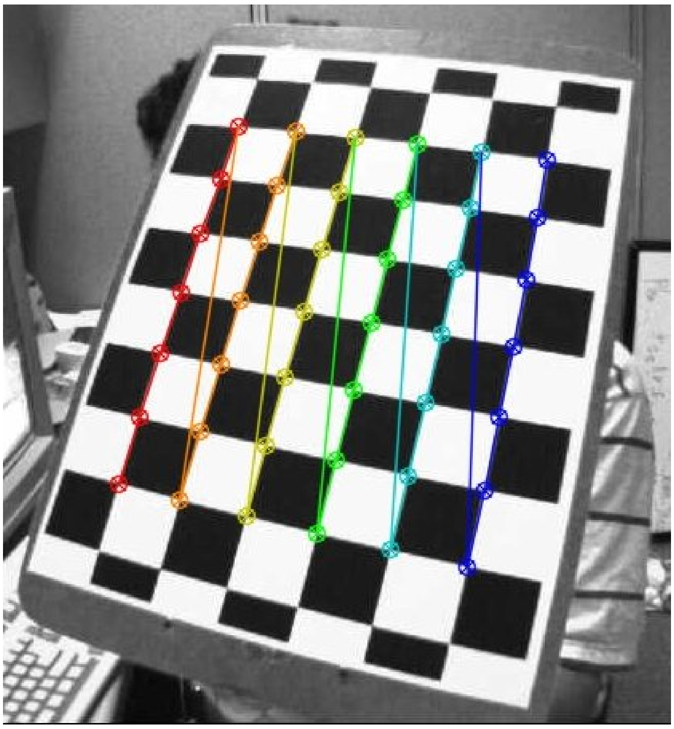


Figure 5.6: OpenCV calibration pattern. The pattern's structure is easily recognised in the image (coloured lines). Enough photos of the patterns from different viewpoints enable numeric recovery of the internal camera parameters. Image Source: [Opef]

5.4 Camera Calibration

The true IAC itself can be used to obtain a camera's calibration, a technique performed in a range of applications, for example camera auto-calibration [Tri97].

However, in our case, we're not interested in the actual IAC, rather than the parameters of the projection producing it.

We use photo cameras available to common consumers. Many of these cameras specify their internal parameters in the Exif data of their images. Cameras and objectives also often provide them in databases which can be accessed on the internet.

An additional source for camera calibration are existing numeric calibration techniques, for example the widely used OpenCV `cvCalibrateCamera2` function [Opec]. This function makes use of a known pattern that is simple and can be easily printed out on a piece of paper and stuck to a flat surface, such as a checkerboard pattern. The shape's projection is straightforward, allowing the function to quickly recover distortion parameters provided photographs of the pattern from enough different angles and constellations. A calibration pattern is depicted in Figure 5.6.

Through combination of various sources, internal camera parameters can usually be obtained to a certain degree of precision.

We observe that we can avoid the nonlinearities introduced by the internal camera parameters K for purposes of numerical optimisation. The internal camera parameters except for the focal length are a property of a camera rather than a particular shot, meaning they remain constant for many, or sometimes even all, shots, once known. Therefore it makes sense to apply the required effects once for every image as a pre-processing step and remove them from our ongoing computations.

We apply the inverse of the internal camera matrix, except focal length, to every pixel of the image, producing an output image with the effects of lens distortion and projection artefacts removed. This is exactly the image undistortion process described in Section 2.3.3. For undistorted image points, we say that their locations are given in *Normalised Image Coordinates.*

We also multiply the inverses of K_l and K_r onto the projections P_l and P_r involved in the fundamental matrix F. The result is called the *Essential Matrix E*.

$$K_l^{-1} * F * K_r^{-T} = E = R * t$$

The essential matrix E describes the movement between two cameras' 2D normalised image coordinate projections of 3D points. E is comprised of a rotation component R and translation component t, and may be decomposed as such.

We made an exception for the focal length *fx* and *fy*, which are multiplied onto the normalised image coordinates to obtain undistorted image coordinates. We do this because in practice, a supplied value of the focal length usually isn't too precise, especially when reading it from the Exif metadata. It's desirable to slightly adjust the focal length as well during numeric optimisation to improve accuracy, which is unproblematic since its only a linear relation.

In summary, we apply the knowledge in the preceding chapters to reconstruct 3D geometry from two photos as follows:

- Calibrate the camera and undistort the photos
- Extract features
- Match features to obtain correspondences
- Use correspondences to estimate the essential matrix E
- Decompose E into rotation R and translation t, placing both cameras in the world
- Triangulate 3D world points from features

We call this process *Pose Recovery.*

CHAPTER 6

Pose Recovery

The centrepiece of pose recovery is finding an adequate estimation of the Essential Matrix E, and obtaining a sensible Euclidean 3D constellation based on the decomposition of E into rotation R and translation t. There are several levels of ambiguity inherent to the problem which make this process fundamentally challenging.

Firstly, reconstruction can only be performed up to a degree of translation, orientation and a degree of uniform scale. A reconstruction purely from photos with no additional information can only ever be consistent within itself, but there is no reference to any absolute coordinate system in global space.

Colloquially speaking, we can reconstruct the shape of a house, but we can't tell if the house stands in Iceland or Siberia, and whether its a luxury villa or a miniature doll house. In fact, every possible location, orientation and uniform scale of the reconstruction is valid. It is a reconstruction *up to similarity.*

Secondly, every camera may be looking forward or backward. In the pinhole camera model, the light rays traveling through the camera centre may be observed in front or behind the pinhole. The resulting images on the image planes on either side of the centre are the same, except flipped on both axes.

Mathematically, we observe a flip in the sign of the depth, that is the z-coordinate of a projected point before dehomogenisation, for an image point. We designate for points in front of the camera *positive* depth, and behind the camera *negative* depth. We know that, in the real world, our cameras can only make photographs in one direction. Therefore, we can resolve this ambiguity by requiring all points to have the same depth sign.

Thirdly, the results of estimating E inherently carry a qualitative ambiguity. When estimating E, that is, when the cameras are calibrated and we have enough observations, there are four unique solutions that all correctly fulfil the requirements.

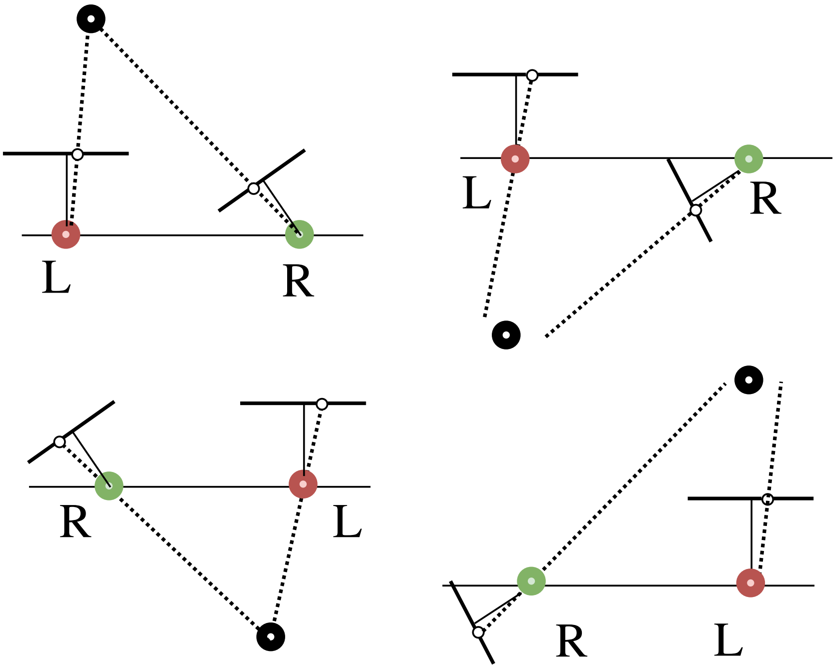


Figure 6.1: Four constellations producing the same image. *top left*: base configuration, *bottom left and bottom right*: the two cameras have swapped positions, *right top and right bottom*: one of the cameras has rotated 180 degrees around the baseline.

The first pair of solutions are the two cameras positioned relative to each other, and the two cameras with their locations swapped. The triangulated observations are either in front or behind both cameras. The second pair of solutions are the same as the first, except the cameras have performed a corkscrew motion around the baseline. The triangulated observations are behind and in front of one camera each. These configurations are illustrated in Figure 6.1.

Fourthly, the estimation of E may only ever yield the direction of the translation component t, but not its length. This is true because, even with constant focal length, the cameras may simply be moved closer together or further apart without changing any observations. The triangulated points would simply move further away from or closer to the cameras. This property is illustrated in Figure 6.2.

This ambiguity is fundamentally part of pose recovery and can not be resolved in general. The only way to determine a plausible length of t is by introducing already existing truth, and scaling t such that the truth is best explained.

In the following, we will discuss the algorithms required to estimate E, decomposing E to ground our reconstruction in the 3D world, and utilising existing knowledge in order to reduce ambiguity until we obtain an internally consistent solution.

6.1 Eight Point Algorithm and Five Point Algorithm

The so-called *Eight Point Algorithm* [Har97] is the numerical way of finding the least squares solution for the 3x3 components of the fundamental matrix F. F describes the epipolar constraint between two cameras, and must fulfill that constraint for all existing observations. Specifically, multiplication of an observation in the left camera with F results in the corresponding observation in the right camera (see Figure 5.4).

Since the constraint itself is linear, we can formulate a set of linear equations that explains all correspondences. In the obvious next step, we sum up over the equations' factors and rearrange to obtain the squared distance to the ideal solution, solving for which yields the parameters for the optimal solution in the least squares sense. Since we are trying to solve for 3x3 parameters, one of which is in relation with the constant factor of the quadratic equation, we need to fill in at least eight point correspondences to saturate the equation, hence the name eight point algorithm.

P is the 3D coordinate of a point in the world, and O are the two camera origins. In an epipolar system, we have

$$\overline{O_lP}, \overline{O_lO_r}, \overline{O_rP} \text{ coplanar, therefore } \overline{O_rP} \cdot \overline{O_lO_r} \times \overline{O_lP} = 0$$

We consider the transformation R, t from left to right camera, such that the image points $x_r = R(x_l - t)$, and we note that the translation is the same as $t = \overline{O_lO_r}$. $P_{l|r}$ are the 3D coordinates of a point in the world with the coordinate frame origin chosen at $O_{l|r}$, i.e. the view ray. This gives

$$(\overline{O_rP})_l = P_l - t = R^{-1}P_r + t - t = R^TP_r$$

and consequently

$$P_r^TR(t \times P_r) = P_r^T(Rt)P_l = 0$$

which is the epipolar constraint for the point P. But since the projections $p_{l|r}$

$$K^{-1}p_l = P_l \quad \text{and} \quad K^{-1}p_r = P_r$$

we get the form of F as

$$P_r^T(Rt)P_l = p_r(K^{-1}RtK)p_l = p_rFpl = 0$$

where F is the fundamental matrix.

$$\text{If} \quad K = I \quad \text{then} \quad E = F$$

where I is the identity matrix and $E = Rt$ is the essential matrix.

Straightforward least squares solution for the components of F gives

$$p_{i_r} F p_{i_l} = 0$$

equivalent

$$\begin{bmatrix} x_{i_r} x_{i_l} & x_{i_r} y_{i_l} & x_{i_r} \\ y_{i_r} x_{i_l} & y_{i_r} y_{i_l} & y_{i_r} \\ x_{i_l} & x_{i_l} & 1 \end{bmatrix} * \begin{bmatrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ f_{31} & f_{32} & f_{33} \end{bmatrix} = 0$$

for the *ith* observation. Filling in `N>=8` observations p_i gives a linear system of equations

$$Af = 0$$

where the matrix inverse A^{-1} is the solution for f and subsequently for F.

Since the observations might be inherently flawed due to measurement errors, using all measurements for estimation at once often leads to inaccuracies and instability. It has proven superior to apply a RANSAC based approach instead. RANSAC is a maximum vote procedure which selects the input subset that explains at least some portion of the entire input. The parameter to control the size of this portion, and the fitness of the result to the input data, is the RANSAC threshold. This threshold should be chosen appropriately high to be accurate, but low enough to generalise, according to the present problem.

The *Five Point Algorithm* [Nis04] is a faster and more robust alternative that requires only five point correspondences, but the mathematics and implementation is a degree more complex than the rather simple eight point algorithm, and requires solving of nonlinear equations. It has been implemented efficiently in OpenCV [Opeg], and we use that in our implementation.

6.2 Decomposition into R and t

Given an affine transform, we want to obtain the three dimensional movements it represents to be able to reason about the scene.

This is fairly straightforward for E, since it is comprised of only the two Euclidean transformations R, t, through a simple singular value decomposition. Interpreting this decomposition is somewhat more involved, however, because we describe the movement through observations made in projective space. This means that we're actually dealing with an equivalence class of movements that could all have lead to the same projections. The candidates we obtain for R and t retain a degree of ambiguity, which we need to resolve.

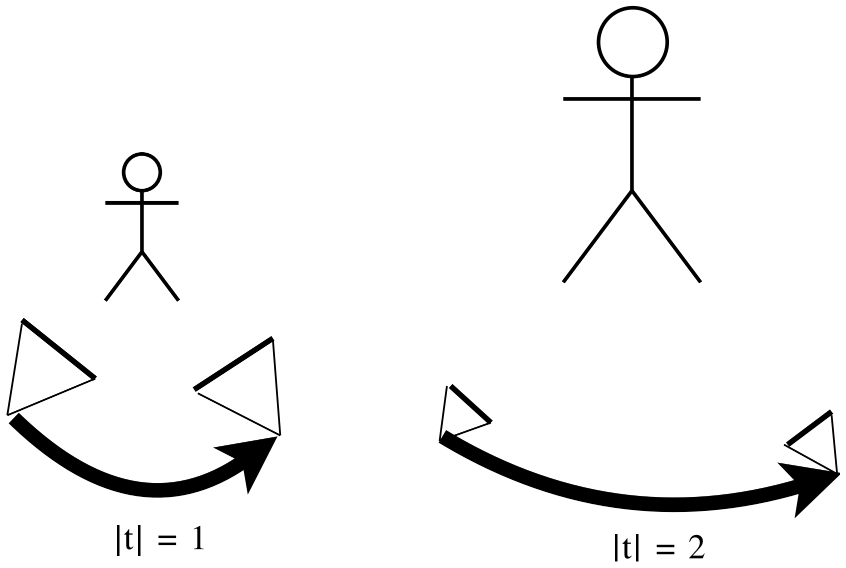


Figure 6.2: The same image is produced using different distances $|t|$ between the two cameras. The 3D scene is simply scaled accordingly.

6.3 Ambiguity Removal

There are infinitely many candidates for R and t in general. We identify three kinds thereof, the ambiguities of the global situation, camera front and back sides, and the distance between two cameras. Special considerations have to be applied in order to find the candidates that are suitable for our purposes, which is to feasibly explain everything we already know about our real-world scene.

Since the motion is only relative between the two cameras, one of the cameras necessarily has to be anchored to a fixed position and orientation to be able to derive the other one.

In the most general case, when there is no other information available, we define a canonical base camera as starting point. We use the *identity camera* for that purpose. Its position is in the origin and its axis points along the negative $\mathbf{z}$ axis of the Euclidean coordinate system. One of the two cameras is assigned as the identity camera, and the other one is positioned relatively by applying R and t to the origin and axis. This canonical base also serves to define a global situation to which we refer in our reconstruction.

In the more specialised case, if there is existing information, one of the two cameras must already have been positioned in space previously. In that case, we simply retain the existing location and orientation for that camera, and apply R and t to situate the remaining camera relative to it. This effectively allows us to 'chain' individual reconstructions together.

Further, we require that all triangulated 3D points are in front of the camera. We assume that a physical camera could have only made a picture in front of itself, but not in the other direction or inside its body. After triangulating the points, they are projected back into both cameras, and all observations are required to have positive depth. As long as solutions for E exist, we can always find one that fulfils this criterion, at least for most points (the outliers stem from measurement errors). This removes the ambiguity whether points are in front or behind the cameras.

Finally follows the estimation of the length of t. Having only the direction of t, the correct *length* of t positions the camera in such a way that any existing 3D points are also explained feasibly, along with the 3D points we are trying to triangulate.

In the most general case, in which there are no existing 3D points, we define a canonical base configuration, in which the length of $t = 1$.

In the other case, prior information exists, we know that one of the two cameras must have already been involved in a previous reconstruction. This yields an additional constraint: those 3D points that already exist must be projected onto their observations in the camera that saw them.

To obtain t, we apply the same rationale as in Section 6.1, which is to find the motion that best explains existing observations, except instead of a general transform we only consider the 1D movement along the translation direction. Again, we utilise RANSAC to find the maximum likelihood solution, discarding outliers as mismatches. Importantly, we also use the same frame of measurement (normalised coordinates) and the same threshold parameter, to maintain parameter consistency. For simplicity, we choose the distance between a pair of corresponding 3D points as candidate for t, because the majority of those must overlap by definition.

The function for scaling the translation vector, Code Listing 6.1,

Listing 6.1: Finding the best scale for the translation t between two cameras if 3D points are available.

```
let findScale realPoints obs cam =
    let points = Map.intersect realPoints obs

    points |> Map.maxBy ( fun (l,r) ->
        let trans = r - l
        let realCam = cam |> Camera.translated trans

        //checks for minimal reprojection errors
        let score = ransac realCam points

        score
    ) |> apply (fun (l,r) -> r - l)
```

and the code for pose recovery, in the following Code Listing 6.2, where the 'left' camera is presumed to be new and the 'right' camera already reconstructed previously, Listing 6.2: Pose Recovery function. Returns the best possible pose if it exists.

```
let recoverPose leftObs rightObs rightCam realPoints =
    //computes pose candidates
    let poses = findEssentialMat leftObs rightObs ...

    poses |> List.filter ( fun (R,t) ->
        let leftCam =
            rightCam |> Camera.transformed (R,t)

        let points =
            realPoints
                |> Map.intersect (triangulate leftObs leftCam
                                      rightObs rightCam)

        points |> Map.forall (fun p ->
            //all points must be in front of the camera
            Vec.dot (p-rightCam.centre).Normalized
                rightCam.forward > 0.0
        )
    )
    |> List.tryHead
    |> Option.map (fun (R,t) ->
        let scale = findScale realPoints
            (rightCam |> Camera.transformed (R,t))

        R, scale * t
    )
```

Note that we require all existing points to be in front of the camera. If only one point can't be explained by the recovered poses, our confidence in the solution is too low and we discard it. In the case that no previous reconstruction exists, the canonical default values are used instead of calling this function.

Being able to align a reconstruction from two views to an already existing reconstruction from two other views, allows us to effectively chain multiple partial reconstructions together to form one global model of scene structure. We call this process *Structure from Motion*.

CHAPTER 7

Structure from Motion

In Chapters 2 to 5, we have disambiguated methods for extracting and understanding salient parts of two photographs, and applying knowledge about the projection process to recover the two camera poses that took the photographs. These techniques will turn out to be the fundamental building blocks for achieving our original goal, creating a consistent 3D reconstruction from many photographs.

In summary, our approach is based on the recursive solution of individual reconstruction problems and connection of individual solutions to an overarching, maximally consistent frame of reference.

Our starting point is an unordered set of images, and each image pair is a potential candidate for reconstruction. Obviously, completely different image pairs won't yield any feature matches whatsoever, but after filtering those out, a set of partial reconstructions with varying consistencies remains. Figure 7.1 shows an image set, where neighbouring photos should be chosen as reconstruction candidates.

Firstly, we analyze the connections between individual reconstructions and the emergent structures that result from transitive properties thereof. Since we establish hypotheses upon the connectedness of partial solutions using inherently stochastic data, we give special attention to the physical plausibility and even possibility of a potential result. In

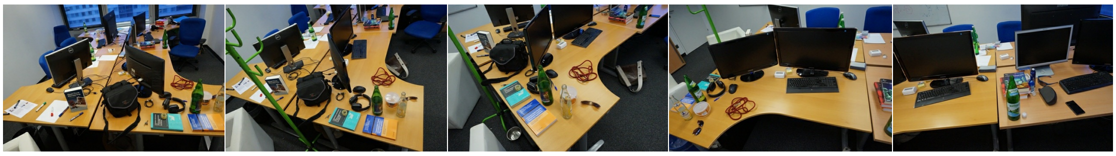


Figure 7.1: An office scene. These photos are suitable for reconstruction because they are rich in features.

practice, we want to track objects across many shots, but the tracking of the object itself may reveal mismatches or measurement errors resulting in logical impossibilities.

Secondly, the resulting connectivity relations establish logical adjacencies between shots, which, when computed appropriately, also correspond to real, physical neighbouring relations between the actual shot locations. We describe pairwise reconstructions in terms of quality measures in order to distinguish strong, likely consistent adjacency relations from weak, unstable ones.

Strong adjacencies are likely to view the same parts of the scene, the same 3D point appearing in a feature across multiple images. Matching these features pairwise each creates sets of equivalent feature tuples, which can be combined to a long chain of matches describing the same 3D point across many images. We exploit this property to establish structure across the entire collection of image pairs, creating an overarching graph structure that brings each camera shot in relation to all the others.

Finally, this match graph is reduced to a maximally consistent tree, which is then traversed recursively, repeatedly performing a pairwise reconstruction between a new camera and an already solved one. The result of this iterative process is a reconstructed network of many photographs that is internally consistent.

7.1 Tracks

In a pair of images looking at the same 3D object, there are (probably) a number of features describing the same 3D point, called feature matches.

The methods for finding matches are discussed in Chapter 4. Specifically, we rely on the feature matching algorithm to produce solid and somewhat physically plausible results. Feature matches should at least be consistent with visual correspondence of the surrounding image patches, as well as the camera motion producing the change in perspective. Only with such a set of reliable feature matches, which is internally free of contradictions, does it make sense to attempt any kind of recovery of 3D information.

Every image is usually part of more than one match. In fact, we initially assume an unordered image set to be a fully connected graph of images, the *Match Graph*, in which every image (node) is logical neighbour (edge) to every other image. By performing feature matching on every neighbouring image pair, these neighbourhood relations are attributed with matches, which double as measure of significance. The higher the number of feature matches, the more similar the visual appearance of the images, the likelier both images are looking at the same object, and the higher the chance reconstruction succeeds. Accordingly, we define the *number of feature matches* as the *stability* of two neighbours, writing them into the edges' weights.

This definition imposes some properties on the matches. Firstly, matches are assumed to be bijective, that is, one feature corresponds to another one as much as vice versa. This might not always be true, for example when the perspective makes many points

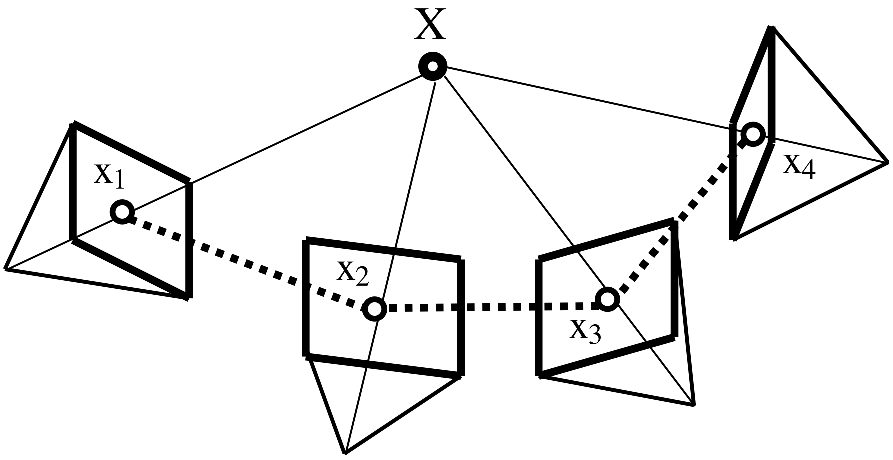


Figure 7.2: A track $[x_1; x_2; x_3; x_4]$ is used to triangulate the point X.

project onto the same feature. Secondly, all matches are assumed to be equally important. Again, when looking at a surface at a very steep angle, the stochastic localisation of a feature becomes very unstable, and even a tiny jitter might produce large differences. These properties are further discussed in Chapter 4. In the scope of this diploma thesis, we rely on the feature matching as well as the pose recovery algorithms to eliminate such outliers.

Similar to how an image connects to several images in the match graph, an individual feature might match to features in several images. This is what introduces overarching 3D structure into our model, since theoretically at least three observations are required to resolve projective ambiguity from a reconstruction (as discussed in Section 6.2).

One feature might match to a feature in a second image, that feature in turn might match to a feature in a third image, then to a feature in a fourth image, and so on. We call such a chain of feature matches a *Track*. Tracks have a *length*, which is the number of features involved in the track. Ignoring degenerate tracks of length 1, the shortest tracks have a length of 2, and the longest possible valid tracks have a length equal to the number of images. A track is shown in Figure 7.2.

The set of all tracks can be viewed as a substructure on the match graph. Each track describes a path within the graph, and the more tracks lead over an edge, the more significant those two neighbours are. In practice, we observe two consequences of viewing points in terms of tracks.

Firstly, we usually want to find the longest tracks possible. Longer tracks are more stable than shorter ones since they confer more witnesses to the real location of a feature, increasing stability of the estimate. In our practical experiments, the length of tracks corresponds directly to the unambiguity of a match graph. If there are longer tracks, the individual images are often more tightly connected and show less overlap with unrelated images. Images that represent physically neighbouring camera locations receive very high weights in the match graph, while distant cameras are close, or equal, to zero. This measure of definiteness, the difference in matches between neighbouring and non-neighbouring cameras, is also a good first measure of quality for a data set, and feature extraction and matching methods.

Secondly, each track ultimately corresponds to one 3D point, and from these tracks we triangulate the final position of a point in a least-squares fashion. Tracks of length equal to or greater than 3 are needed to make triangulation unique, but that doesn't mean that tracks of length 2, usually the highest in number, are useless. As long as their according shots are located and fixed in the world, the triangulation is constrained enough to make it distinct.

We find tracks using a function `mkTracks`, Code Listing 7.1

Listing 7.1: Function generating tracks from feature matches.

```
let extractTracks (pairs : Map<(CameraId * CameraId),
    List<Feature * Feature>>) : Map<TrackId,Map<CameraId,
       Feature>> =

    //traverse camera pairs
    //and find every track
    ...

let mkTracks cameras features getMatches =
    let allPairs =
        List.cross cameras cameras
        |> List.map (fun (left, right) ->
            getMatches features.[left] features.[right]
        )
        |> Map.ofList

    extractTracks allPairs
```

The function considers all camera pairs and their matches, and computes a list of tracks, which are chains of features and their cameras. Our implementation of `extractTrack` simply considers every unvisited match in the set of all matches, and marches along chains of matches until it encounters a different feature from an already visited camera. At that point, it stops and returns the current chain as new track. This implementation is fairly naive because better tracks could be found by incorporating quality measures from the feature matcher. However, it works acceptably well for the scope of this diploma thesis.

In our experiments, for images yielding around 500-3000 features, we want to have at least 40 tracks total and at least 10 tracks of length 3. At least ten times as many are desirable.

7.2 On Contradicting Tracks

The model using match graphs and tracks can exhibit configurations of physical impossibility. For example, a feature in image A matches to one in image B, matches to one in image C, which matches to a different feature in image A. This is a contradiction, because image A now contains two different features observing the same 3D point, which is impossible by definition.

We resolve this by relying on the feature matcher's internal consistency with a physically plausible model. This obviously isn't true for a brute-force matcher operating solely on visual appearance. A common solution is to ensure plausible pieces of information vastly outnumber impossible ones, in which case numerical optimisation takes care of the problem.

In our implementation, however, we prefer to apply our probability and affine matchers (as described in Chapter 4). These matchers ensure that a feature's hypothesised motion is constrained to support every other feature's motions as well, ensure global coherence in the sense of affine transforms. A degree of uncertainty is involved in this assumption due to noisy data, but the resulting deviation is small enough to leave the solution structurally intact. Because our matchers respect an approximate global motion and eliminate ambiguities, the only mismatch that can practically occur is between features that are very close together. In this case, the match is only slightly erroneous and can be rectified during global optimisation.

7.3 Match Graphs

The match graph imposes a structure on our image set. Embedded within this structure lie the actual, physical relations that correspond to real-world distances between shots, which we infer from the number of matching features between images. In practice, larger distances usually mean less stable reconstruction because of less reliable feature matching, so the best candidates for reconstruction are pairs of closest images.

At the same time, we can decide a traversal strategy for the graph by extracting maximum spanning trees over the edges weighted by match counts. Each pair of neighbouring images in the spanning tree is the closest connection to any image found in the entire input set, and therefore the best candidate for reconstruction. In addition, any edge that has not enough, or zero, matches, in which case stable reconstruction becomes increasingly impossible, is dropped from the graph, possibly breaking up the maximum tree into a maximum forest, a set of *Photo Neighbourhoods.* An illustration of a match graph and a photo neighbourhood can be seen in Figure 7.3.

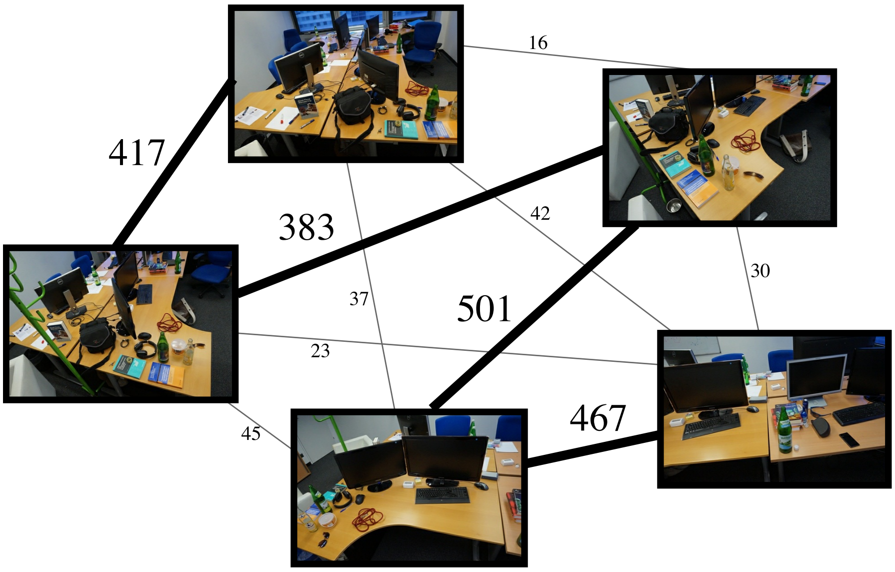


Figure 7.3: A symbolic representation of a match graph. The (example) edge weights are the number of matches. Highlighted with thick lines is the maximum spanning tree.

In our implementation, we use a simple union-find data structure and greedy traversal strategy to find the maximum spanning forest.

These photo neighbourhoods are reconstructed independently, ultimately yielding a *Photo Network*, which can consist of a single or multiple photo neighbourhoods. The traversal order for the spanning trees is simply given by the highest edge weights such that every subsequent result builds on the most stable intermediate result.

7.4 Photo Networks

Given a maximum spanning tree extracted from a match graph, pairwise reconstruction of two adjacent images is done iteratively in order to obtain a photo network, that is, a set of camera locations and triangulated 3D features pertaining to the real-world scenario in which those images were taken.

The algorithm for performing a pairwise reconstruction of two images is detailed in Section 6.2. In particular, the section specifies resolution strategies for the several kinds of emergent ambiguities based on either existing information, if available, or sensible default values, if not. Our maximum spanning tree traversal order is tailored exactly to those resolution strategies, carrying over reconstructed information in a stable a manner as possible from one iteration to the next.

We begin by selecting one particular edge in the match graph as starting point. In the general case, the best choice is the edge with the most matches overall. Since no prior information is yet available, we simply perform the pairwise reconstruction with

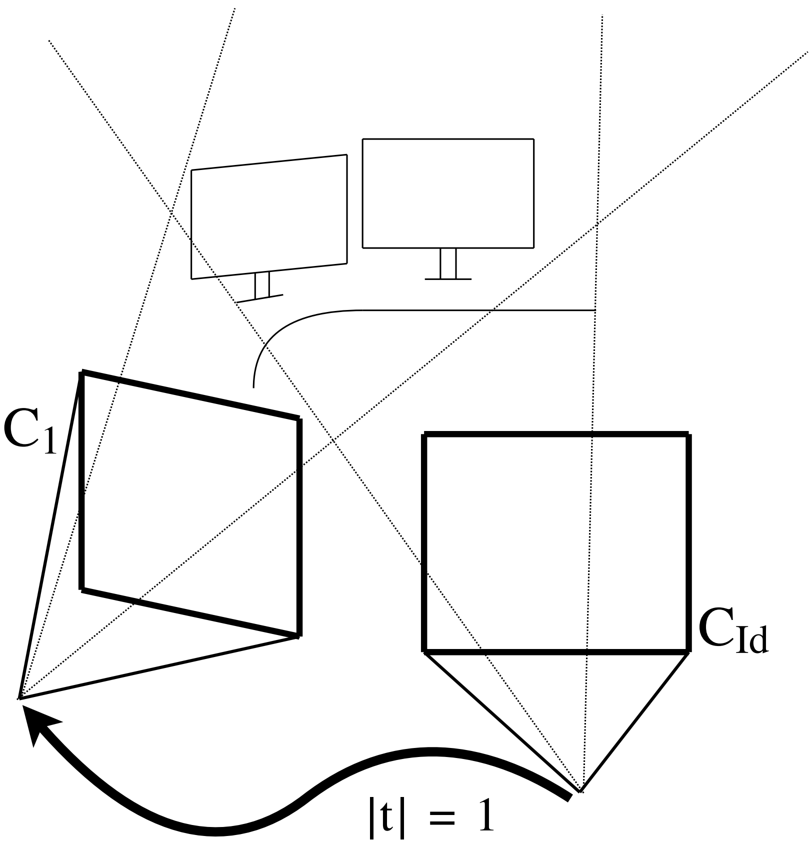


Figure 7.4: In the first iteration, the identity camera C_{Id} and a baseline length of $|t| = 1$ are used as initial configuration for a photo network. The pose for C_1 is recovered using the parts of the scene that appear in both cameras.

appropriate default values. Pose recovery for the two cameras returns the set of possible poses to explain their observations. From this set, we select one that suits us most, which is the one where the 3D points are in front of both cameras and the length of their baseline is a constant value (such as 1).

We designate this result as the first photo network. The first iteration is illustrated in Figure 7.4.

Any evaluation of the photo network might change after every iteration, making the entire process highly recursive. Any tracks too short for triangulation, and any unused observations, are retained without change, since they might become long enough, stable enough, or generally useful once a later camera is added.

Additionally, we re-calculate all reprojection errors in every iteration, the sum of which serves as a measure of improvement over the course of reconstruction. We call this value

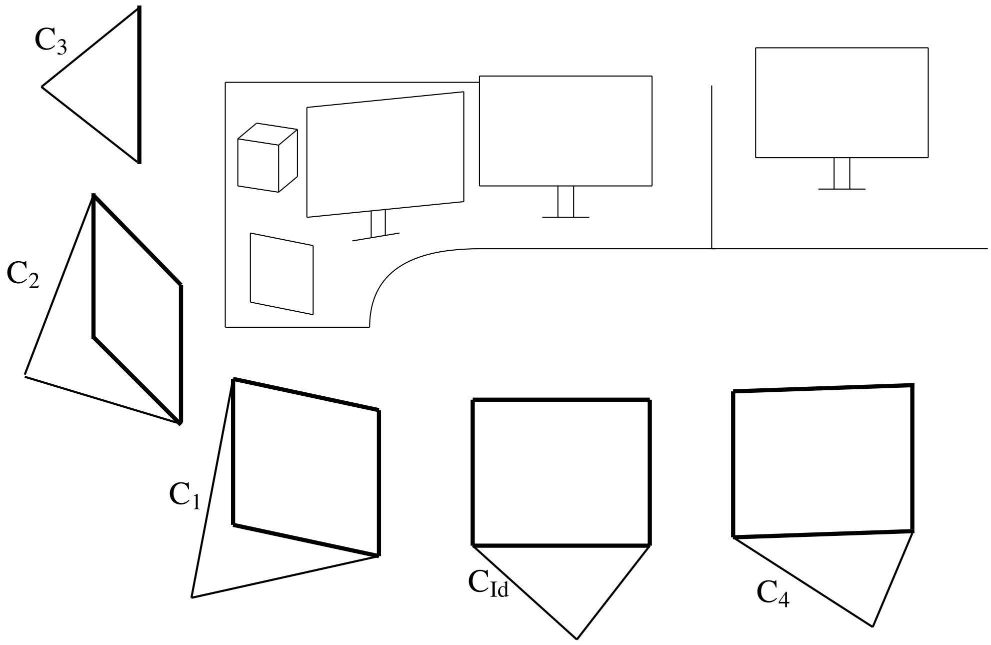


Figure 7.5: Over the course of multiple iterations, one camera is successively added to the photo network. The part of the scene visible by both the camera and the network is used for pose recovery.

the *Cost* of a photo network.

From here on, every step of the reconstruction is aware of the results produced in the previous iterations.

Obviously, this makes the default values chosen in the first iteration rather significant, since they impose the starting orientation, scale and shape of the entire reconstruction. For this reason, to improve global stability, we single out the iteration in which a third camera is added to a photo network with two cameras. In this case, the existing solution is destroyed and both pairs of cameras are reconstructed again. The pair with the lower cost is the one that fits the default values more naturally and is selected as real starting edge. For better readability, we omit this optimisation from the code given in Code Listing 7.3.

A new camera is added to an existing photo network as follows. First, the candidate is chosen from the match graph nodes adjacent to the already reconstructed part, being the one with the most matches. The candidate is reconstructed together with its neighbour, where the neighbour's already existing 3D position and orientation are used as basis for ambiguity resolution in orientation and scale. The length of the translation component is recovered from the set of the photo network's 3D points intersected with the observations of the new camera. The result is a photo network that contains the previous cameras as well as the new one, as well as all triangulated tracks.

Addition of a camera might either succeed or fail, and it fails exactly if pose recovery fails, which is either when there are too few matches between the camera pair, too few matching observations and 3D points, or pose recovery is impossible due to a geometrically undecidable/degenerate situation. In any case, these failed cameras are collected across iterations. After every successful iteration, we attempt to add every failed camera to the current photo network once more, since the new photo network might now permit addition. One iteration of the reconstruction is therefore comprised of multiple camera additions. We refer to such a step as *Epoch*. The result of an epoch is illustrated in Figure 7.5.

Reconstruction is continued until the entire match graph has been traversed. Only after the last epoch is finished do we arrive at the intermediate result, which is a fully reconstructed photo network and a set of remaining cameras which truly don't belong to that network. For the remaining cameras, a new photo network reconstruction is started. This is repeated until there are no more remaining cameras. Our implementation is given in the following Code Listings 7.2, 7.3, 7.4 and 7.5.

We define a photo network as the type

Listing 7.2: Photo network type.

```
type PhotoNetwork =
    {
        config : PhotoNetworkConfig
        cost : float
        cameras : Map<CameraId, Camera>
        points : Map<TrackId, V3d>
        measurements : Map<CameraId, Map<TrackId, Feature>>
    }
```

where the empty photo network simply has empty sets of cameras, points and measurements and a cost of zero. We add a camera to a photo network with

Listing 7.3: Adds one camera to a photo network.

```
let addCamera cid cam obs network =
    { network with
        count = network.count+1
        cameras = Map.add cid cam network.cameras
        measurements = Map.add cid obs network.measurements
        points = triangulate cam obs network
    }

let tryAddFromParent parent cid obs network =
    match recoverPose parent obs network with
         | None -> None
         | Some (R,t) ->
            let cam = parent |> Camera.transformed (R,t)
            addCamera cid cam obs network |> Some

let tryAdd (cid : CameraId) (obs : Map<TrackId, Feature>)
            (network : PhotoNetwork) =
    match network.count with
    | 0 -> addCamera cid rootCam obs network |> Some
    | 1 ->
        let parent = network.cameras.Head
        tryAddFromParent parent cid obs network
    | _ ->
        let parent =
            //the candidate with the most matches
            network.measurements
                |> Map.maxBy(fun (_,ms)
                        ms |> Map.intersect obs
                    )
        tryAddFromParent network.cameras.[parent.Key] cid obs network
```

where `tryBestPose` finds the best hypothesised pose to fit an existing camera, `tryAddFromParent` adds this pose to the result if it exists and signals failure if it does not, and `tryAdd` finds a suitable parent to estimate the essential matrix from.

`tryAdd` is called in every recursion step. The recursion step, named `epoch`, is the recursive attempt to add all cameras to one photo network. The implementation is given by the Code Listing 7.4

Listing 7.4: Tries adding all cameras to a photo network.

```
let rec epoch network observations added notAdded =
    match observations with
    | [] ->
        if added > 0 then
            epoch network (List.rev notAdded) 0 []
        else
            network, added, notAdded
    | (cid,obs)::remaining ->
        match tryAdd cid obs network with
        | Some net ->
            epoch net remaining (added+1) notAdded
        | None ->
            epoch network remaining added ((cid,obs)::notAdded)
```

in which `epoch` attempts to compute a camera from every set of observations, and adds it to the photo network if successful, or adds it to the list of unsuccessful attempts otherwise. Once no more observations remain, the procedure starts anew with the list of unsuccessful attempts as input. In our experiments, most of these attempts turn successful in later iterations. The function terminates as soon as no change happens in an epoch, and returns the photo network plus the remaining unsuccessful observations.

Finally, a new photo network is started for the remaining observations recursively, Code Listing 7.5,

Listing 7.5: Creates photo networks until all observations are consumed.

```
let rec addMany networks observations =
    match observations with
    | [] -> networks
    | _ ->
        let (net,remaining) =
            epoch PhotoNetwork.empty observations 0 []
        addMany (net::networks) remaining

let createPhotoNetwork matches =
    //maximum spanning tree over image pairs
    //returns observations ordered by match count
    let observations = spanningTree matches ...

    addMany [] observations
```

where `createPhotoNetwork` of type `List<CameraId * Map<TrackId * Feature» -> PhotoNetwork` computes a photo network for a given set of tracks.

The final result is the collected set of photo networks. The photo networks are independent of each other, and each photo networks represents the internally most consistent recovery of 3D scene structure from a part of the input.

7.5 Application

In practice, we identify some typical structures of photo networks:

- A single, large photo network: Desirable, but difficult to achieve in most real-world scenarios. Input image sets typically aren't perfect enough to permit a single coherent photo network if no special precautions have been taken. In such cases, feature matching can be made less precise to create a connected photo network, albeit with higher cost.
- Several large photo networks: This is indicative of a particular spot in the real world that hasn't been captured or can't be represented well enough to be contiguous with the rest of the scene (for example, building corners or obstructive plants). The match graph can be used to identify such 'weak links' in the image set, which can be exchanged with more and better photographs.
- Many small photo networks: The photo network disintegrates into many disjoint sets. This often means that feature extraction or feature matching happened in a fashion inappropriate for this particular image set, and too few matches were produced. Trying out different techniques might alleviate this problem, though an image set might also simply be unreconstructible (for example, shiny materials).
- Photo networks with 1 or 2 cameras: No scene information could be reconstructed from these images.

Ultimately, the user has to decide what result is deemed acceptable. The quality of a reconstruction depends very highly on the quality of the input data set. Whereas a good, highly structured and visually unique image set will result in an equally good photo network, photos of a white wall, for instance, will produce no useful result whatsoever. Good input - good result.

In addition to the input data, the method of feature extraction and feature matching, which are our representation of the real world, are equally important. The keypoint features we use are well qualified for some applications, for example a mountain, but entirely unsuitable for others, for example a billiard ball. We specify our reconstruction library in a modular and functional fashion with the express intent to try out different features and see which one works best.

In either case, the solution is structurally intact, but carries the cumulative measurement errors from the necessarily iterative construction. It makes sense to process the photo network globally, slightly adjusting every element to minimise the cost and remove this systemic drift. We refer to this step as *Bundle Adjustment*.

CHAPTER 8

Bundle Adjustment

In practice, we have to respect the effect of measurement errors in conjunction with Structure From Motion. These errors are introduced by imaging artefacts, uncertainties in feature extraction, and mismatches. Though we can expect there to be more correct than incorrect information, and the error for an individual pose recovery is minuscule, these small errors accumulate over the course of iterative reconstruction, introducing a systematic drift that may grow very large for particular inputs.

We alleviate this problem with a global optimisation step called *Bundle Adjustment*.

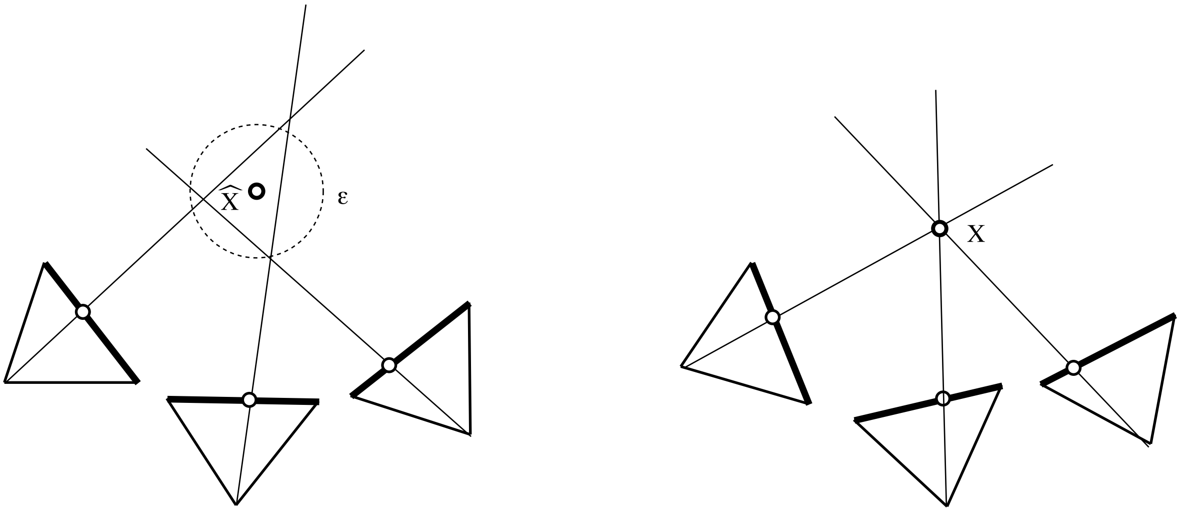


Figure 8.1: Bundle adjustment. *Left*: Photo network containing numeric errors ϵ for a triangulated point $\hat{x}$. *Right*: Bundle adjusted photo network, in which the triangulated point x has numeric errors close to zero.

In principle, our global optimisation works simply by minimising the reprojection errors for all cameras, that is, the pixel difference between the feature positions and the locations of their corresponding 3D points projected back onto the image, in a least-squares sense. An illustration of this process can be seen in Figure 8.1. However, the mathematical complexity of this particular problem poses some additional constraints.

Firstly, our system of equations has unusually many degrees of freedom. Each camera has a location and orientation in 3D space, both of which may change in order to better fit the points they aim to explain, and every individual 3D point is free as well, moving to best possibly conform to the cameras observing them. The consequences are twofold.

On the one hand, the system requires an appropriate number of data points to arrive at any workable solution. While in theory it suffices to have four 3D points described by three observations each, in practice we want to have about ten times as many in order to include the points with only two observations as well, whose hypothesised topology can then become a data point.

On the other hand, the solution is highly sensitive to geometric degeneracy. Specifically, no 3D point should be evidenced only by observations whose view rays happen to be almost parallel. Also, no point can be behind a camera that it should be in front of, which is mainly caused by exceedingly many mismatches.

Both of these problems can be avoided by performing reconstruction, that is feature extraction and matching, as accurately as possible. Consequently, the result of pose recovery must be already **very close to the final result**. Bundle adjustment's only purpose is to make *very slight* adjustments to an already finished reconstruction, and then terminate immediately.

It can be tempting to attempt 'saving' a failed reconstruction by letting the solver adjust the solution structurally, but that often only leads to useless outcomes, such as arbitrary camera poses or the entire solution collapsing into a single point. It is for exactly this reason that we stressed logical soundness and accuracy of the entire reconstruction procedure up to this point.

Secondly, the projection function is not linear. Accordingly, we need to employ numeric solution techniques for systems of nonlinear equations in order to arrive at a solution satisfying our problem statement. Such solution techniques are far from trivial to implement, and algorithms must be fairly refined to work in real-world scenarios. This is why we decided to use a publicly available solver library, Google Ceres [AM+12], to handle this task.

We wrap this library with some convenience mechanisms to allow for a simple and precise problem formulation, and then dispatch one call to have it perform the optimisation step for us, which employs, in principle, a robustified gradient descent algorithm.

The result is a photo network which is similar to the input, but every camera and point is slightly adjusted to best possibly support each other's observations.

8.1 Global Optimisation

The equation for projecting a single point into a camera is

$$\underbrace{\begin{pmatrix} x \\ y \end{pmatrix}}_{\text{observation}} = \underbrace{\mathbf{K}}_{\text{camera matrix}} \times \underbrace{\mathbf{t}}_{\text{3D translation}} \times \underbrace{\mathbf{R}}_{\text{3D rotation}} \times \underbrace{\begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}}_{\text{3D point}}$$

While the matrix vector multiplication itself is linear, the perspective division, which grounds the resulting projective equivalence class of solutions into one specific value with Euclidean meaning, introduces a nonlinear component into the system.

Every point has been triangulated from a corresponding observation in an image. The reconstruction is without error if the distance between a point's projection x_i and its observation x_i' is zero.

$$\|x_i' - x_i\|^2 = 0$$

One point is observed by C many cameras, and the point should perfectly project onto all of them simultaneously

$$\sum_{c=0}^{C} \|x_{i_c}' - x_{i_c}\|^2 = 0$$

And finally, we have N points in our system. Because the observations usually don't have perfect accuracy, we want to minimise the value of the expression

$$\sum_{i=0}^{N} \sum_{c=0}^{C} \|x_{i_c}' - x_{i_c}\|^2 \to \min$$

The project function, including its derivative, is nonlinear, preventing us from writing this as a linear system of equations and solving directly. Instead, this expression is minimized using gradient descent based techniques.

8.2 Nonlinear Least-Squares Solving

The basis of nonlinear least squares is approximating a nonlinear function locally with a linear one and iteratively stepping along its gradient, progressively refining the parameters as we go to make the approximation more accurate. An example for this technique is Newton's method, which evaluates the function's derivatives locally using Taylor

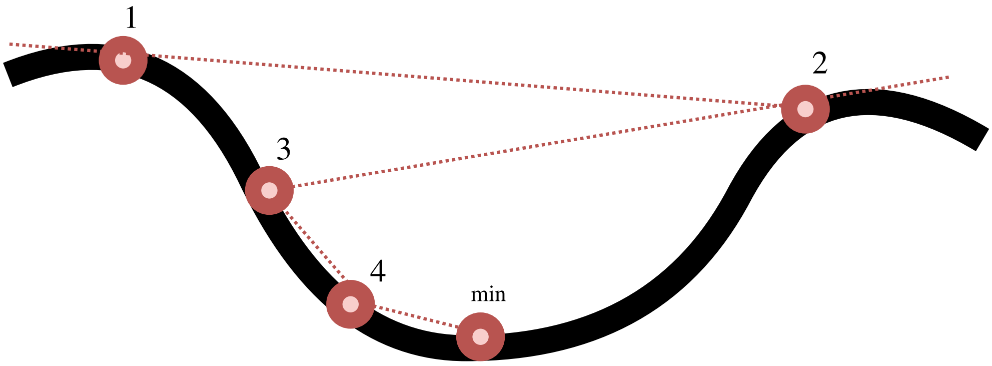


Figure 8.2: Gradient descent algorithm. In every step, the direction of the smallest gradient is followed until the minimum min is reached. This numeric technique is employed when analytical derivation is costly or impossible.

approximation, steps in the derivative's direction, and repeats this process until the derivative is horizontal and an optimum is found. This technique is illustrated in Figure 8.2.

This simple approach gets stuck in local extrema easily, which is why significantly more sophisticated techniques exist that play around with appropriate evaluation strategies and starting values.

There are several solution strategies for such a system. The gradient descent based approaches can be categorised according to how they process the gradient and make optimisation steps. The *line search* methods first determine the direction along which to reduce the objective function using gradient information, and then compute a distance. On the other hand, *trust region* methods first estimate the optimal distance of a step based on the quality of previous iterations, before computing a direction.

For larger reconstructions, which contain many cameras viewing only parts of the scene, the Jacobians, the partial derivatives, can get extremely sparse, since every point and camera has a derivative of zero in almost every other point or camera. Therefore, the solver usually needs to include mechanisms to deal with such extreme sparseness, such as sparse matrix representations, preconditioning and parameter block reordering.

Google Ceres already includes solution strategies and algorithmic options tuned for sparse bundle adjustment problems. We wrap the library with our own memory manager, allowing us to define arbitrary data types as long as we also provide serialise/deserialise functions for native memory. We use this mechanism to define the three dimensional vector `V3d` for points, and the camera type `Camera3d` consisting of an offset vector along with an orientation in axis-angle representation.

For every parameter, we calculate residuals and Jacobians in an evaluation step. While Ceres provides numeric differentiation, we calculate precise derivatives ourselves by implementing *autodiff* [Fou96]. In short, autodiff (Automatic Differentiation) means that every operator also carries along its derivative. Therefore, every expression also causes a composition of derivatives, resulting in the exact gradient at the position of the result.

While this technique is complex for arbitrary expressions, implementation is straightforward for the few operators we need.

A single floating point number is represented by tuple of value and the Jacobian within the current computation. We call this type `scalar`. Scalar addition, subtraction, multiplication and division simply use their respective rules of derivation and compose the result's Jacobians accordingly. These types are shown in Code Listing 8.1.

For convenience, we declare types that group multiple `scalars` together. The `scalar` equivalent of a three dimensional vector `V3d` is called `V3s`. Similarly, the camera type `Camera3d` receives a `scalar` equivalent `Camera3s`. Vector products and vector-matrix multiplications carry their Jacobians representing derivatives in the individual dimensions. Using these lifted operators, we are able to idiomatically express our reprojection error cost function in `scalar` world.

What remains are the residuals: the *Cost Function*. The cost function goes from the current parameter values to a sequence of scalars. The function is called from the Ceres iteration callback, and the scalars directly represent the required residual values and Jacobians. For this application, it suffices to specify residuals directly, since our expressions aren't particularly complex.

8.3 Application

Our solver types are built using the `Jacobian<'a>` type, in the following Code Listing 8.1.

Listing 8.1: Types implementing autodiff [Fou96].

```
type Jacobian<'a> = Map<int,'a>

//maps a jacobian with f
let map f l =
    l |> Map.map f

//combines two jacobians using f
let zip f l r =
    Map.zip (fun (l,r) -> f l r) l r

let add l r =
    zip (+) l r
```

```
let subtract l r =
    zip (+) l (map (-) r)

//multiply, divide, etc.
...
```

and the `scalar` type

```
type Scalar =
    {
        Value : float
        Jacobian : Jacobian<float>
    }

let (+) l r =
    {
        Value = l.Value + r.Value
        Jacobian = Jacobian.add l.Jacobian r.Jacobian
    }

let (-) l r =
    {
        Value = l.Value - r.Value
        Jacobian = Jacobian.subtract l.Jacobian r.Jacobian
    }

let cos v =
    {
        Value = cos v.Value
        Jacobian = Jacobian.multiply (- sin v.Value, v.Jacobian)
    }

//other operators
...
```

Vector, matrix and camera types are defined equivalently.

Using these types and operators, we define the cost function in the following Code Listing 8.2

Listing 8.2: Cost function for bundle adjustment. This syntax hides our autodiff implementation.

```
let costFunction real cam point =
    let obs = Camera.project cam point
    [
        obs.X
        obs.Y
    ]


let createCeresProblem points cameras tracks =
    let prob = Ceres.problem()

    //adds every point as free parameter
    let pointParams =
        points |> Map.map prob.AddParameter

    //adds every camera as free parameter
    let camParams =
        cameras |> Map.map prob.AddParameter

    //adds a cost function for every observation,
    //referring to the free parameters
    for (pid,t) in tracks do
        for (cid,real) in t do
            let cost = costFunction
                        real
                        camParams.[cid]
                        pointParams.[pid]
            prob.AddCostFunction
    prob
```

Thus equipped, we call Ceres to perform the optimisation, like in the following Code Listing 8.3

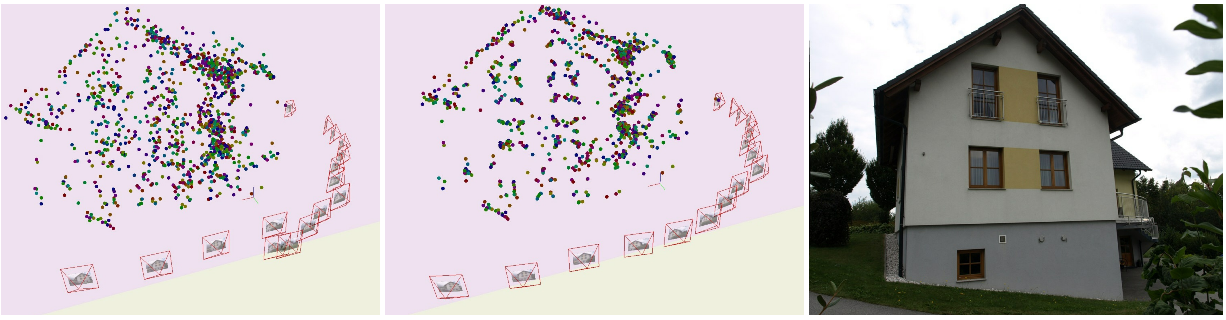


Figure 8.3: Bundle adjustment. *Left*: Before bundle adjustment. *Centre*: After bundle adjustment. *Right*: Photo of the scene.

Listing 8.3: Bundle adjustment with Ceres.

```
let bundleAdjust network =

    use prob = createCeresProblem
                 network.points
                 network.cameras
                 network.measurements

    //dispatches call to Ceres, filling in our values,
    //jacobians and residuals
    do prob.Solve()

    //reads the contents of Ceres memory locations
    let (points,cameras) = getResult prob

    { network with cameras = cameras; points = points}
```

The effect of bundle adjustment is shown in Figure 8.3. Without bundle adjustment, triangulated points and camera locations are unstable, corners and contours appear fuzzy. The optimisation technique adjusts the model to resemble the real scene more plausibly.

CHAPTER 9

A Composable and Reusable Photogrammetric Reconstruction Library

A major goal we set out to achieve with our photogrammetry library is to enable rapid experimentation and flexible extension. We want to be able to rearrange components freely, introduce new ones or leave some out, and, especially, substitute our own. Specifically, implementing an alternative feature extractor and feature matcher pair is of great interest due to the highly domain specific nature of data representation.

For instance, one could imagine using different kinds of point features apart from those we have used, or even different kinds of features entirely, such as line features or rectangle patch features. Similarly, the implementation of different modes of operation should be possible. We could implement machine learning techniques to improve feature extraction and matching, in a way to ignore intrusive vegetation for example. Having the freedom to exchange and re-implement modules is crucial to this end.

Historically, most existing photogrammetry libraries suffer from massive code complexity. This stems in part from how complicated a photogrammetric reconstruction inherently is, but, quintessentially, also from how much experimentation and evaluation is required to create a reconstruction setup suitable for a particular problem.

Free and open source reconstruction frameworks tend to have been developed with a specific use case in mind. Domain-specific understanding and optimisations are in place, such as, for example, operating exclusively on line features for specific buildings, using weak feature matching algorithms assuming a huge number of photos, or restricting cameras in motion and pose. Accordingly, these specialisations make it exceedingly difficult, and sometimes even impossible in practice, to alter such a reconstruction pipeline for one’s own purposes.

Paid and commercial photogrammetry applications usually lack the transparency required for alteration in any meaningful way. This is exemplified by the fact that tests and evaluations often consist only of fixed inputs and expected outputs for the entire pipeline, but seldom for individual components. The reconstruction pipeline is viewed as a 'black box' designed to work for a given set of configuration parameters, but not allowing much structural flexibility. Obviously, extending such a reconstruction program by novel features requires considerable effort, being exerted in the form of ongoing updates by increasingly large commercial software engineering teams. Their tools might also be difficult to use because specifications aren't freely available.

We propose a different approach. The reconstruction library implemented in this diploma thesis is neither a framework nor an application, but a *library.* Every piece of functionality is absolutely modular, there are no interdependencies between functionalities located in separate modules. We greatly emphasise logical and mathematical soundness in the deduction of each module to ensure independent testability and exchangeability. Every input is minimal, and our proposed implementations use small parameter sets that are consistent in units and usages.

Finally, the consumer API is exposed in a useful, extensible and composable way. We heavily borrow concepts from Functional Programming (FP) to encourage a clean coding style. Functionalities are modelled using higher-order functions and function composition. High-level functions are kept pure, meaning that the same input parameters will always evaluate to the same result value. They are also fully parameterised, meaning that every function explicitly receives all inputs it needs, regardless of whether they are user inputs or results of other functions.

Consequentially, every module is practically a standalone command line tool which can used, experimented with and improved on its own. A photogrammetry pipeline is created by function composition, giving the process a sequential computation characteristic. As such, usual control flow constructs like branching and recursion are trivially available.

The external tools we use, OpenCV and Ceres, pose the only exception to this line of thought. Although our wrappers present themselves as pure, modification of the same is difficult. Ultimately, we would like to substitute our own implementations in their stead, but doing so would go beyond the scope of this diploma thesis. This task is left open as future work.

By respecting functional programming experience in API design, we reap the typical FP advantages:

- modularity
- composability
- independent, reusable components
- testability, maintainability

- type safety, documentation through types
- extremely concise and readable code

Coupled with clean conventions, minimal parameters and a sensible code structure, our photogrammetry library should be truly composable and extensible, as basis for experimentation and further research.

9.1 Input Photos

Since a photogrammetric reconstruction can only ever be as good as the input photographs, we would like to briefly touch on the subject. In order to obtain a useful result, the photogrammetry input must fulfil some requirements.

Firstly, the internal camera parameters must be known and photos should be undistorted. In our experiments, we employed the OpenCV algorithms discussed in Section 5.4 to calibrate our cameras. Typically, a camera is calibrated once before taking any photos, and every subsequent photo is required to be taken with the same calibrated camera (i.e. use the same objective and don't zoom). Given that the internal camera parameters remain constant, we can do photo undistortion (using the same OpenCV functionality) as a global preprocessing step.

Secondly, photos must be suitable for reconstruction. This is probably the most difficult and least formalised requirement. Photos must be taken in such a way to permit reconstruction in the first place, which means that every information we want to reconstruct must be represented in the input at high enough a fidelity to permit the desired accuracy. Every photo can be thought as a sample to the camera's 3D motion, and there need to be enough samples overall to accurately recover that motion. This is especially critical in situations where a small camera movement produces large changes in perspective, such as around the corner of a house. Some experimentation and experience is required to get this right, and the accuracy ceiling is high. Good and bad examples of input photos are shown in Figure 9.1 and Figure 9.2.

9.2 Library API

On our way through the previous chapters of this work, we have accrued a collection of function definitions that comprise the main functionality of our library. In this section, we compose these functions together to obtain a high level of abstraction for our photogrammetry library, and present end-to-end example usages thereof.

We use Aardvark, the open source visual computing platform [Aara], for image loading and processing (type `PixImage`). We also use its map and tree implementations and matrix operations due to their efficiency and ease of use.

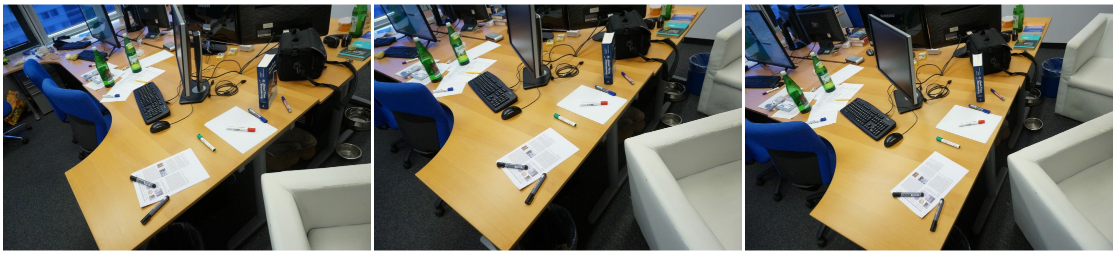


Figure 9.1: Good photos for photogrammetry. The scene is rich in features, and the camera makes small steps to capture crucial detail.

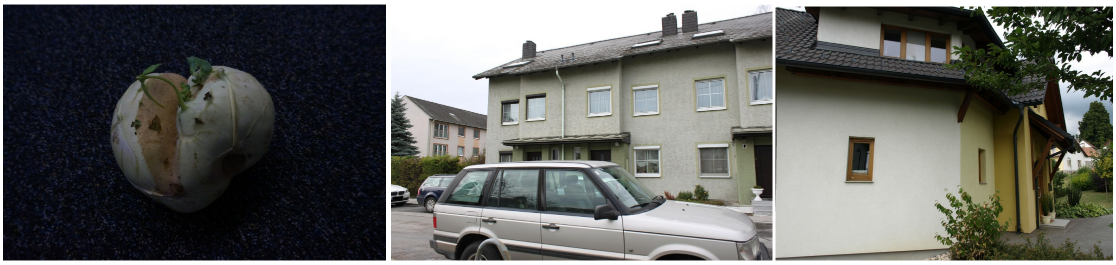


Figure 9.2: Bad photos for photogrammetry. *Left*: The vegetable has too few features and the photo is dark. *Centre*: A shiny, reflective car is blocking the subject, which makes feature matching problematic. *Right*: The plants are swaying in the wind, destroying models of rigid motion.

Aardvark is an open source library for high-performance rendering and visual computing applications. Aardvark implements a number of techniques enabling incremental computation and reactive programming down to the level of graphics hardware [HSMT15] [HSM$^+$14], achieving very high rendering performance. On top of maths, geometry and data structure utilities, Aardvark provides mechanisms for declaring scenes and applications in a declarative fashion, using embedded domain specific languages.

After loading images, we want to extract features from them. A feature extractor takes an image as input and computes a set of features from it. Specific concrete feature extractors are obtained through partial application of the parameters, as demonstrated in Section 3.3.7.

In our experiments, we used some well-known image features provided by OpenCV: SIFT, SURF, BRISK, AKAZE and ORB. To stay within the scope of this diploma thesis, the implementation of structurally more complex features, such as line features, is left as future work. However, our library is generic in its feature kind to support such extension.

Next, features are matched across multiple images. A feature matcher computes the set of corresponding features, returning them as pairs, from two input feature sets. Different feature matchers are presented in Section 4.7.

Since feature matching is transitive, we can find all connected chains of matches, the tracks. We find tracks using the `mkMatches` function discussed in Section 7.1. This is the function which actually calls the feature matcher for every new image pair it encounters.

The function `createPhotoNetwork` given in Section 7.4 is called with the resulting tracks. Starting with the empty photo network, the algorithm extracts a maximum spanning tree from all image pairs, and orders those image pairs by the number of tracks they contains. Then it adds cameras one-by-one to the existing solution. It terminates when all cameras belong to one photo network.

Finally, the resulting photo network can be subjected to bundle adjustment in order to rectify the cumulative measurement errors agglomerated by recursive iteration. The function `bundleAdjust`, as presented in Section 8.3, is used for this purpose.

All data structures used in our reconstruction library are persistent, and computations are implemented as pure functions with composable signatures. A simple end-to-end photogrammetry pipeline is obtained by function composition. An example reconstruction pipeline could look like in this Code Listing 9.1.

Listing 9.1: Simple reconstruction pipeline.

```
let photonetwork =
    @"C:\undistortedPhotos"
        |> Directory.GetFiles
        |> Array.map (fun file -> PixImage.Create file) |> Map.ofArray
        |> Map.map detectAkaze
        |> mkTracks manyMatches
        |> PhotoNetwork.create
        |> List.map bundleAdjust
```

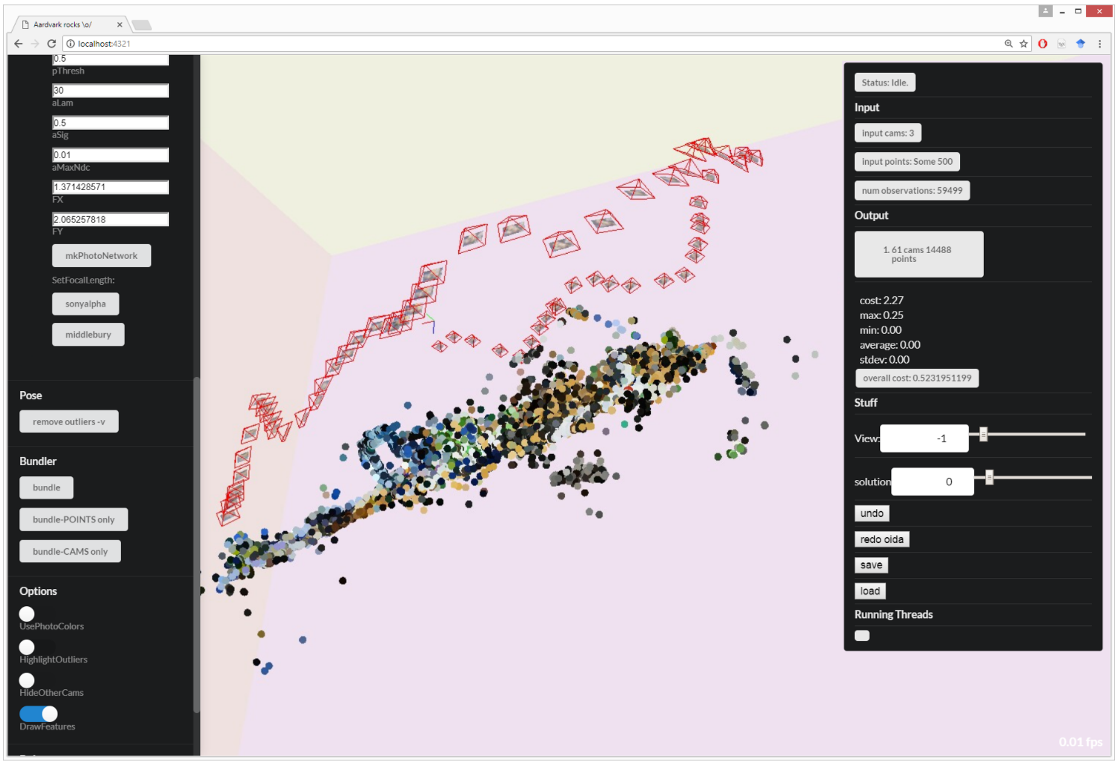
The code is intended to be clear and readable. Individual functions can be replaced while leaving the remaining intact. Maintaining the simplest type signatures encourages higher level abstraction.

For example, our input and output types are at least monoids, which means that they are guaranteed to support the typical set aggregation operations. Let us only keep photo networks with at least 3 cameras using `filter`

```
let atLeastThree nets =
    nets |> List.filter (fun phn -> phn.cameras.Count > 2)
```

As another example, let us stitch a collection of separate camera groups together to a single group using `fold`

```
let connect left right =
    match left with
    | [] -> right
    | _ -> let trafo = //calculate stitching trafo
```



pThresh: 0.5
aLam: 30
aSig: 0.5
aMaxNdc: 0.01
FX: 1.371428571
FY: 2.065257818

Status: Idle.
input cams: 3
input points: Some 500
num observations: 59499
1. 61 cams 14488 points
cost: 2.27
max: 0.25
min: 0.00
average: 0.00
stdev: 0.00
overall cost: 0.5231951199
View: -1
solution: 0

Figure 9.3: A possible photogrammetry user application. This application was coded in a way similar to Code Listing 9.2 using the library for interactive apps Aardvark.Media [Aarb].

```
            computeStitch left.cameras
                (right |> List.collect (fun phn -> phn.cameras)
        PhotoNetwork.add (left |> PhotoNetwork.transformed trafo) right
let stitch photoNetworks =
    photoNetworks
        |> List.fold connect PhotoNetwork.empty
```

Finally, we could create a simple user interactive program by creating a mutable variable containing the program state, and computing a new program state every time the user dispatches an action. Such an application is outlined in Code Listing 9.2.

For this diploma thesis, we implemented a simple interactive reconstruction app for the purpose of exploration and experimentation. The app is built using Aardvark.Media [Aarb], a library for high-performance interactive visualisations, part of the Aardvark platform. The code looks similar to the sketch in the code listing, but is more performant. A screenshot is shown in Figure 9.3.

Listing 9.2: A simple sketch for a photogrammetry user application.

```
type Model =
{
    Photos : List<PixImage * List<Feature>>
    PhotoNetwork : Option<PhotoNetwork>
}

let mutable state = { Photos = []; PhotoNetwork = None }

let view state =
    [
        button Action.LoadPhotos
        button Action.CreatePhotoNetwork
        button Action.BundleAdjust
        sceneGraph state.PhotoNetwork
        ... // additional UI elements producing Actions
    ]

let update state action =
    match action with
    | Action.LoadPhotos ->
        { state with Photos =
            @"C:\undistortedPhotos"
                |> Directory.GetFiles
                |> List.map (fun photo ->
                    photo, detectAkaze photo
                )
        }
    | Action.CreatePhotoNetwork ->
        { state with PhotoNetwork =
            state.Photos
                |> mkTracks manyMatches
                |> PhotoNetwork.create
                |> atLeastThree
                |> stitch
                |> Some
        }
    | Action.BundleAdjust ->
        { state with PhotoNetwork =
            state.PhotoNetwork
                |> Option.map bundleAdjust
        }

async {
    while programIsRunning do
        do setUserInterface view state
        let action = waitForActionFromUserInterface()
        do state <- update state action
} |> Async.Start
```

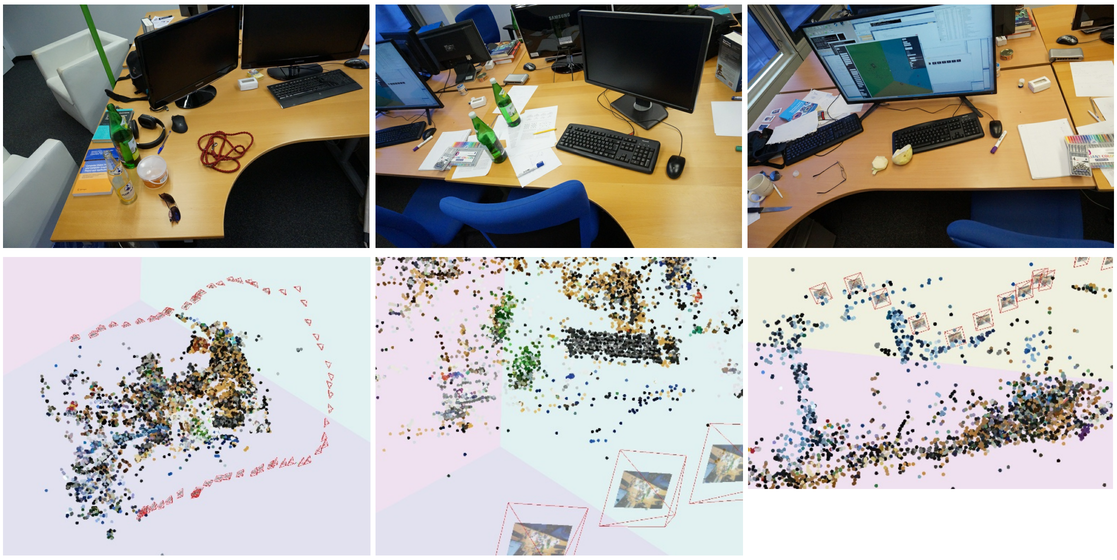


Figure 9.4: Reconstruction of an office scene.

9.3 Results Discussion

In the course of this diploma thesis, we implemented some basic tools to evaluate the individual components of a photogrammetry pipeline. These include a feature matcher parameter tuning app, screenshots seen in Section 4.7, an app for parallel evaluation of many parameters for a photogrammetry pipeline (no visualisation), and the interactive photogrammetry tool shown in the previous Section 9.2. We profited greatly from the functional programming style we adhered to, notably the complete input and output parameter set that is required of every function.

We employed parallelisation where applicable, which proved to be simple for our immutable data structures using the parallel loop and parallel reduction mechanisms included in .NET. Similarly, the immutable data structures made caching for long-running computations straightforward: we simply serialised computation results into files with the hash of their input parameters as name, using the generic value serialiser FsPickler [Tsa13].

These tools were invaluable for rapid iteration upon our library. On a machine with an Intel i5 CPU running Windows 8, our longest runtime was about one hour for a scene with 150 photos and at least 500 features each, where the most time was spent in feature matching. Some example photogrammetry results follow.

9.3.1 Office Scene

These photos were taken in our office using a regular consumer camera. The scene has many features, few shiny or reflective objects and good lighting. The reconstruction consists of 61 cameras and about 16000 points. We used AKAZE features and the `manyMatches` matcher (Section 4.7). Some impressions are shown in Figure 9.4.

9.3.2 Artificial Scene

This is part of a test data set for the evaluation of multi view stereo systems provided by, and available at, `vision.middlebury.edu`[SCD$^+$06]. We used the front half of the 'temple' data set for reconstruction, which is a small model of a temple arch sampled regularly from every angle using a robot arm. ORB features proved superior to AKAZE in this case because of how natural the object appears. The reconstruction (Figure 9.5) resulted in 150 cameras and 6000 points.

9.3.3 House Scene

This scene is reconstructed from photographs of a house. We used AKAZE features and the `veryStable` matcher. However, we were unable to reconstruct the entire house as one coherent photo network because of plants blocking the view of the corners. The reconstructed photo network consists of three parts of about 15 cameras and 2000 points each. The plants make this data set a 'worst case' as it were. In our experiments, no other tool was capable of fully reconstructing this scene either, with the sole exception of Agisoft Photoscan, whose quality is difficult to gauge due to the lack of insight into the program code.

Some impressions are shown in Figure 9.6. The photos are courtesy of rmData [rV].

9.4 Present State

The photogrammetry library presented in this diploma thesis aims to be a useful tool for methodical exploration and iteration in the hands of the scientific community. Our implementation may be freely used for scientific purposes. The source code and assorted utility programs and documentation are available on github at `https://github.com/aardvark-platform/aardvark.mondo`.

The author of this diploma thesis[1] intends to continue improving on this project, and would be very grateful for comments, results, data sets and any other form of feedback.

---

[1] `https://github.com/aszabo314`

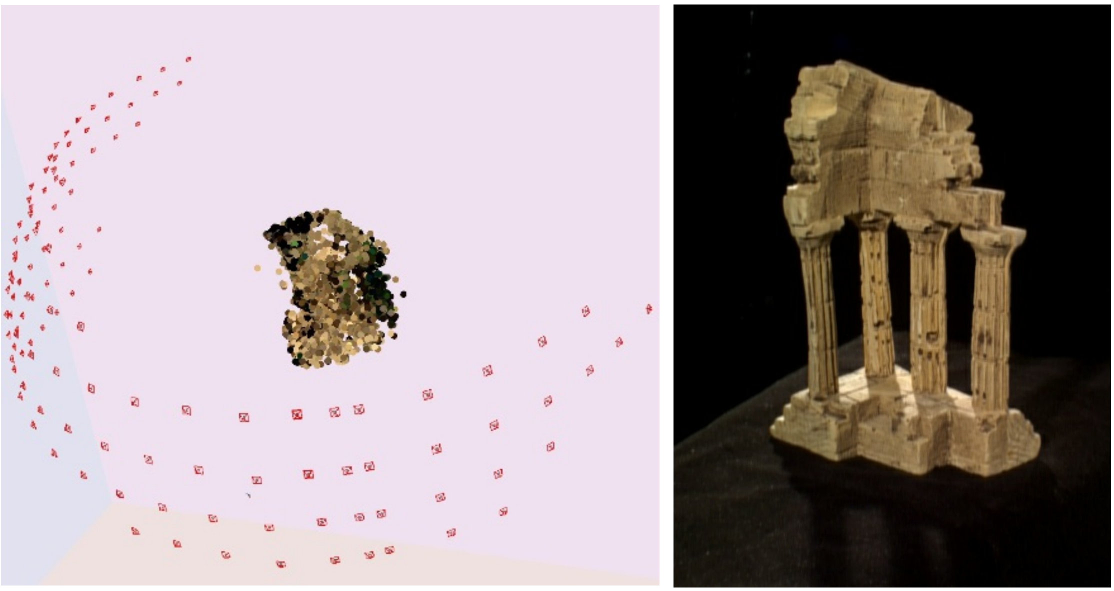


Figure 9.5: One photo and a reconstruction of the ’temple’ data set [SCD+06].

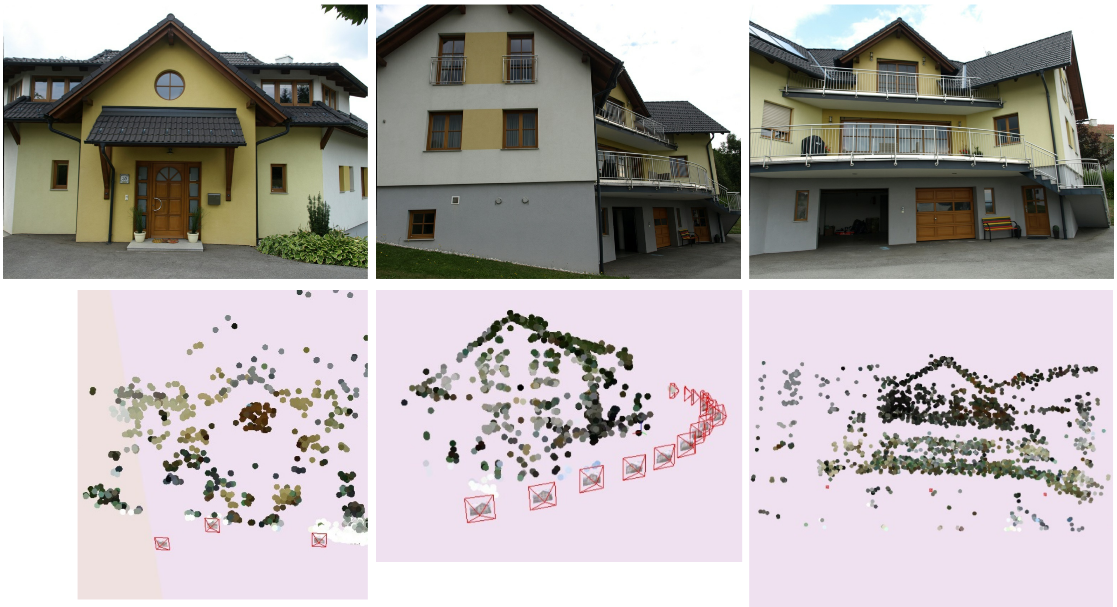


Figure 9.6: Partial reconstructions of a house. Photos courtesy of rmData [rV].

CHAPTER 10

Conclusion and Outlook

In this diploma thesis, we present a composable, reusable library for photogrammetry. The aim is to improve modularity and flexibility of photogrammetry software components, and facilitate rapid and confident experimentation in a scientific context.

Our library emphasises a logically and mathematically sound approach to photogrammetry from first principles, including the components responsible for feature extraction, feature matching, pose recovery, structure from motion and bundle adjustment. We provide reference implementations for all components, but encourage library consumers to modify or replace these components as needed.

We facilitate experimentation through easy modification and extension, considered future work in this diploma thesis.

For example, interesting would be the implementation of an error tracking mechanism, which tracks reconstruction statistics and covariances through the entire pipeline from end to end. Commercial photogrammetry tools often lack the transparency to understand how error statistics are truly calculated, and exact confidence values would allow precise evaluation of a photogrammetry result.

Another example would be the development of novel image features. Line features could be used in combination with point features to reconstruct architecturally complex building facades, while rectangle features could be appropriate to represent white walls in indoor scenes. Related to this, one could imagine incorporating machine learning techniques to exclude disruptive plants or mirror-like surfaces from reconstruction.

The development of user interaction techniques would also be an interesting topic of its own. Currently, few possibilities for a user to influence a photogrammetry pipeline exist. However, reconstruction could potentially profit greatly from user input, for example if the user could approximately pre-sort a photo set to restrict possible motion models, or if the user could manually match a few features in difficult photo pairs.

Ultimately, our results so far show promise, and we hope to open up new avenues of explorative research in the field of photogrammetry.

List of Figures

1.1 Measurements in images during the advent of photography. 2
1.2 Large-scale photogrammetric reconstruction of buildings in Google Maps. Screenshot taken from Google Maps. 3
1.3 General-purpose photogrammetry used to reconstruct a part of Rome from public tourist photos. Image Source: [Sna] 4
1.4 Natural scene in *Battlefield 1*. The assets were created using photogrammetric reconstruction. Image Source: [Sch] 5

2.1 The pinhole camera model. The focal point C lies on the camera axis $\vec{a}$ at the distance of focal length f from the principal point c. Every point X is mapped to its image x_c on image plane I with the line going through C. . 14
2.2 A Camera obscura. Source: [191] 15
2.3 Symbolic structure of a camera objective. The lens system collects as much light from the world as possible, and makes the rays pass through the aperture to hit the sensor in parallel. 16
2.4 Common lens distortion effects in consumer cameras. *Left*: no distortion, *centre*: barrel distortion, *right*: pincushion distortion. 19
2.5 Photo undistortion for a fisheye lens. *Left*: original photo. *Right*: undistorted image, in which straight lines appear straight. Source: The PennCOSYVIO Data Set [Pfr16]. 19

3.1 An example for a blur filter. Each pixel is averaged with its neighbours, using weightings given in the filter kernel. 24
3.2 Corner detection. *Left*: Original image, *centre*: filtered image using corner detector, *right*: Corners (white circles) are the filter's strongest responses. 25
3.3 Different feature extraction algorithms applied to the same image. Features are visualised as red circles. The parameters were chosen manually to produce about 500 keypoints. 30

4.1 Ambiguity in feature matching. The local image patch on the right could be one of many in the left photo. Image Source: [Chu13] 34
4.2 The baseline is the difference between camera centres C_l and C_r. 35
4.3 A small baseline is necessary around house corners because a mere small change in viewpoint makes for a large change in perspective. 36

4.4 Brute force matcher. Matches are few in number, and mismatches often come from vegetation in the image. Matches are visualised as coloured lines. . . 39
4.5 Motion of image regions in a road scene video, visualised as coloured lines. This concept is used heavily in video compression. Source: [Ste04] 40
4.6 Motion consistency matcher. Matches are visualised as coloured lines. . . 43
4.7 Motion prediction matcher. Matches are visualised as coloured lines. . . . 46
4.8 `veryStable` matcher producing about 300 matches. 48
4.9 `middleGround` matcher producing about 500 matches. 48
4.10 `manyMatches` matcher producing about 700 matches. 48

5.1 Perspective used as artistic tool in a photo. Human perception infers depth from perspective distortions. 50
5.2 Geometric transforms visualised. Parallel lines intersect at infinity in Euclidean and Affine space, but the intersection of parallel lines can be localised in Projective space. 52
5.3 Epipolar geometry. The camera centre of the left camera C_l appears in the right image as e_r, the intersection between right image plane and the line between the two centres, which is called Epipole. The triangle of the two camera centres and the world point X forms the Epipolar Plane. 53
5.4 The Fundamental Matrix F_{lr} transfers corresponding points from left image I_l to right image I_r. 54
5.5 Symbolic depiction of the Image of the Absolute Conic (IAC). The IAC looks the same from every viewpoint and therefore puts different cameras in relation. 56
5.6 OpenCV calibration pattern. The pattern's structure is easily recognised in the image (coloured lines). Enough photos of the patterns from different viewpoints enable numeric recovery of the internal camera parameters. Image Source: [Opef] 58

6.1 Four constellations producing the same image. *top left*: base configuration, *bottom left and bottom right*: the two cameras have swapped positions, *right top and right bottom*: one of the cameras has rotated 180 degrees around the baseline. 62
6.2 The same image is produced using different distances $|t|$ between the two cameras. The 3D scene is simply scaled accordingly. 65

7.1 An office scene. These photos are suitable for reconstruction because they are rich in features. 69
7.2 A track $[x_1; x_2; x_3; x_4]$ is used to triangulate the point X. 71
7.3 A symbolic representation of a match graph. The (example) edge weights are the number of matches. Highlighted with thick lines is the maximum spanning tree. 74
7.4 In the first iteration, the identity camera C_{Id} and a baseline length of $|t| = 1$ are used as initial configuration for a photo network. The pose for C_1 is recovered using the parts of the scene that appear in both cameras. . . . 75

7.5 Over the course of multiple iterations, one camera is successively added to the photo network. The part of the scene visible by both the camera and the network is used for pose recovery. 76

8.1 Bundle adjustment. *Left*: Photo network containing numeric errors ϵ for a triangulated point $\hat{x}$. *Right*: Bundle adjusted photo network, in which the triangulated point x has numeric errors close to zero. 83
8.2 Gradient descent algorithm. In every step, the direction of the smallest gradient is followed until the minimum min is reached. This numeric technique is employed when analytical derivation is costly or impossible. 86
8.3 Bundle adjustment. *Left*: Before bundle adjustment. *Centre*: After bundle adjustment. *Right*: Photo of the scene. 90

9.1 Good photos for photogrammetry. The scene is rich in features, and the camera makes small steps to capture crucial detail. 94
9.2 Bad photos for photogrammetry. *Left*: The vegetable has too few features and the photo is dark. *Centre*: A shiny, reflective car is blocking the subject, which makes feature matching problematic. *Right*: The plants are swaying in the wind, destroying models of rigid motion. 94
9.3 A possible photogrammetry user application. This application was coded in a way similar to Code Listing 9.2 using the library for interactive apps Aardvark.Media [Aarb]. 96
9.4 Reconstruction of an office scene. 98
9.5 One photo and a reconstruction of the 'temple' data set [SCD$^+$06]. 100
9.6 Partial reconstructions of a house. Photos courtesy of rmData [rV]. 100

Bibliography

[191] Fizyka z 1910. Camera obscura. `https://commons.wikimedia.org/wiki/File:Camera_obscura_1.jpg`. (Accessed on 01/23/2018).

[Aara] Aardvark. Aardvark. `http://aardvark.graphics`. (Accessed on 01/23/2018).

[Aarb] Aardvark. Aardvark.media. `https://github.com/aardvark-platform/aardvark.media`. (Accessed on 01/23/2018).

[AFS$^+$11] Sameer Agarwal, Yasutaka Furukawa, Noah Snavely, Ian Simon, Brian Curless, Steven M Seitz, and Richard Szeliski. Building rome in a day. *Communications of the ACM*, 54(10):105–112, 2011.

[Agi] Agisoft. Agisoft photoscan. `http://www.agisoft.com`. (Accessed on 01/23/2018).

[Alb02] Jörg Albertz. Albrecht meydenbauer-pioneer of photogrammetric documentation of the cultural heritage. *International Archives of Photogrammetry Remote Sensing and Spatial Information Sciences*, 34(5/C7):19–25, 2002.

[AM$^+$12] Sameer Agarwal, Keir Mierle, et al. Ceres solver, 2012.

[AS11] Pablo F Alcantarilla and T Solutions. Fast explicit diffusion for accelerated features in nonlinear scale spaces. *IEEE Trans. Patt. Anal. Mach. Intell*, 34(7):1281–1298, 2011.

[Aut] Autodesk. Autodesk recap. `https://www.autodesk.com/products/recap/overview`. (Accessed on 01/23/2018).

[Bau00] Adam Baumberg. Reliable feature matching across widely separated views. In *Computer Vision and Pattern Recognition, 2000. Proceedings. IEEE Conference on*, volume 1, pages 774–781. IEEE, 2000.

[BETVG08] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-up robust features (surf). *Computer vision and image understanding*, 110(3):346–359, 2008.

[Cho09] Albert K Chong. New developments in medical photogrammetry. *Geoinf Sci J*, 9(1):41–50, 2009.

[Chu13] Dmitriy Chugai. Texturelib - aged brick building. `http://texturelib.com/texture/?path=/Textures/buildings/buildings/buildings_buildings_0070`, 2013. (Accessed on 01/23/2018).

[CLSF10] Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. Brief: Binary robust independent elementary features. *Computer Vision–ECCV 2010*, pages 778–792, 2010.

[DSC$^+$03] Pierre Drap, Matteo Sgrenzaroli, Marco Canciani, Giacomo Cannata, and Julien Seinturier. Laser scanning and close range photogrammetry: Towards a single measuring tool dedicated to architecture and archaeology. In *CIPA XIXth International Symposium*, pages 1–6, 2003.

[DXW$^+$08] Kaichang Di, Fengliang Xu, Jue Wang, Sanchit Agarwal, Evgenia Brodyagina, Rongxing Li, and Larry Matthies. Photogrammetric processing of rover imagery of the 2003 mars exploration rover mission. *ISPRS Journal of Photogrammetry and Remote Sensing*, 63(2):181–201, 2008.

[Fou96] DA Fournier. Autodiff. a c++ array language extension with automatic differentiation for use in nonlinear modeling and statistics. *Otter Res. Ltd.: Nanaimo, BC, Canada*, 1996.

[FP10] Yasutaka Furukawa and Jean Ponce. Accurate, dense, and robust multiview stereopsis. *IEEE transactions on pattern analysis and machine intelligence*, 32(8):1362–1376, 2010.

[Fur10] Yasutaka Furukawa. Clustering views for multi-view stereo (cmvs). *Website-http://grail. cs. washington. edu/software/cmvs*, 5, 2010.

[GWB10] Sven Grewenig, Joachim Weickert, and Andrés Bruhn. From box filtering to fast explicit diffusion. In *DAGM-Symposium*, pages 533–542. Springer, 2010.

[Har97] Richard I Hartley. In defense of the eight-point algorithm. *IEEE Transactions on pattern analysis and machine intelligence*, 19(6):580–593, 1997.

[Hen34] William Henry. William henry fox talbot and the foundations of spectrochemical analysis. *Raman Spectroscopy of Oil Shale*, 3(7):28, 1834.

[HES15] M Hasheminasab, H Ebadi, and A Sedaghat. An integrated ransac and graph based mismatch elimination approach for wide-baseline image matching. *The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, 40(1):297, 2015.

[Hol] Oliver Wendell Holmes. Stereoscope. `https://commons.wikimedia.org/wiki/File:Holmes_stereoscope.jpg`. (Accessed on 01/23/2018).

[HSM$^+$14] Georg Haaser, Harald Steinlechner, Michael May, Michael Schwärzler, Stefan Maierhofer, and Robert Tobler. Semantic composition of language-integrated shaders. In *International Joint Conference on Computer Vision, Imaging and Computer Graphics*, pages 45–61. Springer, 2014.

[HSMT15] Georg Haaser, Harald Steinlechner, Stefan Maierhofer, and Robert F Tobler. An incremental rendering vm. In *Proceedings of the 7th Conference on High-Performance Graphics*, pages 51–60. ACM, 2015.

[HZ03] Richard Hartley and Andrew Zisserman. *Multiple view geometry in computer vision*. Cambridge university press, 2003.

[IKK$^+$10] Arnold Irschara, Viktor Kaufmann, Manfred Klopschitz, Horst Bischof, and Franz Leberl. Towards fully automatic photogrammetric reconstruction using digital images taken from uavs. 2010.

[LCL$^+$14] Wen-Yan Daniel Lin, Ming-Ming Cheng, Jiangbo Lu, Hongsheng Yang, Minh N Do, and Philip Torr. Bilateral functions for global motion modeling. In *European Conference on Computer Vision*, pages 341–356. Springer, 2014.

[LCS11] Stefan Leutenegger, Margarita Chli, and Roland Y Siegwart. Brisk: Binary robust invariant scalable keypoints. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2548–2555. IEEE, 2011.

[LLJ$^+$16] Wen-Yan Lin, Siying Liu, Nianjuan Jiang, Minh N Do, Ping Tan, and Jiangbo Lu. Repmatch: Robust feature matching and pose for reconstructing modern cities. In *European Conference on Computer Vision*, pages 562–579. Springer, 2016.

[Low99] David G Lowe. Object recognition from local scale-invariant features. In *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, volume 2, pages 1150–1157. Ieee, 1999.

[MBSL99] Jitendra Malik, Serge Belongie, Jianbo Shi, and Thomas Leung. Textons, contours and regions: Cue integration in image segmentation. In *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, volume 2, pages 918–925. IEEE, 1999.

[Mey] Alrbrecht Meydenbauer. Wetzlar dom. `https://commons.wikimedia.org/wiki/File:Wetzlar_Dom_81-008.jpg`. (Accessed on 01/23/2018).

[MH06] Hans-Gerd Maas and Uwe Hampel. Photogrammetric techniques in civil engineering material testing and structure monitoring. *Photogrammetric Engineering & Remote Sensing*, 72(1):39–45, 2006.

[MR96] Werner Mayr and W Reinhardt. Digital photogrammetry joins gis-a powerful combination. *International Archives of Photogrammetry and Remote Sensing*, 31:553–556, 1996.

[MWA$^+$13] Przemyslaw Musialski, Peter Wonka, Daniel G Aliaga, Michael Wimmer, L v Gool, and Werner Purgathofer. A survey of urban reconstruction. In *Computer graphics forum*, volume 32, pages 146–177. Wiley Online Library, 2013.

[Nis04] David Nistér. An efficient solution to the five-point relative pose problem. *IEEE transactions on pattern analysis and machine intelligence*, 26(6):756–770, 2004.

[Opea] OpenCV. Akaze. `https://docs.opencv.org/3.0-beta/doc/tutorials/features2d/akaze_matching/akaze_matching.html`. (Accessed on 01/23/2018).

[Opeb] OpenCV. Brisk. `https://docs.opencv.org/2.4/modules/features2d/doc/feature_detection_and_description.html#id7`. (Accessed on 01/23/2018).

[Opec] OpenCV. Camera calibration and 3d reconstruction. `https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html`. (Accessed on 01/23/2018).

[Oped] OpenCV. Introduction to sift. `https://docs.opencv.org/3.3.0/da/df5/tutorial_py_sift_intro.html`. (Accessed on 01/23/2018).

[Opee] OpenCV. Introduction to surf. `https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_surf_intro/py_surf_intro.html`. (Accessed on 01/23/2018).

[Opef] OpenCV. Opencv camera calibration. `https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_calib3d/py_calibration/py_calibration.html`. (Accessed on 01/23/2018).

[Opeg] OpenCV. Opencv findessentialmat based on the five point algorithm. `https://docs.opencv.org/3.0-beta/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html#findessentialmat`. (Accessed on 01/23/2018).

[Opeh] OpenCV. Orb. `https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_orb/py_orb.html`. (Accessed on 01/23/2018).

[Pfr16] Bernd Pfrommer. Penncosyvio data set. `https://daniilidis-group.github.io/penncosyvio/intrinsic_calib/`, 2016. (Accessed on 01/23/2018).

[PHH15] Shih-Ting Peng, Shih-Yu Hsu, and KC Hsieh. An interactive immersive serious game application for kunyu quantu world map. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 2(5):221, 2015.

[RBN$^+$11] Fabio Remondino, L Barazzetti, Francesco Nex, Marco Scaioni, and Daniele Sarazzi. Uav photogrammetry for mapping and 3d modeling–current status and future perspectives. *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 38(1):C22, 2011.

[RD06] Edward Rosten and Tom Drummond. Machine learning for high-speed corner detection. *Computer Vision–ECCV 2006*, pages 430–443, 2006.

[Rea] RealityCapture. Home - capturingreality.com. `https://www.capturingreality.com/`. (Accessed on 01/23/2018).

[Rec] Brian Recktenwald. Environment art lookdev using unreal photogrammetry. `https://www.thegnomonworkshop.com/tutorials/environment-art-lookdev-using-unreal-photogrammetry`. (Accessed on 01/23/2018).

[RR99] John A Richards and JA Richards. *Remote sensing digital image analysis*, volume 3. Springer, 1999.

[RRKB11] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: An efficient alternative to sift or surf. In *Computer Vision (ICCV), 2011 IEEE international conference on*, pages 2564–2571. IEEE, 2011.

[RSN$^+$14] Fabio Remondino, Maria Grazia Spera, Erica Nocerino, Fabio Menna, and Francesco Nex. State of the art in high density image matching. *The Photogrammetric Record*, 29(146):144–166, 2014.

[rV] rmData Vermessung. rmdata vermessung oesterreich - software fuer vermessung, planerstellung und geoinformation. `http://www.rmdata.at/`. (Accessed on 01/23/2018).

[S$^+$16] Vincent Schülé et al. Path of an object to a game engine via 3d reconstruction. 2016.

[SCD+06] Steven M Seitz, Brian Curless, James Diebel, Daniel Scharstein, and Richard Szeliski. A comparison and evaluation of multi-view stereo reconstruction algorithms. In *Computer vision and pattern recognition, 2006 IEEE Computer Society Conference on*, volume 1, pages 519–528. IEEE, 2006.

[Sch] Anders Schei. Battlefield 1 assets. `https://www.artstation.com/artwork/8OAoO`. (Accessed on 01/23/2018).

[Sil93] Robert J Silverman. The stereoscope and photographic depiction in the 19th century. *Technology and Culture*, 34(4):729–756, 1993.

[SMM16] Yohann Salaün, Renaud Marlet, and Pascal Monasse. Robust and accurate line-and/or point-based pose estimation without manhattan assumptions. In *European Conference on Computer Vision*, pages 801–818. Springer, 2016.

[Sna] Noah Snavely. Building rome in a day. `http://www2.technologyreview.com/tr35/profile.aspx?TRID=1095`. (Accessed on 01/23/2018).

[Sna06] N Snavely. Bundler: Sfm for unordered image collections. *Domain: http://phototour. cs. washington.-edu/bundler/.(15.09. 10)*, 2006.

[Ste04] Fridtjof Stein. Efficient computation of optical flow using the census transform, 08 2004.

[Sze10] Richard Szeliski. *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.

[TM00] Regina Tokarczyk and Sawomir Mikrut. Close range photogrammetry system for medicine and railways. *International Archieves of Photogrammetry and Remote Sensing*, 5, 2000.

[Tri97] Bill Triggs. Autocalibration and the absolute quadric. In *Computer Vision and Pattern Recognition, 1997. Proceedings., 1997 IEEE Computer Society Conference on*, pages 609–614. IEEE, 1997.

[Tsa13] Eirik Tsarpalis. Fspickler. `https://github.com/mbraceproject/FsPickler`, 2013. (Accessed on 01/23/2018).

[VSS14] Rebekka Volk, Julian Stengel, and Frank Schultmann. Building information modeling (bim) for existing buildings?literature review and future needs. *Automation in construction*, 38:109–127, 2014.

[W+11] Changchang Wu et al. Visualsfm: A visual structure from motion system. 2011.

[WAH12] Adam C Watts, Vincent G Ambrosia, and Everett A Hinkley. Unmanned aircraft systems in remote sensing and scientific research: Classification and considerations of use. *Remote Sensing*, 4(6):1671–1692, 2012.

[WCH$^+$92] Juyang Weng, Paul Cohen, Marc Herniou, et al. Camera calibration with distortion models and accuracy evaluation. *IEEE Transactions on pattern analysis and machine intelligence*, 14(10):965–980, 1992.