3D Reconstruction of Buildings From Images with Automatic Façade Refinement

by
Christian Lindequist Larsen

Master's Thesis in
Vision, Graphics and Interactive Systems

September 2009 – June 2010

AALBORG UNIVERSITY

Aalborg University
Department of Electronic Systems
Fredrik Bajers Vej 7
9220 Aalborg Ø
http://es.aau.dk/

Title:
3D Reconstruction of Buildings From Images with Automatic Façade Refinement

Author:
Christian Lindequist Larsen

Project Period:
September 1, 2009 – June 3, 2010

Project Group:
10gr1023

Supervisor:
Thomas B. Moeslund

Copies:
3

Report Pages:
115

Total Pages:
125

Attachments:
1 CD

Abstract:

This project deals with the problem of reconstructing 3D models of real world buildings from images. Potentially real estate marketing can be improved by providing interactive visualizations, e.g. on websites, of properties for sale. For this to be a viable option, however, simpler methods for 3D reconstruction are needed. In particular existing user assisted methods can be improved by automating the process of adding façade details to the reconstructed models.

In this project a proof of concept system covering the whole reconstruction process has been developed. Structure and motion is recovered from an unordered set of images of the building to reconstruct, and this is followed by user assisted reconstruction of a coarse textured 3D model. The primary contribution in the project is the development of a novel method for automatic façade reconstruction, which when applied to the coarse model automatically adds façade details such as recessed windows and doors. The proposed method is based on analyzing the appearance of the façade, and this is achieved using methods for image processing and pattern classification.

The results obtained from using the developed system for reconstructing several refined 3D models of buildings from images show that the proposed reconstruction method is suitable for this purpose. The developed method for automatic façade reconstruction on average correctly detected and reconstructed 89% of recessed windows for the tested buildings. Some work remain for automatic façade reconstruction to be fully automatic, e.g. the depth of recessed regions is specified by the user, but in general it is concluded that the proposed method leads to an improvement compared to existing user assisted reconstruction methods.

Preface

This report constitutes the master's thesis for Christian Lindequist Larsen, group 10gr1023 at the Vision, Graphics and Interactive Systems specialization at Department of Electronic Systems, Aalborg University. The project has spanned the 9^{th} and 10^{th} semesters and extended from September 1, 2009 to June 3, 2010.

References to secondary literature are specified using the syntax [number], where the number refers to the bibliography found at the end of the report, before the appendices. On the following page the vector and matrix notation used in the report is introduced.

During the project a proof of concept system for reconstructing 3D models of buildings from images has been designed and implemented. The implementation primarily consists of two applications written in C++, and various libraries have been used for specific tasks in the implementation. In particular OpenCV is used for image processing, and OpenGL is used for the user interface. Additionally, part of image processing is implemented using Matlab.

Further resources are available on the CD attached to this report. The contents of this CD are:

- PDF version of this report.
- All test data and the obtained results.
- Implementation source code and documentation.

Aalborg – June 3, 2010

Christian Lindequist Larsen

Notation

Vectors are typeset in bold roman typeface, and are column vectors unless explicitly transposed. A vector may appear in text as $\mathbf{x} = [x_1 \ x_2 \ x_3]^\mathsf{T}$, which is equivalent to

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}.$$

Matrices are typeset using capital letters in roman typeface. For instance, a 3×4 matrix P is defined as

$$\mathrm{P} = [\mathbf{p}_1 \ \mathbf{p}_2 \ \mathbf{p}_3 \ \mathbf{p}_4] = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix},$$

where the vectors $\mathbf{p}_i, \ i = 1, \ldots, 4$ are the column vectors of the matrix. Zero entries of matrices may be omitted to emphasize only the important elements, i.e. the following matrices are equivalent.

$$\begin{bmatrix} a & b & c \\ & d & e \\ & & f \end{bmatrix} = \begin{bmatrix} a & b & c \\ 0 & d & e \\ 0 & 0 & f \end{bmatrix}$$

Contents

Part I

Introduction

Chapter 1

Motivation

This is the era of computers. Today computers play an indispensable role in sustaining the standard of living. Almost any man-made product that surrounds us has at some point been treated in digital form, whether it is during development, production, marketing, or consumption. In development and production, computers make assisting technologies such as CAD/CAM[1] available, and many products would be impossible to design and manufacture without the help of computers. In marketing computers are used for both creating and publishing product advertisements. For some groups of consumer products, e.g. electronic devices, it is even becoming the norm to order directly from shops on the internet, without having seen the product in real life. The information available online about products is, in many cases, sufficient for the buyer to make a considered decision as to which product to choose.

Common for the above examples is that computers help design, process, and present objects in the real world. To do this the objects must somehow be represented in digital form. For many types of products the appearance is a key selling point, and therefore they are presented visually in marketing material, both in printed and electronic advertisements. Typically this is done using product photographs, but for online product presentation more interactive options are available. An example of this is visualizing products interactively in 3D on a website, and in figure 1.1 two examples of this are shown using the Holomatix Blaze 3D software [3]. Using photorealistic rendering, the potential customer gets a precise impression of the appearance of the product. Furthermore, the interactivity allows the customer to inspect structure and details of the product that might otherwise be hard to see in a product photograph.

Rendering a product photorealistically requires that a precise 3D model of the object including surface textures exists [36]. For products designed using CAD such information is often readily available, as exact 3D models are a by-product of the process. The cell phone in figure 1.1a is an example of this. But for many products this is not the case, and the house in figure 1.1b is such an example. In this situation it is necessary to somehow obtain a 3D model of the product, i.e. a digital representation, in order to present it interactively.

There are many ways to address the problem of reconstructing a 3D model from a real world object, but it is often a cumbersome and time-consuming

[1] Computer-Aided Design (CAD), and Computer-Aided Manufacturing (CAM).

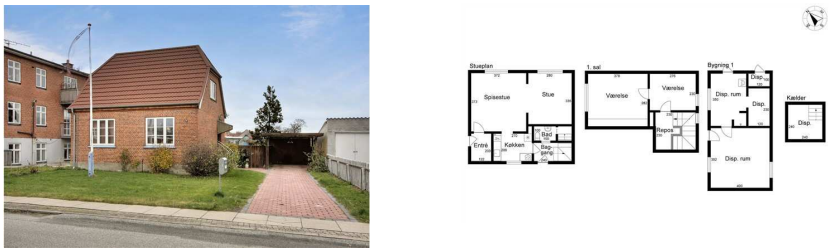
(a) **(b)**

Figure 1.1: Examples of interactive online 3D product presentations using the Holomatix Blaze 3D software. The user can orbit around the product, and zoom in to take a closer look. a) Nokia 6300 cell phone [10]. b) Architectural visualization [3].

task. For instance the object can be reconstructed manually by taking measurements and using CAD tools for creating the 3D model, or laser scanning could be employed to accurately measure the structure of the object [36]. Both options are infeasible in many situations. For instance, manual modeling can be very time-consuming depending on the desired level of detail, and laser scanning equipment is expensive and not available to layman. Therefore to make interactive product presentations available for a wider range of products, a simpler approach to 3D model reconstruction is necessary.

As mentioned above, one type of products for which 3D models typically do not exist is buildings. Real estate marketing material often includes a description with information about the property, one or more photographs of the exterior and interior of the building, and a floor plan, see figure 1.2. Although this provides a lot of useful information for the potential buyer, it is still not enough to really get a *feeling* of the property. If real estate agents could provide an interactive virtual tour of properties for sale, e.g. as in the example in figure 1.1b, on their website, the potential customers would get an improved experience and have a better opportunity to get an impression of the buildings before buying. Of course for real estate, this can never replace actually visiting the property, but it may help making a decision which properties to take a look at in person.

To make online interactive 3D presentations a viable option for real estate agents, 3D model reconstruction of buildings must be simplified compared to the methods briefly mentioned above. This leads to the initiating problem of the project.



(a) **(b)**

Figure 1.2: A typical example of the visual presentation of a property for sale in real estate marketing material [2]. a) Photograph of the building. b) Floor plan.

1.1 Initiating Problem

The introduction has established that there is a need for simpler methods for reconstruction of 3D models from real world objects, which can be used for interactively presenting products e.g. on websites. In particular marketing of real estate, where 3D models of the buildings normally do no exist, could potentially benefit from this. Therefore the initiating problem of the project is as follows:

> *How is it possible to reconstruct a 3D model of a real world building, for interactive visualization, in a simple manner?*

To answer this question, the initiating problem is analyzed in the following chapter. This analysis then forms the basis for defining the specific problem of the project in chapter 3.

Chapter 2

Problem Analysis

The purpose of this chapter is to analyze the initiating problem defined in section 1.1. Based on this analysis the specific problem of the project is defined in the following chapter.

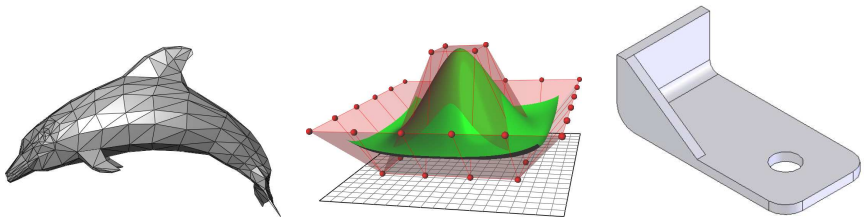
The key question in the initiating problem is how to reconstruct a 3D model from a real world object, in this case a building. It is a prerequisite for answering this question to know what is meant by 3D model, and this is defined in section 2.1 where different 3D model representations are analyzed. Then section 2.2 provides an analysis of different methods for 3D model reconstruction. Furthermore, the initiating problem asks for a 3D model for interactive visualization, which can be reconstructed in a simple manner. So in section 2.3, the method which best fits with the initiating problem is selected based on the analysis of 3D model representations and reconstruction methods.

2.1 Model Representation

As discussed in chapter 1, to render an object photorealistically, a precise 3D model including surface textures is required. That is the 3D model includes both information about the 3D shape of the object and the appearance of the object. Textures which are mapped to the surface of the object are typically represented using images which can be extracted from photographs of the object [36]. The main focus of the following analysis is how to represent the shape of the object.

The shape of a 3D model can be represented in different ways, and these representations can be categorized as: polygon mesh models, surface models, and solid CAD models. The choice of representation depends on the data acquisition method and the final purpose of the reconstructed model.

Polygon Mesh Models The shape of the object is represented using a collection of planar polygons referred to as faces. Faces are typically triangles, quadrilaterals, or simple convex polygons, but may be more complex such as concave polygons or polygons with holes. The corners of the polygons are referred to as vertices, and thus a polygon mesh is a collection of vertices, edges, and faces [51]. Curved surfaces are approximated using many small faceted surfaces. Polygon mesh models having only triangle faces are referred to as triangle



(a) (b) (c)

Figure 2.1: Examples of different representations of the shape of a 3D model. a) Triangle mesh [51]. b) NURBS surface with control points [49]. c) Solid CAD model [29].

meshes, and these are useful for visualization, because modern graphics rendering hardware is optimized for rendering this type of models. In figure 2.1a an example of a triangle mesh is shown.

Surface Models Objects with smooth shapes require dense tessellation when represented as polygon meshes. Such shapes are better represented using surface models which consist of a quilt of curved surface patches. The patches may be Non-Uniform Rational B-Spline (NURBS) surfaces which give a mathematically precise representation of surfaces, e.g. a sphere can be represented exactly. The shape of the surface is determined by a number of weighted control points. NURBS was originally developed by Pierre Bézier and Paul de Casteljau for accurately representing shapes such as car bodies [49]. For interactive rendering NURBS surfaces must be transformed into a polygonal representation, and for this various algorithms exist, e.g. [12]. In figure 2.1b an example of a NURBS surface is shown.

Solid CAD Models This is the most abstract representation of the shape of the object. In polygon mesh and surface models only the surface of the object is considered, whereas in CAD models the object is represented as a 3D solid. These solids are typically created in a way similar to manipulation of real world objects, and may be composed of basic geometric forms such as prisms, cylinders, and spheres [42]. Geometric forms may be combined using Boolean operations, i.e. union, intersection, and difference, and this technique is referred to as Constructive Solid Geometry (CSG) [43], see figure 2.2. CAD models are not limited to representing the shape of an object, they may also contain information about the design intent. Being described parametrically, the shape of an object may be modified simply by changing the parameters, e.g. a sphere is defined by its center point and radius [41]. In figure 2.1c an example of a solid CAD model is shown.

2.2 Reconstruction Methods

Knowing how 3D models can be represented, in the following it is analyzed how to get from a real world object to such representation. Reconstruction of 3D models from real world objects is a field which has been subject to much research, and multiple ways to achieve this goal have been developed. Many of

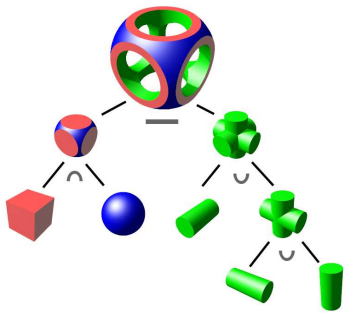


Figure 2.2: Example of a CSG object represented using a binary tree, where leafs represent primitives, and nodes represent operations. The operations are: $\cup$ union, $\cap$ intersection, and $-$ difference [43].

these techniques are not tied to any specific type of object, but may be better suited than others for particular kinds of objects. Common for all methods is that they consist of acquisition of data from the real world followed by reconstruction or modeling of a 3D model from this data. Depending on the method used for acquisition, the data may need additional processing before reconstruction is possible. That is, the process can be seen as consisting of three steps: data acquisition, intermediate processing, and model reconstruction.

Overall reconstruction methods can be divided into three categories: manual, 3D scanning, and photogrammetry. These categories indicate the technology used for data acquisition. Intermediate processing and 3D model reconstruction differ for each of these categories, and the categories are analyzed in the following. An overview of the analyzed reconstruction methods is given in table 2.1 on page 17.

2.2.1 Manual

Manual reconstruction is the most basic method for reconstructing a 3D model of a real world object. No special equipment is required for data acquisition; the object of interest is simply measured up manually using e.g. a folding ruler and a protractor. Several measurements may be necessary in order to achieve the desired level of detail for the final 3D model. If the model is not reconstructed on-site, the measurements must be noted, and drawings and sketches can be used to illustrate the structure of the object.

Reconstruction of the 3D model is also a manual process, which can result in any of the representations in section 2.1 depending on final purpose. Different software packages supporting manual modeling exist. For polygon mesh and NURBS modeling e.g. 3ds Max [1] may be used, while CAD modeling may be done using e.g. AutoCAD [1] or SolidWorks [9]. Texturing the object is typically done using photographs of the object, which are mapped onto the surface of the reconstructed model [36].

In manual reconstruction the user is involved in the whole process, and therefore it is a cumbersome and very labour intensive method. The achievable level of realism and detail is limited [36].

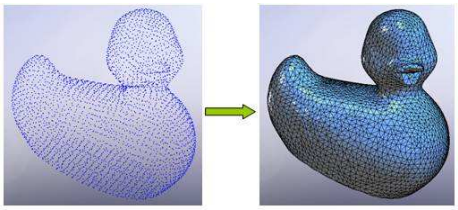


Figure 2.3: A point cloud (left), and an automatically reconstructed triangle mesh (right) [5].

2.2.2 3D Scanning

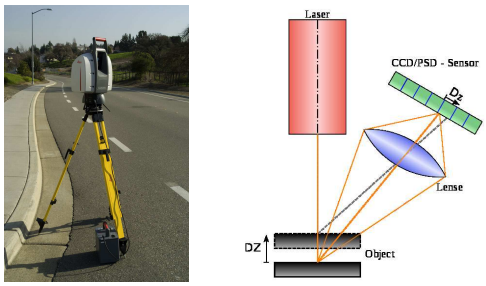
The alternative to manual reconstruction is letting computers take over some of the work, and a well established method is 3D scanning. A 3D scanner is a device that captures detailed information about the shape and possibly appearance, i.e. color, of a real world object [41]. The result of 3D scanning typically is a point cloud, i.e. a large number of points in space sampled from the surface of the object, and from this data a 3D model can be reconstructed. An example of a point cloud and an automatically reconstructed triangle mesh is shown in figure 2.3.

Data Acquisition There are many techniques for performing 3D scanning, but common is that they sample the distance to the real world object and generate a point cloud of these samples. 3D scanning devices are analogous to cameras in that they also have a conic field of view, and only collect information about non-occluded surfaces [41]. But instead of or in addition to color information, each sample is the distance from the scanner to the object, and hence the result is sometimes referred to as a range image, where pixel values represent the distance to the object [53]. The most common type of 3D scanning devices are non-contact active scanners, which are not in physical contact with the scanned object, but do emit radiation or light and detect the reflection to scan the object [41].

The most well-known technique for 3D scanning is using a laser scanner. Actually two techniques exist for laser scanning: time-of-flight and triangulation. A time-of-flight 3D laser scanner measures the distance to the object of interest by knowing the speed of light and measuring the round-trip time of a pulse of light. Only the distance of a single point is scanned at a time, and to scan the whole field of view, the direction of the laser beam is changed for each sample typically using a set of mirrors [41]. In figure 2.4a a time-of-flight laser scanner is shown.

A triangulation 3D laser scanner like a time-of-flight laser scanner directs a laser towards the object of interest. But instead of measuring the round-trip time of a pulse, a camera is used for detecting the dot produced by the laser at the object surface. The position of the point can be triangulated using the known information, which is the distance between the laser emitter and the camera, the direction of the laser, and the position of the dot in the field of view of the camera [41]. This principle is illustrated in figure 2.4b.

An alternative to laser scanning is using what is referred to as structured light, where projected light patterns are emitted using e.g. an LCD projector.



(a) (b)

Figure 2.4: a) A wide range time-of-flight laser scanner. The head is rotated horizontally, and a mirror flips vertically to direct the laser beam to the whole scene [41]. b) Laser triangulation principle, with two object distances shown [41].

The light patterns are reflected from the surface of the scanned object, and these reflections are captured by one or more cameras. From the distortion of the light patterns detected by the cameras, the shape of the object can be determined in a way similar to triangulation described above. The used light patterns range from a single stripe, which is swept across the scene, to more advanced patterns that allow sampling of more points simultaneously [54].

The choice of 3D scanning technique depends on the task at hand. Time-of-flight laser scanners can operate over long distances on the order of kilometers, but their accuracy is limited due to the speed of light and timing precision. On the contrary, triangulation laser scanners are limited in range, typically in the order of meters, but have very high precision [41]. These types of laser scanners are suitable for capturing static scenes, because only one point at a time is sampled. The number of points sampled per second is typically in the order of 10,000–100,000 so high resolution scans can take minutes [41]. Structured light scanning is also limited in range, but the scanning of multiple points simultaneously makes capturing moving objects possible. A structured light system for real-time scanning of deformable objects, which is capable of running at 40 frames per second, has been developed [55].

Intermediate Processing Regardless of the technique used for 3D scanning, the result of data acquisition is a point cloud. But a single scan only samples the depth of the scene from one perspective, and parts of the scene may be occluded. Therefore for almost any object, multiple scans are required to obtain a complete model, and often hundreds of scans from different directions are necessary [41].

The point clouds obtained from each scan are relative to the coordinate system of the scanner, and these must be transformed into a common coordinate system. This process is referred to as registration. In high-end systems, accurate tracking of position and orientation of scanning equipment may be used, such that individual scans can be aligned using this information. In less expensive systems, e.g. turntables may be used which limit the degrees of freedom, but also the size of objects that can be scanned. Some systems rely on human interaction to identify matching features in different scans, but automatic registration using feature matching is a possibility, and this is an active area of research [16].

Model Reconstruction The final step of reconstruction using 3D scanning techniques is transforming the registered point cloud into a 3D model using one of the representations in section 2.1 for the shape. An example of reconstructing a triangle mesh from a point cloud was shown in figure 2.3, and this can be done automatically using e.g. Delaunay triangulation [16], marching cubes [25], or the ball-pivoting algorithm [15]. Reconstruction of surface models, e.g. using NURBS, is typically done by converting from a reconstructed triangle mesh, and automatic algorithms for this purpose exist [16]. Solid CAD models can also be reconstructed from point cloud data, but this typically is a process which requires some degree of user interaction. The process can be greatly simplified, however, by using commercial software packages such as Rapidform XOR [8].

Laser scanners typically are unable to capture the color of the surface, so texturing the reconstructed 3D model requires photographs of the scene, preferably captured with a calibrated camera at the same time as each of the scans [16]. As for 3D scanning devices, the camera can only capture non-occluded surfaces, so several photographs may be needed. Photographs may also be captured independently of the scan process, and then texturing the 3D model is done the same way as for manual reconstruction [16]. It is possible, however, using structured light scanning to capture both shape and color of the scene simultaneously, as demonstrated in [55].

Compared to manual reconstruction, 3D scanning techniques can save a lot of work. Three scanning techniques were analyzed, and the choice depends on the task at hand, which determines requirements for range, accuracy, and speed. Even though 3D scanning simplifies data acquisition compared to manual measurements, some manual work is still necessary. Typically several scans are required, and for large objects the scanning equipment must be moved and possibly calibrated each time to ease registration. For texturing the 3D model, several photographs of the object are also needed. Automatic methods for triangle mesh and surface model reconstruction exist, while reconstruction of CAD models require some degree of user interaction.

One parameter not discussed in the above analysis is cost of the scanning equipment. Generally laser scanning equipment is expensive, e.g. the Leica ScanStation 2 depicted in figure 2.4a has a list price of US $102,375 [22]. Cheaper solutions are available for smaller objects, such as the NextEngine Desktop 3D Scanner HD, which captures both shape and color of objects and is available for US $2,995 [6].

2.2.3 Photogrammetry

A camera captures the appearance of an object or a scene, but also information about the 3D shape can be extracted from photographs. This is evident from the fact that humans are able to perceive and navigate in a 3D world using their vision. Two eyes give humans the ability to perceive the depth of a scene, because the eyes see two slightly different images due to their relative position. The difference in the images depends on the distance to objects in the scene [36]. But even if one eye is closed, it is still possible to get an impression of the 3D structure of the scene by moving the head. Additionally the change in the image seen when moving allows estimating the direction of movement. This means that both structure and motion is recovered simultaneously [36].

Transferring the ability to infer structure from motion to machines is a field in computer vision that has been studied intensively in the last decades. The main focus has been to find algorithms for extracting the necessary information automatically from multiple images, or a video sequence [36]. Notably the book [23] by Richard Hartley and Andrew Zisserman provides very useful insight into the field of reconstructing 3D models from images. Actually the field of photogrammetry is as old as modern photography, and already in the middle of the 19^{th} century, photographs were used for making maps and measuring buildings [36, 50].

To understand how a 3D model can be reconstructed from photographs it is necessary to understand the information that is captured by a camera. For each image coordinate there is a corresponding ray in space passing through the projection centre of the camera, for which the camera captures the appearance, i.e. color, of the closest object that intersects this ray. As opposed to 3D scanning, the distance from the camera to the intersection is not known. It is possible, however, to measure relative distances using an image if certain conditions are met, and this is useful for e.g. making maps. It is required that the points, between which distances are measured, lie in a plane parallel to the image plane, and that the internal calibration of the camera is known [50]. The internal calibration of a camera is determined by a set of intrinsic calibration parameters, which among others include the focal length, and parameters for lens distortion [17].

Since the distance from the camera to the scene is unknown, it is clear that the 3D shape of an object can not be reconstructed from a single image. Similar to human vision, however, it is possible to estimate the 3D position of a point seen in two images captured from different locations. The two images may be captured by different cameras, or a single camera may be moved. The estimation is done using a process called triangulation, in which the intersection between the two rays that correspond to the point seen in the images is found. To find the intersection between the rays, the relative camera motion between the two images, i.e. the external calibration, must be known in addition to the internal calibration of the cameras [36]. The external calibration of a camera is defined by a set of extrinsic parameters describing the 3D position and orientation of the camera [17]. Thus to extract information about the 3D shape of an object, at least two calibrated images captured from different perspectives are needed along with a set of corresponding points in the images that can be triangulated.

Much previous research has focused on extracting the necessary information automatically from images. Algorithms for finding corresponding image features, i.e. image points originating from the same feature in the original scene, is one example. From corresponding features in two images it is possible to estimate the relative camera motion, thus relaxing the requirement of extrinsic camera calibration. Typically intrinsic camera calibration is required, though some algorithms for automatic intrinsic calibration have been developed, e.g. [35]. Some of these results are briefly discussed in the following. Overall 3D model reconstruction using photogrammetry consists of capturing images, recovering structure and motion from the images, and finally building a 3D model.

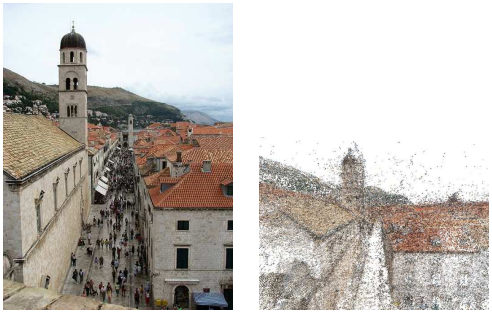
Data Acquisition In photogrammetry data acquisition consists of capturing a set or sequence of images of the real world object to be reconstructed. The images may be photographs or frames of a video sequence. Cameras only capture information about non-occluded surfaces, and therefore as with 3D scanning, multiple images of the object from different perspectives are required. In addition overlap between the images is required for automatic feature matching and motion recovery. If camera calibration is required, this is also part of data acquisition. Camera calibration is usually achieved by capturing a series of images of a calibration pattern from different perspectives, and afterwards using software for estimating the calibration parameters of the camera, see e.g. [17]. If cameras are not extrinsically calibrated, the object can only be reconstructed up to an arbitrary similarity transform[1], so a few manual measurements may be required to recover the correct scale, orientation, and position.

Intermediate Processing The result of data acquisition is a set of overlapping images of the object of interest captured from different perspectives, preferably with no surfaces that are hidden in all images. The intrinsic camera calibration of the images may or may not be known depending on the data acquisition process, but typically position and orientation of cameras, i.e. extrinsic calibration, is unknown. This information is required for reconstruction as discussed above, and the purpose of intermediate processing is to recover both structure and motion from the captured images. Here structure is an estimate of the 3D positions of detected features in the images, and motion is an estimate of the extrinsic calibration of the cameras. Both structure and motion is recovered, because optimal estimation of either depends on estimating the other. Typically structure and motion estimates are optimized using an iterative refinement process which is referred to as bundle adjustment [26].

The first step in recovering structure and motion is to estimate the relative motion between different images, and this requires a set of corresponding points, in each pair of images. A widely used algorithm for finding corresponding features in images automatically is the Scale Invariant Feature Transform (SIFT) [27] due to its robustness. Using the point correspondences, the relative motion between pairs of images is recovered using an algorithm such as the normalized 8-point algorithm [23]. Knowing the relative motion, it is possible to estimate the 3D positions of corresponding points using triangulation. Finally, the complete structure and motion is recovered by clustering the images using a technique such as the one developed in [37], where bundle adjustment is applied during the process. The result of intermediate processing is estimated extrinsic and intrinsic calibration of the cameras, and estimated 3D positions of the set of extracted image features.

A process similar to the one described here has been used in multiple projects to recover structure and motion from large collections of photographs. One example is the project Building Rome in a Day, which seeks to automatically reconstruct entire cities from images harvested on the internet [11]. In figure 2.5 a reconstructed scene from the project is shown. Also Photosynth from Microsoft, which makes interactive 3D exploration of photo collections possible, uses a similar technique [4].

[1] Shape preserving transformation, which is composed of rotation, isotropic scaling, and translation [23].



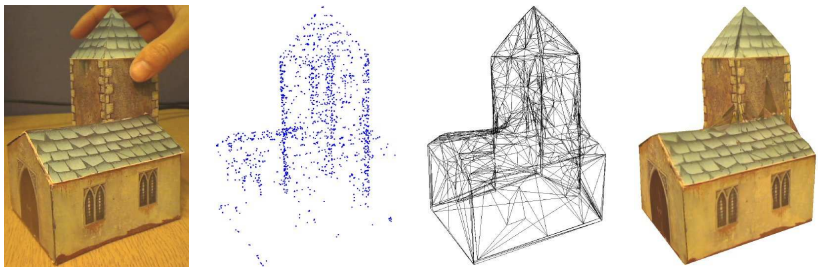
(a) **(b)**

Figure 2.5: A scene from the city Dubrovnik, Croatia is reconstructed from 4,619 images yielding 3,485,717 points [11]. a) Original image. b) Reconstructed point cloud seen from the same viewpoint.

Model Reconstruction As with both manual reconstruction and 3D scanning, the resulting 3D model can be reconstructed using any of the representations discussed in section 2.1. Part of the result of intermediate processing is a point cloud consisting of the extracted image features, and thus the techniques for model reconstruction described in section 2.2.2 are also applicable here. Point clouds obtained using photogrammetry, however, are typically very sparse compared to those obtained using 3D scanning techniques, so using this result directly leads to models with poor visual quality [36]. Typically other methods are used for model reconstruction, and they require different levels of user interaction, ranging from manual to automatic.

Common for the manual reconstruction methods is that they start from the structure and motion recovered from intermediate processing. The user then typically selects features in the scene that are used for model reconstruction, e.g. the corners of the walls on a building. For each feature a point, which is often referred to as a locator, is triangulated from the image coordinates given by the user. To create a locator, the user simply clicks the same feature in a number of images, and the 3D position is recovered using the calibration data of the images. From the set of locators, the shape of the 3D model can be reconstructed e.g. by creating polygon faces using the locators as vertices, or a CAD model can be reconstructed using the locators as guides. An example of a manual reconstruction method is the one developed in [19], where certain geometric constraints, such as orthogonality and distance equality, between locators are specified by the user. In another method for interactive architectural reconstruction, vanishing points estimated from detected lines in the images are used in combination with the feature point cloud for snapping geometry created by the user [38]. These techniques are combined in the commercial application Autodesk ImageModeler [1].

Also a number of automatic reconstruction methods exist, although they have limitations both regarding the types of objects that can be reconstructed and the visual quality of the reconstructions. One example is the method developed in [36], where the result from intermediate processing is used to rectify the input images and then calculate dense depth maps of the scene. From the depth maps, a detailed 3D model is then reconstructed. This method is used



(a) (b) (c) (d)

Figure 2.6: Some steps in model reconstruction using ProFORMA [33]. a) The object rotated in front of camera. b) Obtained point cloud. c) Mesh obtained from carving a Delaunay tetrahedralisation of the point cloud. d) Reconstructed 3D model.

in a system for reconstructing 3D models of urban environments from video in real-time [32]. Recently a real-time model reconstruction method named Probabilistic Feature-based On-line Rapid Model Acquisition (ProFORMA) has been published [33]. In this system, the user rotates an object in front of a stationary video camera, and the system simultaneously tracks and reconstructs a 3D model of the object. In figure 2.6 some steps in the process are illustrated.

One clear benefit of using photogrammetry for 3D model reconstruction is that the appearance of the model is available in the images that are already used for reconstruction of the shape. Thus textures for the model can be extracted without additional data acquisition. Although mapping the images to the reconstructed geometry is simple due to the known relationship between images and geometry, the texture of a surface may be available in more images, and may be partly occluded in some of them. This poses a problem of merging different images together to obtain a complete texture of the surface, and avoiding artifacts from occlusions. An elegant solution to this problem based on graph cuts has been developed in [38].

Using photogrammetry for 3D model reconstruction can, like using 3D scanning techniques, save a lot of work compared to manual reconstruction, especially during data acquisition. Data acquisition is simpler than for 3D scanning, because it involves nothing more than capturing images using a camera. It is still necessary, however, to ensure that all surfaces are covered by the captured images, and that the images overlap for feature matching. Some camera calibration may be needed depending on the specific reconstruction method used, but typically this involves little work compared to calibrating 3D scanning equipment, because the calibration parameters can be estimated from the images themselves. From the captured images, both structure and motion is recovered, and this result is used as the basis for various model reconstruction methods. Methods for model reconstruction range from manual to automatic, and the choice depends largely on the desired quality and purpose of the final model. Finally, cost is a parameter where photogrammetry has an advantage over 3D scanning techniques. Compared to e.g. a time-of-flight laser scanner, a digital camera is very inexpensive.

	Acquisition	Intermediate	Reconstruction	Notes
Manual	**Manual measurements**	Notes and drawings	Manual: • Polygon mesh model • Surface model • CAD model Textures from photographs.	No special equipment is required. High workload and user interaction. No cost.
3D Scanning	**Laser scanning** • Time-of-flight • Triangulation **Structured light**	Multiple scans ↓ Registration ↓ Point cloud	Automatic: • Point cloud → mesh • Mesh → surface model User assisted: • CAD model Textures from photographs.	Requires advanced equipment. Medium workload, and low to medium user interaction. High cost.
Photogrammetry	**Photography** **Video capture**	Images ↓ Structure and motion (sparse point cloud, and calibrated cameras)	Automatic: • Point cloud → dense depth maps → mesh • Mesh → surface model User assisted: • Polygon mesh model • CAD model Textures from images.	Requires a camera. Medium workload, and low to medium user interaction. Low cost.

Table 2.1: Overview of 3D model reconstruction methods. The arrows (↓ and →) indicate conversion of data. The cost in the last column is based solely on the price of equipment, and does not account for any work involved in the process.

2.3 Method Selection

The initiating problem in section 1.1 asks how a 3D model for interactive visualization of a real world building can be reconstructed in a simple manner. The purpose of this section is to find the 3D model representation and reconstruction method which best answers this question based on the analyses in section 2.1 and 2.2 respectively.

In the following, the applicability of each of the analyzed reconstruction methods is discussed with respect to the initiating problem. In section 2.3.1 the primary focus is on the data acquisition process, and a specific method for data acquisition is selected. In particular the process is treated in the context of a real estate agent preparing a property for sale. Then in section 2.3.2 a specific method for reconstructing the 3D model from the acquired and processed data is selected. This choice is based on a balance between the level of user interaction during reconstruction and the final purpose of the reconstructed model. The ultimate goal would be a system, which allows any real estate agent to easily reconstruct 3D models of buildings, such that they can be used for interactive online presentation. Although some of the analyzed reconstruction methods are reaching for this goal, there is still lots of room for improvement.

2.3.1 Data Acquisition

As discussed in chapter 1, the marketing material for a property for sale typically consists of an informative description of the property, one or more photographs of the exterior and interior of the building, and a floor plan. To obtain this information, the real estate agent needs to visit the property to capture photographs and take measurements for the floor plan. If in addition a 3D model of the building must be reconstructed, it is important that the data acquisition process fits well with the existing practice when visiting the property. In the following each of the data acquisition methods analyzed in section 2.2 are evaluated in this context.

Manual If a floor plan of the building does not exist, some measurements are needed for creating one. Compared to taking measurements for a floor plan, however, a vast amount of measurements are required for reconstructing a 3D model of the building. Furthermore the 3D model must be modeled manually from the measurements, and this requires a lot of work.

3D Scanning To reduce the amount of manual measurements that must be taken, 3D scanning techniques may be employed. This also allows taking advantage of user assisted or automatic reconstruction methods, and therefore might save time compared to manual reconstruction. Not all 3D scanning techniques are suitable for scanning buildings, however. Triangulation laser scanners and structured light techniques only have limited range, and thus are not suited for larger objects such as buildings. This leaves time-of-flight laser scanning as an option, but such equipment is very expensive. Several scans of the building are necessary, and each scan takes time and may need on-site calibration. Therefore 3D scanning is considered an intrusive method in the workflow of a real estate agent.

Photogrammetry An alternative to 3D scanning which also reduces the amount of manual measurements necessary is using photogrammetry. 3D model reconstruction using photogrammetry may also take advantage of user assisted or automatic methods, and therefore is an attractive alternative to manual reconstruction. Image data may be acquired using photography or video capture, and the range is not limited to objects of a specific size. As a real estate agent already brings a camera for capturing photographs of the building, using this camera for data acquisition is a simple, cost-neutral solution. Still, reconstruction using photogrammetry requires several overlapping images of the building, so the workload of the real estate agent will increase.

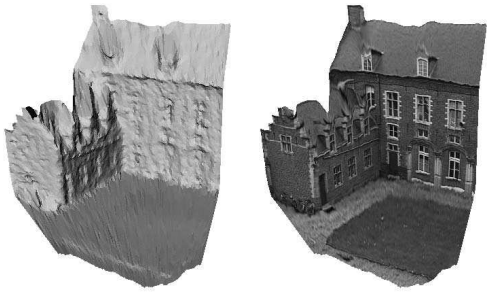
Based on the above discussion it is assessed, that capturing photographs for 3D model reconstruction using photogrammetry is the simplest, and most effective method for data acquisition. Although several photographs are needed in addition to the typical marketing photographs, this data acquisition method does not change the workflow of the real estate agent significantly. Moreover this solution is cost-neutral as opposed to investing in expensive 3D scanning equipment, and little or no on-site calibration is required. An additional advantage of using photogrammetry is that textures for the reconstructed model are available in the captured images, whereas additional photographs would be needed for manual reconstruction or 3D scanning.

2.3.2 Model Reconstruction

Having selected photography as the data acquisition method, in this section a specific method for model reconstruction using photogrammetry is selected. From the summary in table 2.1 it is seen that reconstruction using photogrammetry can result in any of the 3D model representations analyzed in section 2.1, i.e. polygon mesh model, surface model, and solid CAD model. According to the initiating problem, the final purpose of the reconstructed 3D model is interactive visualization, and therefore a model reconstruction method resulting in a polygon or triangle mesh model is preferable, because this type of model is directly supported by modern graphics rendering hardware. The additional abstraction provided by the other model representations is not necessary in this situation. For reconstructing a polygon mesh model using photogrammetry, both automatic and user assisted methods exist, see table 2.1. These methods are analyzed in the following in order to select the reconstruction method that best fits with the initiating problem.

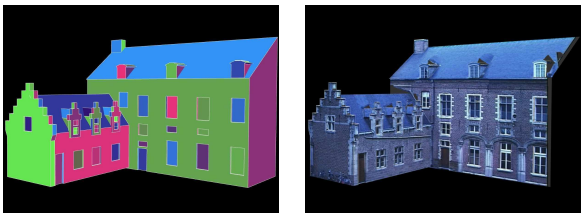
Automatic In section 2.2.3 two automatic methods for model reconstruction were discussed, one being ProFORMA which is based on interactive video capture [33]. This method is deemed unsuitable, because it is incompatible with the selected data acquisition method. Specifically, ProFORMA requires the camera to be static and the method is interactive, and thus it is necessary to bring a computer to the field during data acquisition. The other automatic method discussed is based on estimating dense depth maps of the scene, and reconstructing a detailed 3D model from these as in [36].

Recall that the result of intermediate processing in photogrammetry is structure and motion, i.e. estimated 3D positions of the set of extracted image features and estimated extrinsic and intrinsic calibration of the cameras. From



(a) (b)

Figure 2.7: Automatic triangle mesh reconstruction from a single dense depth map [28]. a) Geometric model. b) Textured model.



(a) (b)

Figure 2.8: User assisted polygon mesh reconstruction [38]. a) Geometric model. b) Textured model.

this it is possible to rectify the input images, such that the pixels in a scanline of one image map to pixels in the corresponding scanline of another image. By matching all pixels in the scanlines of the two images, a dense depth map can be estimated from the disparity of the matched pixels [36]. This approach has problems with reflective surfaces such as windows, because it is difficult to find correct matches between different images in such areas. In figure 2.7 an example of a triangle mesh reconstructed from a single dense depth map is shown.

Such reconstruction, however, is not sufficient because the result is not a complete 3D model. A dense depth map estimated from two images is similar to the result of a single 3D scan, and thus the obtained depth maps need to be registered into a single point cloud, in the same fashion as when using 3D scanning, in order to reconstruct a complete 3D model. Refer to section 2.2.2 for examples on how a triangle mesh model can be obtained from a point cloud.

User Assisted The user assisted model reconstruction methods discussed in section 2.2.3 all start from the structure and motion recovered in intermediate processing. From the calibrated images the 3D positions of a set of locators are triangulated from corresponding image coordinates given by the user. Based on the locators the user reconstructs a polygon mesh model, and various intelligent measures can be employed to simplify this process, see the discussion in section 2.2.3. User assited reconstruction directly results in a 3D model, and there is no need for additional point cloud registration etc. In figure 2.8 an example of a polygon mesh model obtained with the user assisted reconstruction method developed in [38] is shown.

(a) (b) (c)

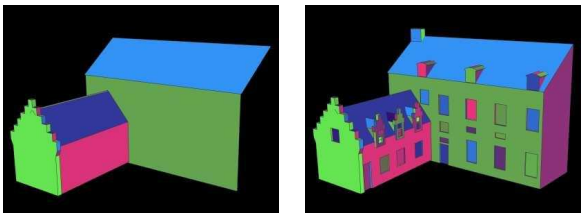
Figure 2.9: Three images of a recessed window captured from different angles to show the effect of changing viewpoint. a) The left edge of the window frame is hidden behind the wall. b) The whole frame is visible when seen from the front. c) The right edge of the frame is hidden behind the wall.

The 3D models reconstructed using automatic and user assisted methods are very different in character. Typically automatically reconstructed models are highly tessellated, and contain lots of geometric detail as is evident in figure 2.7a. These details are not necessarily due to the original scene, but may be caused by noise and imprecision in the reconstruction process, and thus may result in poor visual quality of the model. The structure of buildings normally satisfies geometric constraints such as walls being planar and vertical, windows being rectangular etc., and it is difficult to enforce such constraints in the point cloud obtained from dense depth maps.

The result of user assisted reconstruction typically is polygon mesh models that have much simpler structure than what is obtained using automatic methods, see figure 2.8a. Due to the low resolution of the reconstructed model, the noise that is typical of automatically reconstructed models is not present, and constraints such as walls being planar are automatically enforced by the inherent planarity of the polygons of the model. Of course the low resolution may cause details to be lost, but the user can identify the features of the building that are important, and add extra detail in these places.

This is the approach used in [38], where the user typically starts by reconstructing a coarse model of the building consisting of large polygons representing the walls, roof etc. Then this model is refined by manually adding details such as recessed windows and doors. E.g. a window region is marked by the user on the polygon representing a wall, and this region is recessed by an amount specified interactively by the user. Time is spent adding such geometric details because it makes the model look more realistic, especially when seen from different angles. In figure 2.9 the effect of changing viewpoint is shown for a window of a real building.

Although in [38] such cutouts for windows can be copied for adding multiple windows of the same dimensions, adding these details is still a time consuming process. This is evident from the progressive states of a reconstructed model shown in figure 2.10. Nearly $\frac{3}{4}$ of the time is spent adding details to the coarse model, which is reconstructed within the first 4 minutes [38]. From this it is clear that user assisted reconstruction can be simplified and made faster by developing an automatic method for adding details such as recessed windows and doors.



(a) (b)

Figure 2.10: Progression of a model during user assisted reconstruction in [38]. a) A coarse model is reconstructed after 4 minutes of modeling. b) Details are added after 15 minutes of modeling.

From the above discussion, polygon mesh models reconstructed with user assisted methods are judged qualitatively to have higher visual quality than automatically reconstructed models. As discussed in chapter 1 the appearance of a product is often a key selling point, and as the final purpose of the model is interactive presentation, the visual quality of the model has high priority. The level of user interaction in existing user assisted reconstruction methods, however, is significantly higher than that of the automatic methods. As demonstrated by the method developed in [38], it is possible to make user assisted reconstruction simple and effective, but still this is far from allowing any real estate agent to easily reconstruct 3D models of buildings. In particular the process of adding details such as recessed windows and doors to the coarse model is time consuming, and could potentially be optimized by developing an automatic method for this purpose. Based on this discussion, user assisted reconstruction of a coarse polygon mesh model, combined with a novel automatic method for adding façade details, is selected as the model reconstruction method, which best fits the initiating problem.

2.4 Conclusion

In this chapter the initiating problem defined in section 1.1 has been analyzed. In section 2.1 it was defined that a 3D model contains information about both the shape and textures of an object, and three possible representations of the shape of a 3D model were analyzed: polygon mesh models, surface models, and solid CAD models. Section 2.2 then contains an analysis of different methods for reconstructing 3D models from real world objects. In general reconstruction methods consist of three steps: data acquisition, intermediate processing, and model reconstruction. These steps were analyzed for three reconstruction methods, namely manual reconstruction, reconstruction using 3D scanning, and reconstruction using photogrammetry. The analysis of reconstruction methods is summarized in table 2.1 on page 17.

Based on these analyses, in section 2.3 the reconstruction method which best fits with the initiating problem was selected. The selected reconstruction method is photogrammetry, which uses photography for data acquisition. This method was selected, because it has many advantages compared to the other methods analyzed. In particular the data acquisition process integrates well with the existing workflow of a real estate agent preparing a property for sale,

and it does not require expensive equipment other than a camera, which is already available. Intermediate processing in photogrammetry consists of estimating structure and motion, i.e. estimating the 3D positions of a set of features detected in the input images and the calibration parameters of the cameras used for capturing the input images, and this process can be automated. For model reconstruction a two step approach was selected. First step consists of user assisted reconstruction of a coarse polygon mesh model. This coarse model is then refined automatically by adding façade details such as recessed windows and doors, using a novel method developed in this project. In the following chapter, the specific problem of the project is defined in more detail.

Chapter 3

Problem Formulation

In chapter 1, the need for simpler methods for 3D model reconstruction of real world buildings was established, and the initiating problem of the project was defined. To answer the initiating problem the problem was analyzed in chapter 2, and based on this analysis the reconstruction method which best fits with the initiating problem was selected. The purpose of this chapter is to define the specific problem of the project in more detail.

The selected reconstruction method is photogrammetry, where data acquisition consists of capturing images of the building to reconstruct. During intermediate processing, structure and motion is recovered from the captured images, and finally this information is used for model reconstruction. Model reconstruction consists of user assisted reconstruction of a coarse model followed by automatic refinement, where façade details such as recessed windows and doors are added to the model using a novel method developed in this project. This leads to the following problem formulation for the project:

> *Using photogrammetry and user assisted model reconstruction, how is a system for reconstructing a textured polygon mesh model of a real world building from an unordered set of images developed, and how can the process of adding façade details to the model be automated?*

In the following section, a more detailed concept for the system is developed. The chapter concludes with the problem delimitation in section 3.2, which specifies the areas of the problem on which focus lies in the project.

3.1 System Concept

In the following a concept for how the selected reconstruction method can be applied is developed. This concept covers the whole reconstruction process on an overall level, and serves as an overview of the proposed reconstruction method.

The input to the system is an unordered set of digital images of the building to reconstruct. Unordered means that there is no particular order in which the images must be supplied to the system. It is only required that there is overlap between the images, and that they capture all surfaces of the building for which reconstruction is wanted.

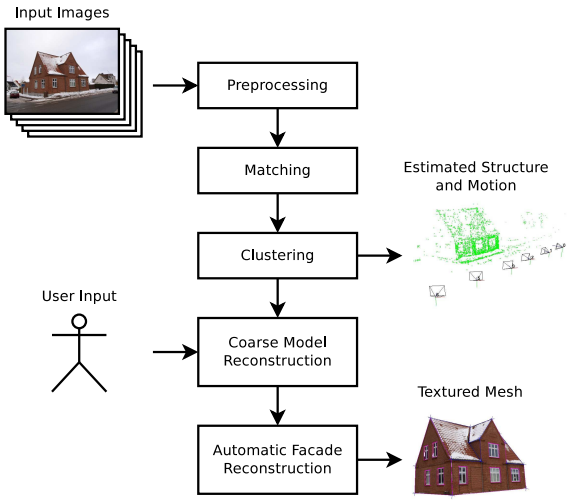


Figure 3.1: Overview of the steps in the proposed reconstruction method, with input and output of the system indicated.

The primary output of the system is a reconstructed 3D model of the building in the form of a textured polygon mesh model. Henceforth polygon mesh model is referred to simply as mesh. In addition to the model, intermediate results such as structure and motion recovered during intermediate processing is available.

3.1.1 Method

The concept developed here is based on the analysis in section 2.2.3 and on the reconstruction method selected in section 2.3.2. The proposed method consists of the following steps: preprocessing, matching, clustering, coarse model reconstruction, and automatic façade reconstruction. All steps are automatic, except for coarse model reconstruction, which requires user assistance. The steps are illustrated in figure 3.1, and each of them is explained on an overall level in the following. The first three steps correspond to intermediate processing in the previous analysis, while the last two steps correspond to model reconstruction.

Preprocessing In this step the input images are loaded. Then the intrinsic calibration parameters are estimated for all images, and keypoints in all images are detected.

Matching Matching is performed for all pairs of input images, and the goal is to estimate the relative motion between the cameras in each pair if possible. For each pair, a robust set of corresponding keypoints in the two images is identified. Using the corresponding keypoints and the known intrinsic calibration parameters of the images, the relative motion is recovered.

Clustering The goal of clustering is to recover structure and motion, i.e. both positions of keypoints in space and extrinsic camera parameters, for all input images. Clustering is performed by starting from the best matching image pair,

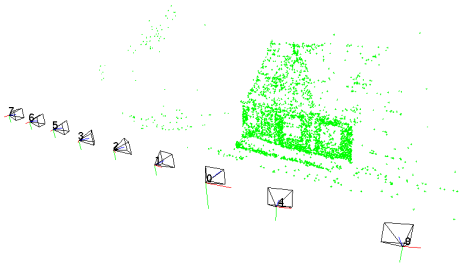


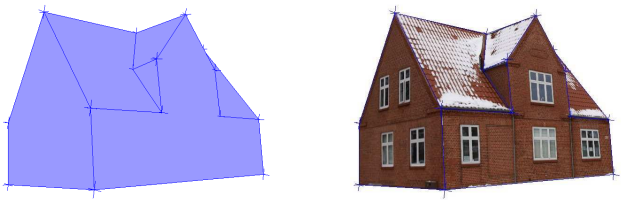
Figure 3.2: Structure and motion recovered during clustering. Estimated keypoint positions are shown as green dots, and estimated camera poses for the input images are shown as pyramids.

and iteratively adding images using the relative motion recovered for image pairs in the matching step. In this process keypoints are triangulated, and the recovered structure and motion is optimized using bundle adjustment. In figure 3.2 an example of structure and motion recovered during clustering is shown.

Coarse Model Reconstruction The next step in the process is user assisted reconstruction of a coarse 3D model of the building. Here a coarse model is defined as a textured mesh which consists of polygons representing large planar surfaces of the building, i.e. walls, roof etc. In figure 3.3 an example of a coarse model is shown. Note that the coarse model does not contain details such as recessed windows and doors.

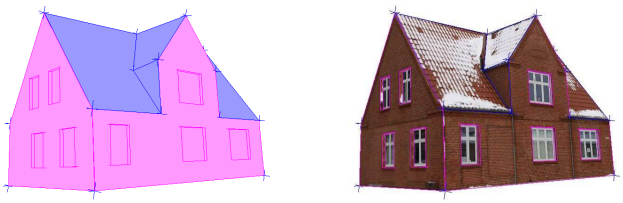
The reconstruction of a coarse model is done interactively in two steps. First a set of locators, which represent 3D positions of selected features of the building, e.g. the corners of a wall, is defined. A locator is defined by clicking the same feature in two or more of the input images. The 3D positions of the locators are triangulated using the camera calibration information available from the clustering step. Second, polygons that span the planar surfaces of the building are defined by clicking the locators to use as corners of the polygons.

The user is responsible only for defining the shape of the coarse model. The appearance of the model, that is textures for the individual polygons, is automatically extracted from the input images in this step.



(a) **(b)**

Figure 3.3: An example of a coarse model that is reconstructed with user assistance. Large planar surfaces such as walls and roof are represented using polygons. a) Geometric model. b) Textured model.



(a) (b)

Figure 3.4: An example of a model that has been refined using automatic façade reconstruction for selected polygons (magenta) of the coarse model. a) Geometric model. b) Textured model.

Automatic Façade Reconstruction This is the final step of the proposed reconstruction method, and it is based on the coarse model that is reconstructed in the previous user assisted step. The purpose of this step is to automatically refine the coarse model by adding façade details such as recessed windows and doors to selected polygons. In existing user assisted reconstruction methods, this task is cumbersome and time-consuming, and by developing a novel method for automatically reconstructing façade details, an advantage compared to existing methods is achieved. The developed method utilizes the information obtained in the previous steps for automatic façade reconstruction. In figure 3.4 an example of a refined model is shown.

3.2 Problem Delimitation

In the previous section a concept for how the selected reconstruction method can be applied was developed. In this project the proposed reconstruction method is implemented as a proof of concept system, which covers the whole process from data acquisition to model reconstruction. Although a complete system is developed, this system is not meant to be a full application, which can be used directly by real estate agents for reconstructing buildings. Its purpose is instead to investigate ways to improve existing architectural reconstruction methods, and to demonstrate the feasibility of the proposed reconstruction method.

An area that has received little focus in previous work is that of automatically reconstructing façade details of a reconstructed coarse model, and as previously discussed existing methods can be improved by developing a method for this purpose. Therefore the primary contribution in this project is development of a novel method for automatically refining a coarse model with façade details such as recessed windows and doors. As is evident from the analysis in chapter 2, much previous work related to solving the structure from motion problem exists, and this project is not meant to provide any revolutionary improvements in this field. However, as the proposed reconstruction method depends largely on solving this problem, methods for recovering structure and motion from images are also treated in detail.

The remaining part of the report is divided into two parts: method, and results and discussion. In the method part, the development of the proposed reconstruction method is documented. The five chapters of this part, i.e. chapters 4 through 8, correspond directly to the steps of the concept developed in section 3.1. In each of these chapters, the specific problem of that step is analyzed, methods for solving the problem are developed, and the developed methods are evaluated.

The results and discussion part of the report documents the results obtained from testing the three main parts of the system. The results are discussed, and a conclusion for the project as a whole is provided. In chapter 9, an introduction to the performed system evaluation is given, and the used test data is presented. In chapter 10, the results of recovering structure and motion for the data sets are documented and discussed. Then the results of user assisted coarse model reconstruction for the data sets are documented and discussed in chapter 11. And in chapter 12, the results of applying the developed method for automatic façade reconstruction are documented and discussed. Finally, chapter 13 contains the overall conclusion and perspectives of the project.

Part II

Method

Chapter 4

Preprocessing

In this chapter the preprocessing step of the proposed reconstruction method is documented. As discussed in section 3.1.1, in this step the input images are loaded into the system, intrinsic calibration for the images is estimated, and keypoints in the images are detected.

For 3D reconstruction from images to be possible, a mathematical model of a camera is needed. Therefore in section 4.1, the camera model used in this project is introduced. Then in section 4.2, a method for estimating the intrinsic calibration parameters of the images is developed. Finally, in section 4.3, the problem of robustly detecting keypoints in the input images is analyzed, and a suitable algorithm for this purpose is selected.

4.1 Camera Model

In this section the mathematical camera model used in this project is introduced. This model covers both the intrinsic and extrinsic calibration parameters. The intrinsic calibration of a camera may include parameters describing lens distortion. In this project, however, any lens distortion present in the input images is assumed to be negligible, and hence lens distortion is not part of the used camera model. If significant lens distortion is present in the input images, any known method for correcting this can be applied to the images before they are supplied to the system, see e.g. [17]. The camera model described in the following is based on [23].

4.1.1 Basic Pinhole Camera

A basic pinhole camera can be modelled as the central projection of points in space onto the image plane. Let the centre of projection be the origin of an Euclidean coordinate system, and let the image plane be $Z = f$. Then a point in space $\mathbf{X} = [X\ Y\ Z]^\mathsf{T}$ is mapped to the point $\mathbf{x}$ in the image plane where the line through $\mathbf{X}$ and the centre of projection intersects the image plane, see figure 4.1. By similar triangles, the point $\mathbf{X}$ in space is mapped to the point $\mathbf{x} = [fX/Z\ fY/Z\ f]^\mathsf{T}$ which lies in the image plane. Ignoring the last

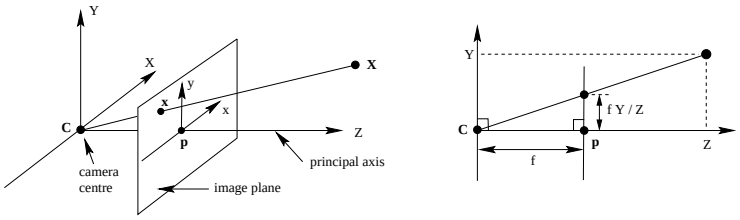


Figure 4.1: The geometry of a basic pinhole camera. **C** is the centre of projection, also referred to as the camera centre, and **p** is the principal point [23].

image coordinate, this becomes

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \mapsto \begin{bmatrix} fX/Z \\ fY/Z \end{bmatrix}, \tag{4.1}$$

which is a mapping from Euclidean 3-space $\mathbb{R}^3$ to Euclidean 2-space $\mathbb{R}^2$. The centre of projection is also called the camera centre, and the line from the camera centre perpendicular to the image plane is called the principal axis. The intersection of the principal axis and the image plane is called the principal point.

By representing points in space and image points using homogeneous vectors, the mapping in (4.1) becomes the simple linear mapping

$$\begin{bmatrix} fX \\ fY \\ Z \end{bmatrix} = \begin{bmatrix} f & & & 0 \\ & f & & 0 \\ & & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}. \tag{4.2}$$

The homogeneous vectors $[fX\ fY\ Z]^\mathsf{T}$ and $[fX/Z\ fY/Z\ 1]^\mathsf{T}$ both represent the same point, and thus Euclidean image coordinates can be obtained by perspective division.

The matrix in (4.2) may be written as $\mathrm{diag}(f, f, 1)[\mathrm{I}\ \mathbf{0}]$, where $\mathrm{diag}(f, f, 1)$ is a diagonal matrix, and $[\mathrm{I}\ \mathbf{0}]$ is a matrix consisting of two blocks, a 3×3 identity matrix and the zero column vector. Now let $\mathbf{X} = [X\ Y\ Z\ 1]^\mathsf{T}$ be the homogeneous 4-vector representing a point in space, and let $\mathbf{x}$ be the homogeneous 3-vector representing the associated image point. Then by introducing P as the 3×4 camera projection matrix, (4.2) can be written compactly as

$$\mathbf{x} = \mathrm{P}\mathbf{X}, \tag{4.3}$$

where $\mathrm{P} = \mathrm{diag}(f, f, 1)[\mathrm{I}\ \mathbf{0}]$ for the basic pinhole camera model.

4.1.2 Digital Cameras

In this section the basic pinhole model is extended to a general model which is able to represent e.g. digital cameras, where the image is formed by pixels. In general the origin of coordinates in the image plane does not coincide with the principal point as assumed in (4.1), so there is a mapping

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \mapsto \begin{bmatrix} fX/Z + p_x \\ fY/Z + p_y \end{bmatrix}, \tag{4.4}$$

where $[p_x\ p_y]^\mathsf{T}$ are the coordinates of the principal point in the image plane. In homogeneous coordinates this becomes

$$\begin{bmatrix} fX + Zp_x \\ fY + Zp_y \\ Z \end{bmatrix} = \begin{bmatrix} f & & p_x & 0 \\ & f & p_y & 0 \\ & & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}. \tag{4.5}$$

Introducing K as the camera calibration matrix

$$\mathrm{K} = \begin{bmatrix} f & & p_x \\ & f & p_y \\ & & 1 \end{bmatrix}, \tag{4.6}$$

then (4.5) can be written concisely as

$$\mathbf{x} = \mathrm{K}[\mathrm{I}\ \mathbf{0}]\mathbf{X}. \tag{4.7}$$

For digital cameras, the image is formed on a sensor and represented as pixels. The sensor may have non-square pixels, and measuring image coordinates in pixels, adds an extra possibly non-uniform scale. Let m_x and m_y be the number of pixels per unit distance in image coordinates in the x and y directions. Now the camera calibration matrix becomes

$$\mathrm{K} = \begin{bmatrix} \alpha_x & & x_0 \\ & \alpha_y & y_0 \\ & & 1 \end{bmatrix}, \tag{4.8}$$

where $\alpha_x = fm_x$ and $\alpha_y = fm_y$ represent the focal legnth of the camera in terms of pixel dimensions in the x and y directions respectively. Similarly, $[x_0\ y_0]^\mathsf{T}$ are the coordinates of the principal point in terms of pixel dimensions, with $x_0 = m_x p_x$ and $y_0 = m_y p_y$.

In this model it is assumed, as is the case for normal digital cameras, that there is no skew of the image sensor axes. Thus the camera calibration matrix K defined in (4.8) contains all relevant intrinsic calibration parameters of the camera.

4.1.3 Camera Position and Orientation

Above it is assumed that the points in space are given in the camera coordinate frame, but typically points are expressed in terms of a world coordinate frame. The camera and world coordinate frames are related via a rotation and a translation defined by the orientation and position of the camera. A point in the world coordinate frame represented by the homogeneous 4-vector $\mathbf{X}_w$ is mapped to the corresponding point $\mathbf{X}_c$ in the camera coordinate frame by

$$\mathbf{X}_c = \begin{bmatrix} \mathrm{R} & \mathbf{t} \\ & 1 \end{bmatrix} \mathbf{X}_w, \tag{4.9}$$

where R is a 3×3 rotation matrix representing the orientation of the world coordinate frame relative to the camera coordinate frame, and $\mathbf{t}$ is a column 3-vector representing the position of the origin of the world coordinate frame

in the camera coordinate frame. In other words, the matrix R and the vector $\mathbf{t}$ are the extrinsic calibration parameters of the camera.

Combining (4.9) with (4.7), the mapping from the homogeneous world point $\mathbf{X}$ to the homogeneous image point $\mathbf{x}$ can be written as

$$\mathbf{x} = \mathrm{K}[\mathrm{R}\ \mathbf{t}]\mathbf{X}. \tag{4.10}$$

From this it is seen that for a camera model which can represent digital cameras, the camera projection matrix is

$$\mathrm{P} = \mathrm{K}[\mathrm{R}\ \mathbf{t}], \tag{4.11}$$

where the camera calibration matrix K has the form given in (4.8). This is the camera model used in this project.

4.2 Intrinsic Calibration

To recover structure and motion using the process outlined in section 2.2.3, the intrinsic calibration parameters for each of the input images must be known in advance or estimated. In this section two methods for estimation of the camera calibration matrix K given in (4.8) are treated. One method analyzes images of a calibration pattern captured by the camera, and the other method uses information available in the image files created by modern digital cameras.

For images captured using the same camera with a constant focal length, the camera calibration matrix does not change. Therefore if all input images are captured with the same camera, and without altering the zoom level, it is possible to estimate a common K for all images. This intrinsic calibration can be achieved e.g. by capturing a sequence of images of a calibration pattern from different perspectives with camera settings identical to those used when capturing the input images. From the images of the calibration pattern, the camera calibration matrix can be estimated using a method such as [17].

Although high precision can be achieved using the above method, it is impractical in the context of this project. It may be difficult to capture some parts of a building without zooming, and ensuring that all images captured by a real estate agent in the field are captured with identical focal lengths may be hard. Using the above method, it would be necessary to perform intrinsic calibration of the camera for each focal length used in the input images, and this would quickly become labour intensive. Therefore a more flexible solution is preferred.

Modern digital cameras store a lot of metadata in addition to the captured image when creating an image file. This metadata includes the camera settings used when capturing the image, e.g. aperture, shutter speed, and focal length, but also static information such as the make and model of the camera is stored. This information is stored using the Exchangeable Image File Format (EXIF), see appendix A for more details. By extracting the focal length from the EXIF data available in the input image files, it is possible to obtain a fairly good estimate of the camera calibration matrix, and below a method based on this approach is developed.

4.2.1 Estimation from EXIF Data

The intrinsic calibration parameters that must be estimated for each input image are the focal length measured in pixels α_x and α_y, and the coordinates of the

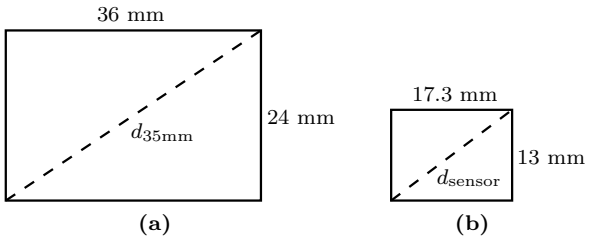


Figure 4.2: Illustration of the dimensions of traditional 35 mm film format and an image sensor. a) 135 film full-frame [40]. b) Olympus E-520 image sensor [7].

principal point $[x_0 \ y_0]^\mathsf{T}$ in pixels. These four parameters define the camera calibration matrix K given in (4.8).

With no previous knowledge, the best guess for the location of the principal point is the centre of the image. Given the width w and height h of the image in pixels, the estimated coordinates of the principal point simply become

$$\begin{bmatrix} x_0 \\ y_0 \end{bmatrix} = \begin{bmatrix} w/2 \\ h/2 \end{bmatrix}. \tag{4.12}$$

Under the assumption that sensor pixels are square, let $\alpha = \alpha_x = \alpha_y$ be the focal length measured in pixels. To estimate α, the following additional information is required: the focal length f measured in mm, and the crop factor c of the image sensor. For modern digital cameras, f can be extracted from the EXIF data in the image file as discussed above. The crop factor depends on the make and model of the camera, and it describes the size of the image sensor relative to the traditional 35 mm film format. More specifically the crop factor is the ratio of the diagonal of a 35 mm frame to the diagonal of the sensor in question [44]. That is

$$c = \frac{d_{35\text{mm}}}{d_{\text{sensor}}}, \tag{4.13}$$

where $d_{35\text{mm}}$ and d_{sensor} are the diagonals measured in mm of a 35 mm frame and the image sensor respectively.

In figure 4.2a the dimensions of a 35 mm frame is shown, and its diagonal is calculated as

$$d_{35\text{mm}} = \sqrt{36^2 + 24^2} \approx 43.27 \text{ mm}. \tag{4.14}$$

In figure 4.2b the dimensions of the image sensor in an Olympus E-520 is shown. This is the camera model used for capturing images in this project, and this sensor has a crop factor of 2. Inserting the crop factor $c = 2$ into (4.13) it is seen that for this camera $d_{\text{sensor}} \approx 21.63$ mm.

Still assuming square pixels, let $m = m_x = m_y$ be the number of pixels per unit distance in image coordinates. As f is measured in mm, the unit of m becomes pixels/mm. Introducing d_{image} as the number of pixels along the image diagonal given by

$$d_{\text{image}} = \sqrt{w^2 + h^2}, \tag{4.15}$$

m is calculated as

$$m = \frac{d_{\text{image}}}{d_{\text{sensor}}}. \tag{4.16}$$

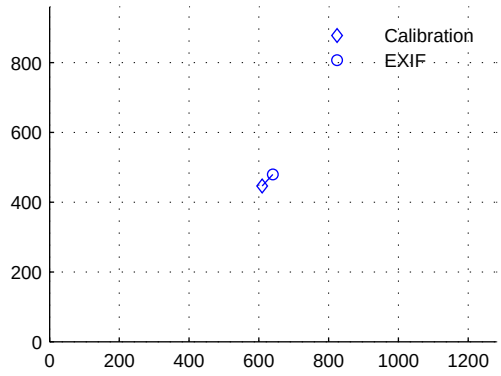


Figure 4.3: The principal points obtained from calibration using [17] and estimated from EXIF data.

Now with $\alpha = fm$, cf. section 4.1.2, from above it is seen that

$$\alpha = f \cdot \frac{\sqrt{w^2 + h^2}}{d_{35\text{mm}}/c}, \tag{4.17}$$

and finally, the estimated camera calibration matrix becomes

$$\mathrm{K} = \begin{bmatrix} \alpha & & w/2 \\ & \alpha & h/2 \\ & & 1 \end{bmatrix}. \tag{4.18}$$

4.2.2 Evaluation

For evaluating the results of estimating the camera calibration matrix from the EXIF data available in the input images, intrinsic calibration using [17] has been performed from a set of images of a calibration pattern. The camera calibration matrices K_c obtained from this calibration and K_e estimated using the method developed above are

$$\mathrm{K}_c = \begin{bmatrix} 1009 & 0 & 609 \\ 0 & 1009 & 447 \\ 0 & 0 & 1 \end{bmatrix} \qquad \mathrm{K}_e = \begin{bmatrix} 1035 & 0 & 640 \\ 0 & 1035 & 480 \\ 0 & 0 & 1 \end{bmatrix}, \tag{4.19}$$

where the dimensions of the images are 1280×960 pixels.

The difference in the estimated principal points of the two methods is approximately 45 pixels, and in figure 4.3 the principal points for both methods are shown. If it is assumed that the calibration obtained using [17] is correct, estimation from EXIF data leads to an error in the principal point equal to 2.8% of the image diagonal, and the error in the estimated focal distance is 2.6%. In this project, even though estimation of the camera calibration matrix from EXIF data may not be perfect, this method is chosen due to the increased flexibility.

4.3 Detection of Keypoints

The purpose of the matching step, see chapter 5, is estimation of the relative pose of the cameras in each pair of input images. Part of this process is to identify a robust set of corresponding keypoints in the two images. A prerequisite for identifying such sets is that keypoints are detected in each of the input images, and therefore detection of keypoints is part of the preprocessing step.

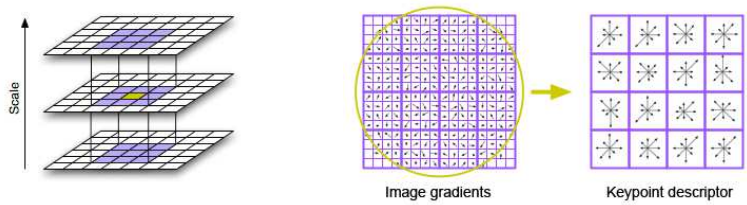
It is essential for establishing correspondence between keypoints during the matching step that keypoints detected in one image are likely to be detected in another image of the same scene; this is referred to as repeatability. Furthermore it is important that keypoints originating from images of the same scene feature can be matched; this is referred to as distinctiveness. Both repeatability and distinctiveness of keypoints is necessary, even when the images are captured from different perspectives and under varying lighting conditions. More specifically detection of keypoints must be invariant to changes such as image scale and rotation, changes in 3D viewpoint, presence of noise, e.g. sensor noise and image compression, and changes in illumination.

Lots of previous work exists on developing algorithms which aim to robustly detect keypoints that are invariant to these changes. Two algorithms in particular perform well on a wide variety of images, namely the Scale Invariant Feature Transform (SIFT) algorithm [27], and the Speeded-Up Robust Features (SURF) algorithm [14]. The following section contains a short comparison of the two algorithms, in order to select the algorithm best suited for this project.

4.3.1 Comparison of SIFT and SURF

The SIFT and SURF algorithms are similar in the way that they both consist of the two main steps: detection of interest points, and extraction of descriptors for the neighbourhood of every interest point. Detection of interest points is the first step, which is identifies points in the image which are good candidates for distinctive keypoints. The performance of this step determines the repeatability. Extraction of descriptors is the second step, where the appearance of the neighbourhood of each interest point is encoded into a descriptor. The encoding influences robustness towards the various changes discussed above and the distinctiveness of keypoints. The difference between SIFT and SURF is how these two steps are performed.

SURF is the newest of the two algorithms, and it is inspired by SIFT, which is very well established. The goal of SURF is to obtain results that are on par with the best algorithms available, but with less computational complexity. The authors of SURF claim that their algorithm is faster than SIFT, while still obtaining the same keypoint quality with respect to repeatability and distinctiveness [14]. Comparisons of the two algorithms have been made, and to some degree they support this claim. In both [13] and [39] it is concluded that the quality of the detected keypoints is slightly better for SIFT, but the speed of SURF is much higher. However, one important difference between the algorithms is clear: The number of keypoints detected in an image is significantly higher for SIFT. In this project detecting a large number of keypoints is more important than speed, because the system has no real-time requirements, and recovery of structure and motion performs better with more keypoints. Therefore SIFT is selected for keypoint detection in this project.



(a) (b)

Figure 4.4: The two main steps of the SIFT algorithm. a) Detection of interest points using scale-space extrema detection. b) Extraction of keypoint descriptors from local image gradients.

4.3.2 Keypoint Detection Using SIFT

Given a grayscale image as input, the output of the SIFT algorithm is a set of keypoints detected in that image. On an overall level the two main steps of the SIFT algorithm are performed as follows. Detection of interest points is performed by searching over all image locations and scales of the image for extrema of a difference-of-Gaussian function, see figure 4.4a. The potential interest points that are identified are thus invariant to scale and orientation [27]. A keypoint is localized for each interest point, and an orientation is assigned based on local image gradient directions. Finally extraction of descriptors is performed by measuring the local image gradients at the detected scale, and transforming them into a representation that allows for significant levels of local shape distortion and changes in illumination [27], see figure 4.4b.

Each detected keypoint is described by its location, scale, orientation, and descriptor. Only the location, which is the image coordinates of the keypoint in pixels, and the descriptor, which is a 128 dimensional vector, are used in the later steps of structure and motion recovery.

4.3.3 Evaluation

In figure 4.5 an example of keypoint detection using SIFT is shown. The number of keypoints detected in an image depends a lot on the contents of the image, but generally more keypoints are detected in images with higher resolution. The images used for testing in this project all have a size of 1280×960 pixels, and the average number of keypoints detected per image is 6233. With a high number of keypoints, both the likelihood of identifying a large set of corresponding keypoints in the matching step, and the quality of the recovered structure and motion increases.

4.4 Conclusion

In this chapter, the preprocessing step of the proposed reconstruction method has been treated in detail. In section 4.1 the mathematical camera model used in this project was introduced. Then in section 4.2 different ways to estimate the intrinsic calibration of the input images were discussed, and a method based on extraction of the EXIF data available in the image files was developed. Finally

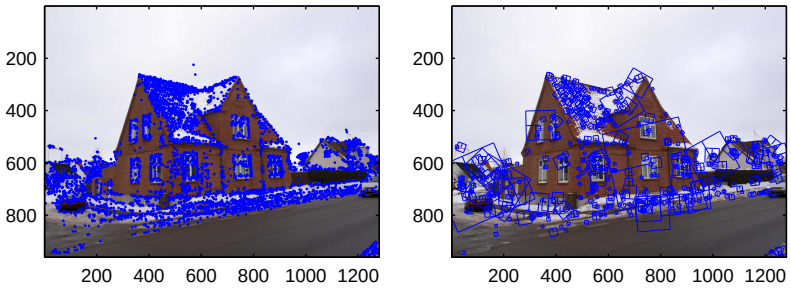


Figure 4.5: The 4966 keypoints detected by SIFT in a 1280 × 960 pixel image. a) Location of all detected keypoints. b) Orientation and scale of detected keypoints (only 10% are shown to avoid clutter).

in section 4.3, the SIFT algorithm was selected for robustly detecting keypoints in the input images. The SIFT algorithm was selected both due to its robustness and the high number of keypoints detected in images.

Chapter 5

Matching

This chapter contains documentation of the matching step in the proposed reconstruction method. The purpose of the matching step is to estimate the relative pose of the cameras in all pairs of input images if possible. This is achieved using the results obtained during preprocessing. For each pair of input images the overall approach is as follows. A robust set of corresponding keypoints in the two images is identified, and then using the corresponding keypoints and the intrinsic calibration of the images, the relative pose of the cameras is recovered.

The task of recovering the relative pose relies on constraints imposed by epipolar geometry, which describes the relation between two cameras, points in space, and the corresponding points in the images. Therefore this chapter begins with an introduction to epipolar geometry in section 5.1. Identification of a robust set of corresponding keypoints in two images is achieved in two steps. First an initial correspondence between the keypoints is obtained by matching the keypoint descriptors, and this process is treated in section 5.2. Second a robust correspondence between keypoints is established by removing outliers that do not satisfy the epipolar constraint of the cameras. The epipolar geometry of two cameras is described by a matrix referred to as the fundamental matrix. In this second step, both the fundamental matrix is estimated, and a robust set of corresponding keypoints, referred to as keypoint inliers, is obtained. This process is treated in section 5.3. Finally, having a set of keypoint inliers and knowing the fundamental matrix, the relative pose of the two cameras can be recovered, and this process is treated in section 5.4.

5.1 Epipolar Geometry

Epipolar geometry is the geometry of stereo vision. When two cameras view a scene, there are certain geometric relations between the 3D points observed in the scene and their 2D projections in the images. Epipolar geometry describes these relations, which are independent of scene structure. The following description is based on [23].

The fundamental matrix F is a 3×3 matrix having rank 2, which encapsulates these geometric relations for cameras with unknown intrinsic calibration. Let $\mathbf{X}$ be a point in 3-space, which is imaged as the point represented by the homogeneous 3-vector $\mathbf{x}$ in the first image and $\mathbf{x}'$ in the second image, then the

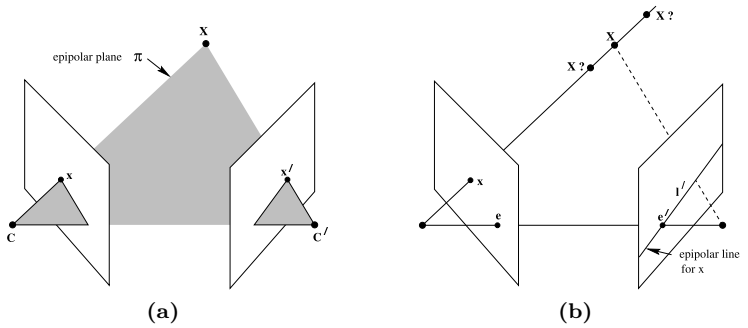


Figure 5.1: Epipolar geometry [23]. a) Two cameras with centres $\mathbf{C}$ and $\mathbf{C}'$ and their image planes. The camera centres, space point $\mathbf{X}$ and its image points $\mathbf{x}$ and $\mathbf{x}'$ all lie in the epipolar plane π. b) The space point $\mathbf{X}$ lies on the ray intersecting the first camera centre and the image point $\mathbf{x}$. This ray is imaged as the line $\mathbf{l}'$ in the second image, and hence $\mathbf{x}'$ must lie on this line.

image points satisfy the relation

$$\mathbf{x}'^{\mathsf{T}}\mathrm{F}\mathbf{x} = 0. \tag{5.1}$$

From the illustration in figure 5.1a it is seen that the image points $\mathbf{x}$ and $\mathbf{x}'$, the space point $\mathbf{X}$, and the camera centres $\mathbf{C}$ and $\mathbf{C}'$ are coplanar, and all lie in the epipolar plane π. All planes that coincide with the baseline joining the two camera centres are epipolar planes.

Given a point $\mathbf{x}$ in the first image, in the following it is explained how the point $\mathbf{x}'$ in the second image is constrained. The epipolar plane π can be defined by the baseline and the ray intersecting the camera centre $\mathbf{C}$ and the image point $\mathbf{x}$. From above it is known that the point $\mathbf{x}'$ lies in this plane, and hence the intersection between π and the second image plane determines a line $\mathbf{l}'$ on which $\mathbf{x}'$ must lie. The line $\mathbf{l}'$ is the epipolar line corresponding to $\mathbf{x}$, and this is illustrated in figure 5.1b. Algebraically the relation is

$$\mathbf{l}' = \mathrm{F}\mathbf{x}. \tag{5.2}$$

All epipolar lines $\mathbf{l}'$ meet at the point of intersection between the baseline and the second image plane, and this point is referred to as the epipole $\mathbf{e}'$ in the second image, see figure 5.1b. This epipole is the left null space of F, that is

$$\mathrm{F}^{\mathsf{T}}\mathbf{e}' = \mathbf{0}. \tag{5.3}$$

The above derivation is equally valid for a point $\mathbf{x}'$ in the second image, which determines an epipolar line $\mathbf{l}$ intersecting the epipole $\mathbf{e}$ in the first image. In this case $\mathbf{l} = \mathrm{F}^{\mathsf{T}}\mathbf{x}'$, and $\mathrm{F}\mathbf{e} = \mathbf{0}$ with the epipole $\mathbf{e}$ thus being the right null space of F. In figure 5.2 the epipolar lines for a set of corresponding points in a pair of images are illustrated.

Again consider (5.1), which is true for any pair of corresponding points $\mathbf{x}$ and $\mathbf{x}'$. As only the fundamental matrix and point correspondences are part of the equation, it is possible to compute F from point correspondences alone. At least seven point correspondences are needed for this computation, and several algorithms exist for this purpose. Robust estimation of the fundamental matrix F for a pair of input images is treated in section 5.3.

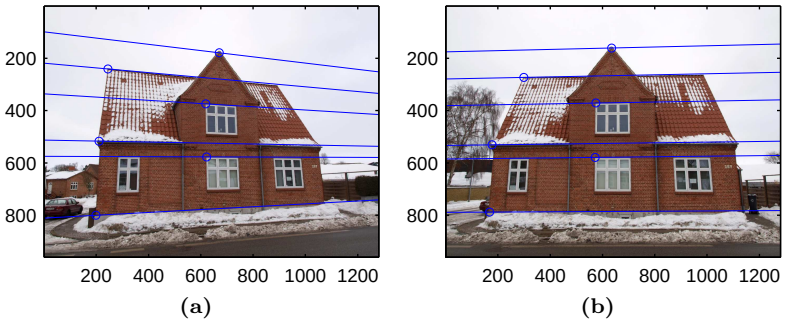


Figure 5.2: Illustration of 6 corresponding points and their epipolar lines in a pair of images. A point in the left image defines an epipolar line in the right image on which the corresponding point must lie, and vice versa.

5.2 Keypoint Matching

Given two sets of keypoints detected in one pair of input images, the purpose of keypoint matching is to establish an initial correspondence between the keypoints in the first image and the keypoints in the second image. An initial set of keypoint correspondences is necessary for estimating the fundamental matrix. Recall from section 4.3.2 that each keypoint is described by its location, scale, orientation, and descriptor. Only the descriptor is used for matching keypoints, and this is the primary reason that the descriptor must be distinctive as previously discussed.

The best candidate match for a keypoint in the first image is the nearest neighbour of the keypoint in the second image. The nearest neighbour is defined as the keypoint with minimum Euclidean distance between the descriptors [27]. With n and m being the number of keypoints detected in the first and second image respectively, let $\mathbf{d}_i,\ i = 1, \ldots, n$ be the 128 dimensional descriptor of keypoint i in the first image. Similarly let $\mathbf{d}'_j,\ j = 1, \ldots, m$ be the descriptor of keypoint j in the second image. Then the closest neighbour of keypoint i in the first image, is keypoint j in the second image for which

$$j = \arg\min_j ||\mathbf{d}_i - \mathbf{d}'_j||. \tag{5.4}$$

However, not all keypoints detected in the first image necessarily correspond to a keypoint in the second image and vice versa. Hence simply selecting the nearest neighbour as a match is insufficient, and a way to discard mismatches is needed. The method proposed in [27] is to compare the distance of the closest neighbour to the distance of the second-closest neighbour. Let $\mathbf{d}'_k$ be the descriptor of the second-closest keypoint in the second image. Then two keypoints are only considered a match if the following relation is true.

$$\frac{||\mathbf{d}_i - \mathbf{d}'_j||}{||\mathbf{d}_i - \mathbf{d}'_k||} < \tau \tag{5.5}$$

The value of the threshold τ recommended in [27] based on experiments is 0.8, where it discards 90% of false matches, while discarding less than 5% of correct matches. In this project, a threshold of $\tau = 0.6$ is used based on the discussion in the following section.

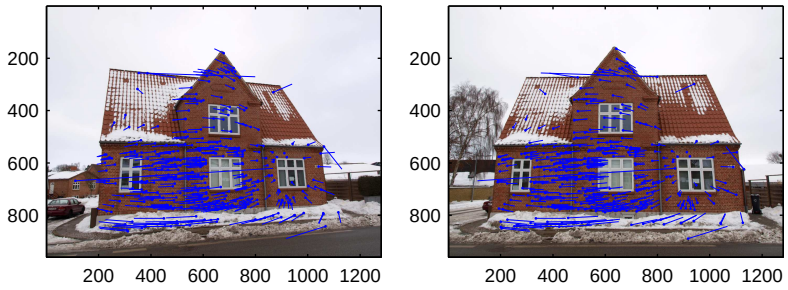


Figure 5.3: Keypoint matches between the two images shown in (a) and (b) using the threshold $\tau = 0.6$. Dots represent the location of keypoints in the image shown, and the other end of the lines indicate the location of the matched keypoints in the other image (only 25% of the 1527 matches are shown).

To identify an initial correspondence between keypoints in the two images, matching pairs of keypoints (i, j) may be found by applying (5.4) for all keypoints i in the first image to find corresponding keypoints j in the second image, and only keeping matches for which (5.5) is satisfied. Using this approach, however, it may be the case that two or more keypoints i match a single keypoint j in the second image. A one-to-one mapping is obtained by keeping only the best match in this case, i.e. the pair (i, j) for which $||\mathbf{d}_i - \mathbf{d}'_j||$ is minimum.

5.2.1 Evaluation

In figure 5.3 the result of applying the above method for keypoint matching is shown for a pair of images. For the images of buildings used in this project, a threshold of $\tau = 0.6$ has shown to perform better than 0.8, which is recommended for the general case in [27]. A significant amount of false matches are not discarded if the higher threshold is used. This may be due to the presence of repeated patterns, such as joints between bricks, often present on buildings. In figure 5.4 a comparison of keypoint matches obtained using the two thresholds is shown. Note the presence of several transverse or crossing lines, which represent false matches, in figure 5.4b.

Presence of many false matches may have negative impact on estimation of the fundamental matrix, which is explained in the following section. But as can be seen in figure 5.4a, even using the low threshold, some false matches remain. A small number of false matches is not a problem, however, because they are discarded as outliers when estimating the fundamental matrix.

5.3 Estimation of the Fundamental Matrix

In this section a method for robust estimation of the fundamental matrix F and identification of a robust correspondence between keypoints in the two images, i.e. a set of keypoint inliers that satisfy the epipolar constraint, is developed.

Several algorithms exist for estimating the fundamental matrix from a set of corresponding points in two images. The 7-point algorithm is a solution to this problem requiring the minimal number of points, but this algorithm

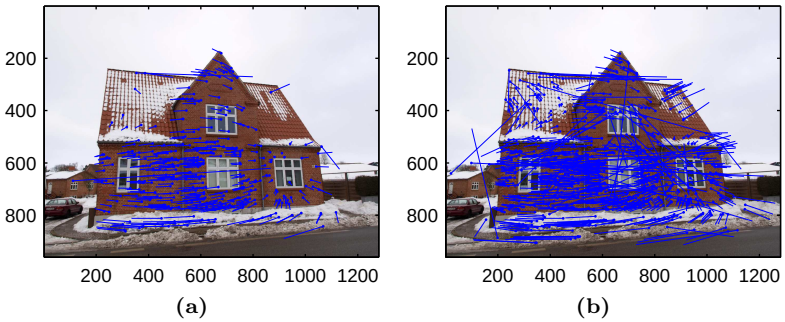


Figure 5.4: Keypoint matches between two images for different thresholds (only 25% of the matches are shown). Dots represent the location of keypoints in the image shown, and the other end of the lines indicate the location of the matched keypoints in the other image. a) Threshold $\tau = 0.6$ resulting in 1527 matches. b) Threshold $\tau = 0.8$ resulting in 3445 matches. Note the presence of several false matches here.

involves solving non-linear equations [34]. A more direct approach is to use the 8-point algorithm, which solves the problem using linear equations only. If used naively, this algorithm is susceptible to noise, but in [24] it has been shown that a normalized version of this algorithm performs very well. Therefore, in this project, the normalized 8-point algorithm is used for computing the fundamental matrix, and this algorithm is analyzed in the following section.

As discussed in section 5.2.1 the result of keypoint matching may contain outliers that do no satisfy the epipolar constraint. It is important that such outliers are not included when computing the fundamental matrix, because this would lead to a wrong relative pose of the cameras. Therefore to achieve robustness, RANdom SAmple Consensus (RANSAC) is employed to identify a robust set of keypoint inliers and compute the corresponding fundamental matrix in the same process. In section 5.3.2, the RANSAC algorithm itself is analyzed, and then in section 5.3.3 a robust method for estimation of the fundamental matrix using the normalized 8-point algorithm in combination with RANSAC is developed.

5.3.1 The Normalized 8-Point Algorithm

This section contains an analysis of the normalized 8-point algorithm, and it is based on [23] and [24]. First the basic 8-point algorithm is introduced, and then the normalization, which reduces sensitivity to noise, is explained. As suggested by the name, 8 corresponding points in the two images are needed for computing the fundamental matrix. Given 8 points the solution involves solving a set of linear equations. In the case where more than 8 points are known, the problem can be solved using linear least squares minimization. Computation of the fundamental matrix using this algorithm is simple, and with the normalized version accurate results can be obtained.

Linear Solution Recall from section 5.1 that the fundamental matrix F is defined by the relation $\mathbf{x}'^{\mathsf{T}}\mathrm{F}\mathbf{x} = 0$, where $\mathbf{x}$ and $\mathbf{x}'$ are corresponding points in the first and second image respectively. Given at least 8 corresponding points $\mathbf{x}_i$

and $\mathbf{x}'_i$, the unknown matrix F can be computed. Now letting $\mathbf{x} = [x\ y\ 1]^\mathsf{T}$ and $\mathbf{x}' = [x'\ y'\ 1]^\mathsf{T}$, each point correspondence gives rise to one linear equation in the unknown entries of F. This equation can be written in terms of the known point coordinates as

$$\begin{aligned} x'x\, f_{11} + x'y\, f_{12} + x'\, f_{13} + \\ y'x\, f_{21} + y'y\, f_{22} + y'\, f_{23} + \\ x\, f_{31} + y\, f_{32} + f_{33} = 0. \end{aligned} \tag{5.6}$$

By adding a row $[x'x, x'y, x', y'x, y'y, y', x, y, 1]$ to a matrix A for each point correspondence, a set of linear equations on the form

$$\mathrm{A}\mathbf{f} = \mathbf{0} \tag{5.7}$$

is obtained, where $\mathbf{f}$ is a 9-vector containing the entries of the matrix F in row-major order. As this is a homogeneous set of equations the vector $\mathbf{f}$, and hence F, can only be determined up to an unknown scale, and therefore the additional condition $||\mathbf{f}|| = 1$ is added.

The solution to this system can be found using Singular Value Decomposition (SVD), where A is factorized as $\mathrm{A} = \mathrm{UDV}^\mathsf{T}$. Then $\mathbf{f}$ is the column of V corresponding to the smallest singular value of A. The factorization can be carried out such that the diagonal entries of D are sorted in descending order, and this is assumed for all subsequent uses of SVD in the report. In this case the solution vector $\mathbf{f}$ is simply the last column of V. See appendix B for more details on solving homogeneous equations using this method.

Constraint Enforcement A property of the fundamental matrix F introduced in section 5.1 is that it has rank 2 and hence is singular. The matrix F found by solving the linear equations in (5.7) will in general not have rank 2, because of noise in the point coordinates. For a non-singular matrix F, the epipolar lines will not intersect in the epipoles of the two images, which is required for a valid fundamental matrix. Therefore it is necessary to enforce the constraint that F has rank 2.

This singularity constraint can be enforced by replacing F with the matrix F' which minimizes the Frobenius[1] norm $||\mathrm{F} - \mathrm{F}'||$ subject to $\det(\mathrm{F}') = 0$. This can be achieved by again using the SVD, and letting $\mathrm{F} = \mathrm{UDV}^\mathsf{T}$. With the diagonal matrix $\mathrm{D} = \mathrm{diag}(r, s, t)$ satisfying $r \geq s \geq t$, then $\mathrm{F}' = \mathrm{U}\,\mathrm{diag}(r, s, 0)\mathrm{V}^\mathsf{T}$ is the matrix that minimizes the Frobenius norm $||\mathrm{F} - \mathrm{F}'||$.

Normalization The above two steps, linear solution and constraint enforcement, constitute the basic 8-point algorithm. To improve the precision of the algorithm, these steps can be preceded by a normalization step and followed by a denormalization step, and this is the essence of the normalized 8-point algorithm.

The suggested normalization is to transform the point coordinates such that the centroid of the points is at the origin, and the RMS distance of the points from the origin is equal to $\sqrt{2}$. Transform the point coordinates according to $\hat{\mathbf{x}}_i = \mathrm{T}\mathbf{x}_i$, and $\hat{\mathbf{x}}'_i = \mathrm{T}'\mathbf{x}'_i$, where T and T' are normalizing transforms consisting

[1] The Frobenius norm of an $m \times n$ matrix A is defined as $||A|| = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}|a_{ij}|^2}$.

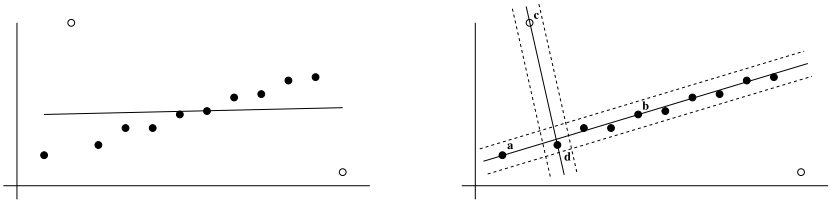


Figure 5.5: Robust line estimation [23]. Filled points are inliers, and open points are outliers. a) Least-squares orthogonal regression is severely affected by the outliers. b) Two lines resulting from random selection of two points during RANSAC. The dotted lines indicate the distance threshold. Support for the line **a** to **b** is 10, whereas for the line **c** to **d** it is 2.

of translation and scaling. Then a fundamental matrix $\hat{\mathrm{F}}'$ can be found for the transformed correspondences $\hat{\mathbf{x}}_i$ and $\hat{\mathbf{x}}'_i$ using the two steps above. To get the final estimated fundamental matrix F corresponding to the original points $\mathbf{x}_i$ and $\mathbf{x}'_i$, denormalize by setting $\mathrm{F} = \mathrm{T}'^{\mathsf{T}}\hat{\mathrm{F}}'\mathrm{T}$.

5.3.2 Random Sample Consensus (RANSAC)

This section contains an analysis of the generic RANSAC algorithm, which is an iterative method for estimating the parameters of a mathematical model from a set of observed data containing outliers. The following is based on [23] and [21], in which the algorithm was first introduced.

Methods for parameter estimation such as least squares optimize parameters to fit a model for all observed data, and these methods have no mechanism for rejecting outliers. It is assumed that there are always enough good values to smooth out any gross deviations in the observed data. But this assumption does not hold in many situations. For instance consider the set of 2D points in figure 5.5a, where the best fitting line according to least squares orthogonal regression is severely affected by the outliers. RANSAC is a well proven method for solving this problem in a robust manner. It is based on the assumption that the observed data consists of a set of inliers, which can be explained by the model for some set of parameters, and outliers which do not fit the model.

Employing RANSAC, the problem illustrated in figure 5.5a can be stated as follows: Given a set of 2D points, find the orthogonal regression line, subject to the condition that no valid points (inliers) deviate from this line by more than t units. This problem consists of two sub-problems: line fitting, and classification of points into inliers and outliers. The idea is very simple. By selecting two of the points randomly, a line passing through these points is defined. The points within the distance threshold t of this line define a consensus set, and the support for the line is measured as the number of points in the consensus set. The random selection of two points is repeated a number of times N, and the line with most support is deemed the robust fit. For instance the line from **c** to **d** in figure 5.5b has a support of 2, whereas the line from **a** to **b** has the maximum support of 10 and thus is the robust fit. When a robust fit has been determined, the consensus set defines the inliers, and the final line is fitted using the whole consensus set. Optionally the algorithm can be terminated if the

Objective

Robust fit of a model to a data set S which contains outliers.

Algorithm

1. Randomly select a sample of s data points from S and instantiate the model from the subset.
2. Determine the set of data points S_c which are within a distance threshold t of the model. The set S_c is the consensus set of the sample and defines the inliers of S.
3. If the size of S_c (the number of inliers) is greater than some threshold T, re-estimate the model using all the points in S_c and terminate.
4. If the size of S_c is less than T, select a new subset and repeat the above.
5. After N trials the largest consensus set S_c is selected, and the model is re-estimated using all the points in the subset S_c.

Algorithm 5.1: The RANSAC robust estimation algorithm as listed in [23]. A minimum of s data points are required to instantiate the free parameters of the model.

number of points in the consensus set for a selection of two points exceeds a threshold T.

In the example of figure 5.5 the fitted model is a line, and 2 points are sufficient to estimate the free parameters of the model. For other models the number of points necessary may vary, e.g. using the normalized 8-point algorithm a minimum of 8 points are required to estimate the fundamental matrix. The idea in RANSAC is in each iteration to fit the model using a random sample of minimum size, and then enlarge the consensus set with data that is consistent with the fitted model. The generic RANSAC algorithm is summarized in algorithm 5.1.

When applying RANSAC to a specific problem it is necessary to find suitable values for the parameters t, N, and T of the algorithm. The threshold t can be thought of as defining the minimum quality of inliers, and it depends on the error measurement used. The number of iterations N and the threshold T can both be determined from knowledge about the fraction of data consisting of outliers. Let ϵ be the probability that any selected data point is an outlier, then the number of samples N necessary to ensure with probability p that at least one sample contains only inliers can be calculated as

$$N = \frac{\log(1-p)}{\log\left(1-(1-\epsilon)^s\right)}, \tag{5.8}$$

where s is the size of the random samples. In the example of figure 5.5 the fraction of outliers is $\epsilon = 2/12 \approx 0.17$, and the sample size $s = 2$. Usually p is set to 0.99, and by inserting into (5.8) the number of iterations necessary in the example becomes $N = 4$. It is important to note that this is a theoretical minimum, and that a sample consisting of inliers only does not quarantee a model with good support. Therefore in practice the number of iterations necessary to ensure good performance may be significantly higher. For selection of the threshold T, a rule of thumb is to set it to the number of inliers believed to be in the data set, that is $T = (1-\epsilon)n$. In the example of figure 5.5, the algorithm can thus be terminated if the consensus set reaches a size of at least $T = 10$.

5.3.3 Robust Estimation

Having introduced the normalized 8-point algorithm and the general RANSAC algorithm in the previous sections, in this section a method for identifying a robust set of keypoint inliers and computing the corresponding fundamental matrix using these algorithms in combination is developed.

The overall approach is to use the RANSAC algorithm outlined in algorithm 5.1. The model to fit in this case is the epipolar constraint given in (5.1), and estimation of the model parameters, i.e. the fundamental matrix, is done using the normalized 8-point algorithm. Let $\mathbf{x}_i$ and $\mathbf{x}'_i$ denote the keypoint locations in the first and second image for the i^{th} pair of matching keypoints found during the keypoint matching step treated in section 5.2. The data set S, for which a robust model fit is sought, is then the set of corresponding keypoint locations. At least 8 corresponding points from this set are necessary for instantiating the model, but in this project random samples of size $s = 12$ are used based on the discussion in the following section.

For each iteration of the algorithm, a random sample of 12 point correspondences $\mathbf{x}_i$ and $\mathbf{x}'_i$ is selected from S, and a corresponding fundamental matrix F is estimated as described in section 5.3.1. Then the consensus set S_c is determined as the subset of point correspondences in S for which some error measure is below a threshold t. Here a first-order approximation to the geometric reprojection error of the points, referred to as the Sampson distance [23], is used. For corresponding points $\mathbf{x}_i$ and $\mathbf{x}'_i$ and fundamental matrix F, the Sampson distance $d_\perp$ can be computed from

$$d_\perp^2 = \frac{(\mathbf{x}_i'^{\mathsf{T}}\mathrm{F}\mathbf{x}_i)^2}{(\mathrm{F}\mathbf{x}_i)_1^2 + (\mathrm{F}\mathbf{x}_i)_2^2 + (\mathrm{F}^{\mathsf{T}}\mathbf{x}'_i)_1^2 + (\mathrm{F}^{\mathsf{T}}\mathbf{x}'_i)_2^2}, \tag{5.9}$$

where $(\mathrm{F}\mathbf{x}_i)_k^2$ represents the square of the k^{th} entry of the vector $\mathrm{F}\mathbf{x}_i$. Thus point correspondences for which $d_\perp < t$ are included in the consensus set. The value of t is discussed in the following section.

For simplicity, a fixed number of iterations is used in this project, and the value of N is discussed in the following section. After the N iterations are completed, the largest consensus set S_c is selected, and the final fundamental matrix F is estimated from this set. To check whether an acceptable match between the pair of input images has been found, the number of keypoint inliers is compared to a threshold M. If the number of inliers is less than M, this pair of input images is not an acceptable match, and further processing is not performed for this pair. In this project the value $M = 100$ is selected to limit processing in later steps of reconstruction, and to prevent bad matching pairs from affecting performance during clustering.

For efficient implementation the normalization step described in section 5.3.1 is performed before applying RANSAC as described above, and therefore the data set S actually consists of the normalized point correspondences $\hat{\mathbf{x}}_i$ and $\hat{\mathbf{x}}'_i$, cf. section 5.3.1. A summary of the complete method developed in this section is given in algorithm 5.2.

5.3.4 Evaluation

In figure 5.6 the result of applying the above robust method to the same pair of input images as in figure 5.3 is shown. Comparing the figures, it is seen

Objective

Given a set of matching keypoint locations $\mathbf{x}_i$ and $\mathbf{x}'_i$, determine a robust set of keypoint inliers and the corresponding fundamental matrix F.

Algorithm

1. *Normalization:* Let S be the set of transformed point correspondences $\hat{\mathbf{x}}_i = \mathrm{T}\mathbf{x}_i$ and $\hat{\mathbf{x}}'_i = \mathrm{T}'\mathbf{x}'_i$, cf. section 5.3.1.
2. *Robust Estimation:* Repeat for N iterations:
   (a) Select a random sample of 12 correspondences from S and estimate the corresponding fundamental matrix $\hat{\mathrm{F}}'$, cf. section 5.3.1.
   (b) For all correspondences in S compute the Sampson distance $d_\perp$ given $\hat{\mathrm{F}}'$ using (5.9).
   (c) Determine the consensus set S_c consisting of the correspondences in S for which $d_\perp < t$.

   Select the largest found consensus set S_c as the set of keypoint inliers, and re-estimate $\hat{\mathrm{F}}'$ from all correspondences in S_c.
3. *Denormalization:* Set the final fundamental matrix $\mathrm{F} = \mathrm{T}'^{\mathsf{T}}\hat{\mathrm{F}}'\mathrm{T}$.

An acceptable match between the two input images has been found if the number of keypoint inliers is at least M.

Algorithm 5.2: Robust estimation of the fundamental matrix and identification of keypoint inliers using RANSAC and the normalized 8-point algorithm.

that most of the false matches are discarded as outliers. This image pair has 1527 keypoint matches, and 1508 of these are now identified as keypoint inliers. There may be a few remaining false matches that by chance satisfy the epipolar constraint, but their influence on the results is negligible.

The following is a discussion of the parameters selected for robust estimation of the fundamental matrix. Although 8 correspondences are sufficient for computing a fundamental matrix, a sample size $s = 12$ is chosen. The reasoning is that an initial least-squares fit from 12 true inliers increases the probability of other true inliers being within the distance threshold, and thus a larger consensus set can be obtained.

The distance threshold t is set to the value 0.01, and this is the maximum Sampson distance allowed for inliers. Note however that this threshold is applied to normalized image coordinates, cf. algorithm 5.2, and therefore the value does not directly correspond to a pixel distance. In the images of figure 5.6 an average scale of 0.0065 was applied during normalization, and thus $t = 0.01$ corresponds to a maximum reprojection error of approximately 1.5 pixels. For the inliers of this image pair the actual RMS reprojection error is 0.16 pixels, which is very good compared to the distance threshold. For all matching image pairs in the data sets the RMS reprojection error of inliers is less than 1 pixel.

The number of iterations is fixed to the value $N = 500$, even though less iterations are sufficient for most image pairs in the data sets. Here this is compared to the theoretical number given by (5.8) in section 5.3.2. First the ratio of inliers to matches, which gives an approximate probability of a correspondence being an inlier, is determined, and from this a suitable ϵ can be chosen. For all matching image pairs in the data sets, the average number of keypoint matches is 455, and on average there are 441 keypoint inliers. This gives a ratio of 0.97

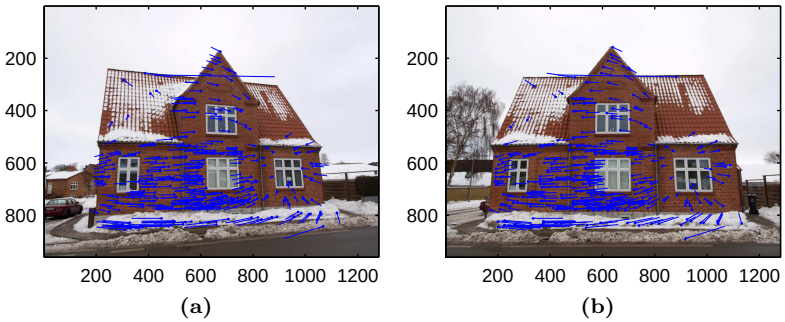


Figure 5.6: Keypoint inliers for the two images shown in (a) and (b). Dots represent the location of keypoints in the image shown, and the other end of the lines indicate the location of the corresponding keypoints in the other image (only 25% of the 1508 inliers are shown).

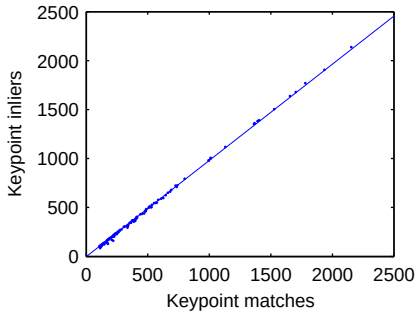


Figure 5.7: Number of keypoint matches and inliers for all matching pairs of input images in the data sets used in this project. The ratio of inliers to matches is computed as 0.98 using linear regression.

inliers per match. Employing linear regression as illustrated in figure 5.7 gives a ratio of 0.98 inliers per match. As can be seen from this figure, the number of inliers is very close to being a linear function of matches, and this testifies that the quality of the SIFT keypoints is high. From the above a conservative guess is $\epsilon = 0.05$, and inserting this into (5.8) in theory 6 iterations are sufficient. In other words, using 500 iterations ensures that around 83 random samples contain only inliers, and of these samples the one with the highest number of inliers is used for estimating the fundamental matrix.

5.4 Recovery of Relative Pose

From the estimated fundamental matrix it is possible to recover the relative pose of the two cameras. This involves computing a matrix referred to as the essential matrix, and this is treated in the following section. Then from the essential matrix, the relative pose between the cameras can be recovered, and this process is explained in section 5.4.2.

5.4.1 Computing the Essential Matrix

The first step of recovering the relative pose of the two cameras is to compute the essential matrix. The essential matrix E is a 3×3 matrix similar to the fundamental matrix F, but it applies to cameras with known intrinsic calibration. Therefore this step depends on the intrinsic calibration of the images obtained in the preprocessing step as described in section 4.2.

Recall from section 4.1 that a camera matrix is defined as $\mathrm{P} = \mathrm{K}[\mathrm{R}\ \mathbf{t}]$, and let $\mathbf{x} = \mathrm{P}\mathbf{X}$ be a point in the image. Knowing the camera calibration matrix K from preprocessing, its inverse can be applied to the point $\mathbf{x}$ to obtain the point $\hat{\mathbf{x}} = \mathrm{K}^{-1}\mathbf{x}$, which is expressed in normalized coordinates[2]. This removes the effect of the intrinsic parameters as can be seen from $\hat{\mathbf{x}} = [\mathrm{R}\ \mathbf{t}]\mathbf{X}$. A camera matrix of the form $\mathrm{K}^{-1}\mathrm{P} = [\mathrm{R}\ \mathbf{t}]$ is referred to as a normalized camera matrix [23].

In the following two normalized camera matrices $\mathrm{P} = [\mathrm{I}\ \mathbf{0}]$ and $\mathrm{P}' = [\mathrm{R}\ \mathbf{t}]$ corresponding to the first and second image respectively in a pair of input images are considered. The corresponding camera calibration matrices K and K′ are known from preprocessing, and thus for corresponding points $\mathbf{x}$ and $\mathbf{x}'$ the normalized points $\hat{\mathbf{x}} = \mathrm{K}^{-1}\mathbf{x}$ and $\hat{\mathbf{x}}' = \mathrm{K}'^{-1}\mathbf{x}'$ can be computed. The essential matrix E is now defined by

$$\hat{\mathbf{x}}'^{\mathsf{T}}\mathrm{E}\hat{\mathbf{x}} = 0. \tag{5.10}$$

Comparing this to (5.1) it is seen that the essential matrix is the fundamental matrix for normalized cameras. From above it can be derived that

$$\mathrm{E} = \mathrm{K}'^{\mathsf{T}}\mathrm{F}\mathrm{K}. \tag{5.11}$$

The fact that the cameras are normalized imposes additional constraints on the essential matrix. For an essential matrix to be valid, it must have two singular values that are equal, and one that is zero [23]. For an essential matrix E computed using (5.11), this constraint can be enforced in a way similar to the singularity constraint for the fundamental matrix as explained in section 5.3.1. Employing SVD let $\mathrm{E} = \mathrm{U}\mathrm{D}\mathrm{V}^{\mathsf{T}}$, with $\mathrm{D} = \mathrm{diag}(r, s, t)$ satisfying $r \geq s \geq t$. Then the constraint can be enforced by computing $u = (r + s)/2$ and setting the final essential matrix $\mathrm{E} = \mathrm{U}\,\mathrm{diag}(u, u, 0)\mathrm{V}^{\mathsf{T}}$.

5.4.2 Relative Pose from the Essential Matrix

From the essential matrix E computed using the above method, the relative pose between the two cameras can be recovered up to an unknown scale. With the normalized camera matrices $\mathrm{P} = [\mathrm{I}\ \mathbf{0}]$ and $\mathrm{P}' = [\mathrm{R}\ \mathbf{t}]$, the relative pose is defined by the rotation matrix R and the translation vector $\mathbf{t}$. The following is based on [23] and [31].

Disregarding the unknown scale, the SVD of the essential matrix can be written as $\mathrm{E} = \mathrm{U}\,\mathrm{diag}(1, 1, 0)\mathrm{V}^{\mathsf{T}}$, with U and V chosen such that $\det(\mathrm{U}) > 0$ and $\det(\mathrm{V}) > 0$. Then the translation vector $\mathbf{t}$ equals either $+\mathbf{u}_3$ or $-\mathbf{u}_3$, and thus $||\mathbf{t}|| = 1$. By introducing the matrix

$$\mathrm{W} = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{5.12}$$

[2]Note that this normalization is *not* the same as in the normalized 8-point algorithm.

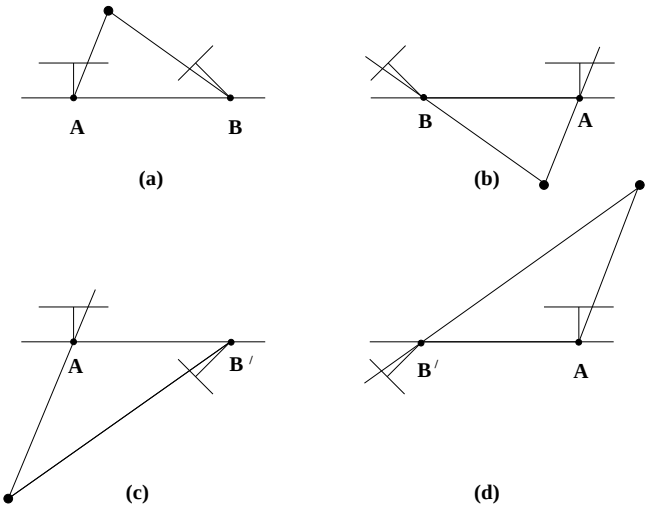


Figure 5.8: Geometrical illustration of the four possible camera configurations [23]. Between left and right the baseline is reversed. Between top and bottom rows the camera **B** rotates 180° about the baseline. Only in configuration (a) the triangulated point is in front of both cameras.

the rotation matrix R is equal to either $\mathrm{R}_a = \mathrm{U}\mathrm{W}\mathrm{V}^\mathsf{T}$ or $\mathrm{R}_b = \mathrm{U}\mathrm{W}^\mathsf{T}\mathrm{V}^\mathsf{T}$. This leads to 4 possible solutions for P' given by

$$\begin{aligned} \mathrm{P}'_A &= [\mathrm{R}_a \ \mathbf{u}_3], & \mathrm{P}'_B &= [\mathrm{R}_a \ -\mathbf{u}_3], \\ \mathrm{P}'_C &= [\mathrm{R}_b \ \mathbf{u}_3], \text{ and} & \mathrm{P}'_D &= [\mathrm{R}_b \ -\mathbf{u}_3]. \end{aligned} \tag{5.13}$$

Only one of these correspond to the true configuration, and to determine which one, the chirality constraint is imposed. By triangulating[3] a space point $\mathbf{X}$ from two corresponding points in the images using the camera matrices P and P'_A the ambiguity can be resolved. The four solutions are illustrated in figure 5.8. As can be seen from the figure, the triangulated point is only in front of both cameras in one of the configurations.

Let $\mathbf{X} = [X_1 \ X_2 \ X_3 \ X_4]^\mathsf{T}$ be the homogeneous 4-vector representing the triangulated point, and compute $c_1 = X_3 X_4$ and $c_2 = (\mathrm{P}'_A\mathbf{X})_3 X_4$. Then if $c_1 > 0$ the point is in front of the first camera, and if $c_2 > 0$ the point is in front of the second camera. Hence if both $c_1 > 0$ and $c_2 > 0$ the true configuration is $\mathrm{P}' = \mathrm{P}'_A$. If $c_1 < 0$ and $c_2 < 0$ the baseline is reversed and $\mathrm{P}' = \mathrm{P}'_B$. On the other hand if $c_1 c_2 < 0$ it is necessary to check the twisted pair, so compute $c_3 = (\mathrm{P}'_C\mathbf{X})_3 X_4$. Then if $c_1 > 0$ and $c_3 > 0$ the true configuration is $\mathrm{P}' = \mathrm{P}'_C$. Finally, if $c_1 < 0$ and $c_3 < 0$ the baseline is reversed and $\mathrm{P}' = \mathrm{P}'_D$.

5.4.3 Evaluation

One point correspondence is sufficient to determine the correct camera configuration using the chirality constraint. Such point correspondence is conveniently obtained from the locations of a keypoint inlier in the two images.

[3] Triangulation is treated in section 6.1.1.

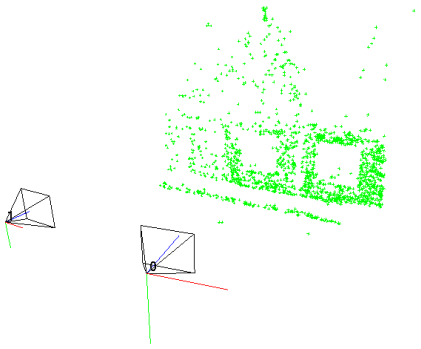


Figure 5.9: Example of the relative pose recovered for a pair of input images. In addition to the cameras, triangulated keypoint inliers are shown as green dots.

It was discovered, however, that in practice it is not enough to use a single keypoint correspondence. As discussed in section 5.3.4, some false matches may by chance satisfy the epipolar constraint and are thus wrongly identified as keypoint inliers. When triangulating these false matches, the resulting point may end up behind the cameras, and therefore the camera configuration is checked for all keypoint inliers. The most frequent configuration is deemed correct, and any mismatching points are discarded as outliers. For instance in the image pair shown in figure 5.6, four keypoint inliers were discarded because they did not match the configuration of the majority of correspondences.

In figure 5.9 an example of recovering the relative pose of two cameras using this method is shown. In addition to the two cameras, also the triangulated keypoint inliers are shown.

5.5 Conclusion

In this chapter, the matching step of the proposed reconstruction method has been treated in detail, and a robust method for estimating the relative pose in all matching pairs of input images has been developed. Epipolar geometry and the fundamental matrix were introduced in section 5.1, and in section 5.2 a method for establishing an initial correspondence between keypoints in a pair of input images was developed. Then in section 5.3, a robust method for identifying a set of keypoint inliers and estimating the fundamental matrix, based on a combination of RANSAC and the normalized 8-point algorithm, was developed. Finally in section 5.4 a method for extracting the relative pose of the two cameras in a pair of input images was described. A lot of ground has been covered in this chapter, and this provides the basis for recovering structure and motion in the clustering step, which is the topic of the following chapter.

Chapter 6

Clustering

In this chapter, the clustering step of the proposed reconstruction method is documented. The purpose is to recover structure and motion for all input images. That is the positions of keypoints in space and the extrinsic parameters for all input images are estimated, yielding a result as illustrated in figure 3.2. In this context a cluster is a collection of cameras with estimated extrinsic parameters and a point cloud of estimated keypoint positions. The developed method builds on the the recovered relative pose and the set of keypoint inliers identified for each matching pair of input images in the previous step.

Traditionally methods for recovering structure and motion are based on finding a global initial guess for the extrinsic parameters of all cameras and all point positions, and then performing a final optimization using bundle adjustment. For bundle adjustment to succeed, a good initial guess of the global structure and motion is necessary [37]. In this project a bottom up approach, in which bundle adjustment is applied throughout the clustering process, has been developed. This minimizes the risk that a bad initial guess prevents successful recovery of structure and motion. A convenient consequence of applying bundle adjustment this way is that simpler and less accurate methods for initial estimation can be employed without affecting the end result.

The overall approach of the developed clustering method is to initialize the cluster from the best matching image pair, and then add images to the cluster iteratively until all input images are included. When adding an image to the cluster, the relative pose recovered in the matching step is used for computing an initial estimate of the extrinsic camera parameters, and keypoint inliers are triangulated to obtain an estimate of keypoint positions. Each time an image has been added to the cluster, the recovered structure and motion is optimized by applying bundle adjustment.

In the following sections the details of the developed clustering method are documented. In section 6.1 the process of adding the first pair of images to the cluster is treated. This corresponds to reconstruction from stereo images and includes triangulation of keypoints. Section 6.2 then covers addition of a third image to the cluster. Having introduced these two basic steps, the developed clustering algorithm itself is treated in section 6.3. For each iteration of clustering an optimization step, consisting of bundle adjustment and various robustness measures, is applied. Section 6.4 provides an overview of bundle adjustment, and finally in section 6.5 the optimization step is treated.

6.1 The First Pair of Images

In this section, the first step of clustering is treated, namely initializing the cluster from the best matching image pair found during the matching step. This pair is simply the pair of input images having the largest set of keypoint inliers. Let m be the number of input images. Then the best matching pair consists of images (a, b), for some a and b satisfying $1 \leq a < b \leq m$.

Now let $\mathrm{P}_{(a,b)}$ denote the relative pose recovered for this pair. This is a normalized camera matrix composed of a rotation and a translation as discussed in section 5.4.2. This directly leads to an initial estimate of the extrinsic parameters for the cameras corresponding to images a and b. The cluster is simply initialized using the two camera matrices $\mathrm{P}_a = [\mathrm{I}\ \mathbf{0}]$ and $\mathrm{P}_b = \mathrm{P}_{(a,b)}$.

The relative pose can only be recovered up to an unknown scale, and this limitation also applies to the structure and motion recovered during clustering. By selecting the identity matrix for P_a, the world coordinate system is defined by the camera of image a, and thus structure and motion is only recovered up to an arbitrary similarity transform.

6.1.1 Triangulation of Keypoints

With two cameras in the cluster, the positions of the keypoint inliers of the image pair are estimated using triangulation. As discussed in the chapter introduction, bundle adjustment is applied in each iteration of clustering, and therefore a simple method can be used for triangulation. Here the linear triangulation method described in [23] is used. In the following the generic method is described, and then it is applied to the problem of triangulating the keypoint inliers of the image pair.

Given measured corresponding points $\mathbf{x}$ and $\mathbf{x}'$ in a pair of images with camera matrices P and P′, the objective of triangulation is to estimate the space point $\mathbf{X}$. Under ideal circumstances $\mathbf{x} = \mathrm{P}\mathbf{X}$ and $\mathbf{x}' = \mathrm{P}'\mathbf{X}$, but due to noise in the image points an exact solution can not be found, see figure 6.1. As these equations involve homogeneous vectors, the relation is rather $\mathbf{x} \propto \mathrm{P}\mathbf{X}$, and likewise for the second image. That is, vectors $\mathbf{x}$ and $\mathrm{P}\mathbf{X}$ are not necessarily equal but have the same direction and may differ in magnitude by a non-zero scale factor. Employing the Direct Linear Transformation (DLT) algorithm [23], the triangulation problem can be expressed in the form $\mathrm{A}\mathbf{X} = \mathbf{0}$, which is an equation linear in $\mathbf{X}$.

The unknown homogeneous scale factor can be eliminated by utilizing that two non-zero vectors $\mathbf{a}$ and $\mathbf{b}$ are parallel if and only if $\mathbf{a} \times \mathbf{b} = \mathbf{0}$. Then for the first image $\mathbf{x} \times (\mathrm{P}\mathbf{X}) = \mathbf{0}$. Introducing the notation $\mathbf{a} \times \mathbf{b} = [\mathbf{a}]_\times \mathbf{b}$, where $[\mathbf{a}]_\times$ is the skew-symmetric matrix

$$[\mathbf{a}]_\times = \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix}, \tag{6.1}$$

this can be written $[\mathbf{x}]_\times(\mathrm{P}\mathbf{X}) = \mathbf{0}$. Letting $\mathbf{x} = [x\ y\ 1]^\mathsf{T}$ and writing out gives

$$\begin{bmatrix} 0 & -1 & y \\ 1 & 0 & -x \\ -y & x & 0 \end{bmatrix} \begin{bmatrix} \mathbf{p}^{1\mathsf{T}}\mathbf{X} \\ \mathbf{p}^{2\mathsf{T}}\mathbf{X} \\ \mathbf{p}^{3\mathsf{T}}\mathbf{X} \end{bmatrix} = \mathbf{0}, \tag{6.2}$$

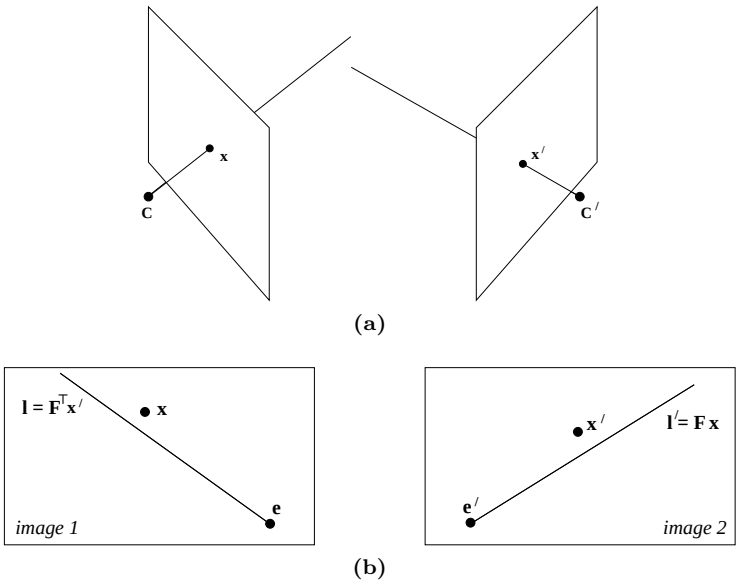


Figure 6.1: The effect of imperfectly measured points [23]. a) The rays back-projected from measured points $\mathbf{x}$ and $\mathbf{x}'$ are skew in general. b) The measured points do not satisfy the epipolar constraint. The epipolar line $\mathbf{l}$ is the image of the ray through $\mathbf{x}'$, and $\mathbf{l}'$ is the image of the ray through $\mathbf{x}$. Since the rays do not intersect, $\mathbf{x}$ does not lie on $\mathbf{l}$, and $\mathbf{x}'$ does not lie on $\mathbf{l}'$.

where $\mathbf{p}^{i\mathsf{T}}$ are the rows of P. This leads to three equations

$$x(\mathbf{p}^{3\mathsf{T}}\mathbf{X}) - (\mathbf{p}^{1\mathsf{T}}\mathbf{X}) = 0 \tag{6.3}$$

$$y(\mathbf{p}^{3\mathsf{T}}\mathbf{X}) - (\mathbf{p}^{2\mathsf{T}}\mathbf{X}) = 0 \tag{6.4}$$

$$x(\mathbf{p}^{2\mathsf{T}}\mathbf{X}) - y(\mathbf{p}^{1\mathsf{T}}\mathbf{X}) = 0 \tag{6.5}$$

of which two are linearly independent.

By including two equations for each image, a set of four equations with four unknowns is obtained in the form $\mathrm{A}\mathbf{X} = \mathbf{0}$, with A being the 4×4 matrix

$$\mathrm{A} = \begin{bmatrix} x\mathbf{p}^{3\mathsf{T}} - \mathbf{p}^{1\mathsf{T}} \\ y\mathbf{p}^{3\mathsf{T}} - \mathbf{p}^{2\mathsf{T}} \\ x'\mathbf{p}^{3\mathsf{T}} - \mathbf{p}'^{1\mathsf{T}} \\ y'\mathbf{p}^{3\mathsf{T}} - \mathbf{p}'^{2\mathsf{T}} \end{bmatrix}. \tag{6.6}$$

The solution is only determined up to scale, so the additional constraint $||\mathbf{X}|| = 1$ is added. This system can be solved using the SVD, as previously discussed, by letting $\mathrm{A} = \mathrm{UDV}^{\mathsf{T}}$. Then the solution is the last column of V, i.e. the estimated position of the space point $\mathbf{X} = \mathbf{v}_4$.

Returning to the problem of triangulating the keypoint inliers for the image pair (a, b) in the cluster, the goal is to obtain a set of space points $\mathbf{X}_i$, which are triangulated from the locations of the keypoint inliers. Denote by $\mathbf{x}_{i,a}$ and $\mathbf{x}_{i,b}$ the locations expressed in normalized coordinates of keypoint i in image a and b respectively. To obtain normalized coordinates, the inverse camera calibration

matrices K_a^{-1} and K_b^{-1} corresponding to the images are applied to the keypoint locations. The space points $\mathbf{X}_i$ are then triangulated using the above method for corresponding points $\mathbf{x}_{i,a}$ and $\mathbf{x}_{i,b}$, and camera matrices P_a and P_b.

The triangulated points are added to the cluster, which now consists of two cameras defined by the camera matrices $\mathrm{P}_j,\ j \in \{a, b\}$, and the set of points $\mathbf{X}_i$ triangulated from the keypoint inliers of the image pair. The final step of cluster initialization is to perform optimization of the recovered structure and motion as explained in section 6.5. The method for cluster initialization covered here is evaluated in section 6.2.4 below.

6.2 Adding a Third Image

With the cluster consisting of two cameras and points triangulated from the keypoint inliers of the corresponding image pair, the next task of clustering is to augment the cluster with another image. The image selected for addition to the cluster is the image c satisfying $1 \leq c \leq m$ and $c \notin \{a, b\}$, which has most keypoint inliers with any of the images already in the cluster. That is, image c is part of a matching image pair that also contains either image a or b and thus is a neighbour of the cluster. In the following assume that this image pair is (b, c) where $b < c$. Augmenting the cluster consists of three main steps, namely obtaining an initial pose estimate with unknown scale for the new camera, recovering the relative scale, and triangulating new keypoint inliers. These steps are explained in the following sections.

6.2.1 Initial Pose Estimate

From matching the estimated relative pose $\mathrm{P}_{(b,c)}$ between image b and c is known. An initial pose estimate for the camera corresponding to image c can be obtained up to scale by combining this information with the current state of the cluster. The extrinsic parameters for the camera corresponding to image b are defined by P_b, which is part of the cluster. With $\mathrm{P}_{(b,c)} = [\mathrm{R}_{(b,c)}\ \mathbf{t}_{(b,c)}]$ and $\mathrm{P}_b = [\mathrm{R}_b\ \mathbf{t}_b]$, the initial estimate is obtained by computing

$$\begin{bmatrix} \mathrm{R}_c & \hat{\mathbf{t}}_c \\ & 1 \end{bmatrix} = \begin{bmatrix} \mathrm{R}_{(b,c)} & \mathbf{t}_{(b,c)} \\ & 1 \end{bmatrix} \begin{bmatrix} \mathrm{R}_b & \mathbf{t}_b \\ & 1 \end{bmatrix} \tag{6.7}$$

and letting $\hat{\mathrm{P}}_c = [\mathrm{R}_c\ \hat{\mathbf{t}}_c]$, where the hat indicates the unknown relative scale.

In the case where $c < b$, the relative pose recovered during matching would instead be $\mathrm{P}_{(c,b)}$. This is the inverse Euclidean transformation of $\mathrm{P}_{(b,c)}$, so it is necessary to first compute

$$\mathrm{P}_{(b,c)} = \begin{bmatrix} \mathrm{R}_{(c,b)}^\mathsf{T} & -\mathrm{R}_{(c,b)}^\mathsf{T}\mathbf{t}_{(c,b)} \end{bmatrix}, \tag{6.8}$$

and then the initial pose estimate $\hat{\mathrm{P}}_c$ can be found using (6.7).

6.2.2 Recovering the Relative Scale

The relative poses recovered during matching all have a unit translation vector, and therefore the distance between the cameras defined by P_a and P_b is 1. This

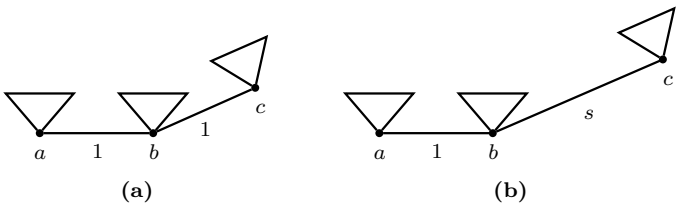


Figure 6.2: Illustration of the relative scale of distances between cameras a, b, and c in the cluster. a) The initial pose estimate for camera c with unknown scale. b) The pose estimate for camera c after the relative scale has been recovered.

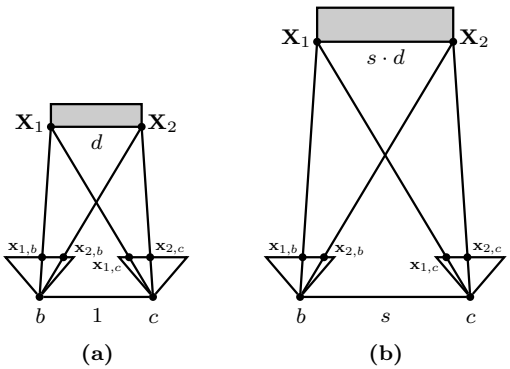


Figure 6.3: The true distance between two cameras can not be determined given point correspondences alone. Two space points $\mathbf{X}_1$ and $\mathbf{X}_2$ are observed by cameras b and c. The focal length and image plane dimensions are the same in both figure (a) and (b), and as can be seen the scale s has no influence on the position of the projections $\mathbf{x}_{1,b}$, $\mathbf{x}_{2,b}$, $\mathbf{x}_{1,c}$, and $\mathbf{x}_{2,c}$.

is also the case for the cameras defined by P_b and $\hat{\mathrm{P}}_c$, but the true distances between the cameras in image pairs (a, b) and (b, c) are unlikely to be identical. Therefore it is necessary to recover the relative scale by computing a scale factor s, which puts camera c in the correct position relative to the cluster, see figure 6.2. The problem arises because the true distance between two cameras can not be determined from a set of corresponding points in the images as illustrated in figure 6.3. Fortunately it is possible to recover the relative scale if at least one point is observed in all three images [37].

Suppose that a keypoint inlier in image pair (a, b) and a keypoint inlier in image pair (b, c) share a common keypoint in image b, then this keypoint is observed in all three images. Let $\mathbf{X}_i$ be the position of this keypoint triangulated using the keypoint locations $\mathbf{x}_{i,a}$ and $\mathbf{x}_{i,b}$ and camera matrices P_a and P_b as explained in section 6.1.1. Another space point $\hat{\mathbf{X}}_i$ can now be triangulated using the keypoint locations $\mathbf{x}_{i,b}$ and $\mathbf{x}_{i,c}$ in image pair (b, c) and the camera matrices P_b and $\hat{\mathrm{P}}_c$. It is known that $\mathbf{X}_i$ and $\hat{\mathbf{X}}_i$ represent the same point in the scene, but due to the unknown scale the triangulated position of these points may differ as illustrated in figure 6.4a.

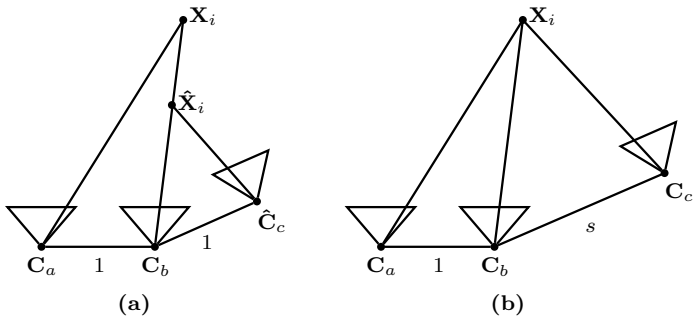


Figure 6.4: Recovering the relative scale using two space points $\mathbf{X}_i$ and $\hat{\mathbf{X}}_i$ triangulated using camera matrices $(\mathrm{P}_a, \mathrm{P}_b)$ and $(\mathrm{P}_b, \hat{\mathrm{P}}_c)$ respectively. The triangle formed by $\hat{\mathbf{X}}_i$, $\mathbf{C}_b$ and $\hat{\mathbf{C}}_c$ in figure (a) is similar to the triangle formed by $\mathbf{X}_i$, $\mathbf{C}_b$, and $\mathbf{C}_c$ in figure (b), and thus the relative scale s can be recovered.

The figure hints how the relative scale may be recovered. The triangle formed by $\hat{\mathbf{X}}_i$, and the camera centres $\mathbf{C}_b$ and $\hat{\mathbf{C}}_c$ is similar to the triangle in figure 6.4b formed by $\mathbf{X}_i$, $\mathbf{C}_b$, and $\mathbf{C}_c$. Therefore the relative scale can be computed as

$$s = \frac{||\mathbf{X}_i - \mathbf{C}_b||}{||\hat{\mathbf{X}}_i - \mathbf{C}_b||}, \tag{6.9}$$

where the camera centres $\mathbf{C}_b$ and $\hat{\mathbf{C}}_c$ are given by

$$\mathbf{C}_b = -\mathrm{R}_b^\mathsf{T}\mathbf{t}_b, \text{ and} \tag{6.10}$$

$$\hat{\mathbf{C}}_c = -\mathrm{R}_c^\mathsf{T}\hat{\mathbf{t}}_c. \tag{6.11}$$

The pose estimate for camera c with correct relative scale can now be computed by first obtaining the corrected camera position as

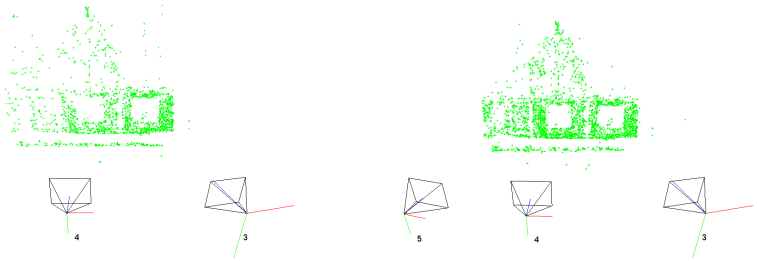
$$\mathbf{C}_c = \mathbf{C}_b + s \cdot (\hat{\mathbf{C}}_c - \mathbf{C}_b), \tag{6.12}$$

and then computing $\mathbf{t}_c = -\mathrm{R}_c\mathbf{C}_c$. Finally the corrected pose estimate becomes $\mathrm{P}_c = [\mathrm{R}_c \ \mathbf{t}_c]$.

Typically there is more than one keypoint that is observed in all three images, and a better estimate of the relative scale s can be obtained by utilizing all common keypoints. A simple average of relative scales computed using the above method for all common keypoints suffices, because gross outliers have been discarded in previous steps and bundle adjustment is applied afterwards. When the extrinsic parameters for camera c have been recovered, the camera matrix P_c is added to the cluster.

6.2.3 Triangulation of New Keypoints

When recovering the relative scale as described in the previous section, a set of keypoints that are observed in all three images is identified. The space points $\mathbf{X}_i$ corresponding to these keypoints are already triangulated using the projections $\mathbf{x}_{i,a}$ and $\mathbf{x}_{i,b}$ from image pair (a, b) as explained in section 6.1.1, and they are part of the cluster. However, the purpose of bundle adjustment is to optimize structure and motion using observations from all images, and therefore



(a) **(b)**

Figure 6.5: Example of structure and motion recovered during the first two steps of clustering. a) After initialization, the cluster consists of two cameras and 1771 keypoints triangulated from the corresponding image pair. b) Adding another image results in a third camera and additional triangulated keypoints, leading to a total of 2837 keypoints.

the common keypoints $\mathbf{X}_i$ are updated with the additional observations $\mathbf{x}_{i,c}$ in image c. I.e. the cluster now contains a set of points $\mathbf{X}_i$ which are observed only in image pair (a, b), and another set of points $\mathbf{X}_i$ which are observed in both image pairs (a, b) and (b, c).

When adding the third image, new keypoints that are observed in image pair (b, c) but not in (a, b) may be present. These keypoints should also be triangulated and added to the cluster. Therefore new space points $\mathbf{X}_i$ representing these keypoints are triangulated using corresponding points $\mathbf{x}_{i,b}$ and $\mathbf{x}_{i,c}$ and camera matrices P_b and P_c.

In summary, after adding image c the cluster contains three cameras defined by the camera matrices P_j, $j \in \{a, b, c\}$, and the set of points $\mathbf{X}_i$ with corresponding projections $\mathbf{x}_{i,j}$ in two or three of the images a, b, and c. As in cluster initialization, the final step of augmenting the cluster is to perform optimization as explained in section 6.5. The process outlined here is the general method for adding an image to the cluster.

6.2.4 Evaluation

In this section examples of results obtained using the two basic steps of clustering treated above, namely initialization of the cluster and augmenting the cluster with a third image, are shown. In figure 6.5a an example of structure and motion recovered using the method for cluster initialization covered in section 6.1 is shown. In this example the best matching image pair is (3,4) which has 1771 keypoint inliers.

The result of adding a third image to the cluster is shown in figure 6.5b. In this example, the image pair with most keypoint inliers that also contains either image 3 or 4 is image pair $(4, 5)$ with 1504 inliers. An initial pose estimate for camera 5 is computed, and a set of 438 keypoints that are observed in all three images is identified. Using the common keypoints, the relative scale is estimated to be 1.05, and a corrected pose estimate for camera 5 is computed. Finally the remaining 1066 keypoint inliers of image pair $(4, 5)$ are triangulated and added to the cluster, and the cluster is optimized.

6.3 The Clustering Algorithm

With the two basic steps of clustering covered in the previous sections, in this section the developed clustering algorithm itself is documented. An important factor driving the development of the clustering algorithm is that it should make effective use of the available data, and preserve as much information as possible. That is, the algorithm should strive to obtain a maximum number of stable keypoints, but without introducing bad measurements. Having a large number of triangulated points with correct projections in the images of the cluster improves the quality of the recovered structure and motion, because more information is available for bundle adjustment in the optimization step.

To obtain the maximum number of keypoints, a good strategy is to build the cluster using the matching image pairs with most keypoint inliers. This approach was taken in the first two steps of clustering explained in sections 6.1 and 6.2, and it can be applied each time an image is added to the cluster. The strategy is analogous to finding a maximum spanning tree in a connected graph formed by vertices representing the input images and edges representing matching pairs of input images. In the following section an introduction to the concept of maximum spanning trees is given, and then in section 6.3.2 this is applied to the problem of clustering.

6.3.1 Maximum Spanning Trees

A connected, undirected graph is given by $G = (V, E)$, where V is a set of vertices, and E is a set of edges representing possible connections between the vertices. For each edge $(u, v) \in E$ there is an associated weight $w(u, v)$ specifying the cost of connecting the vertices u and v using this edge. In many applications an acyclic subset $T \subseteq E$ that connects all the vertices, and which minimizes the total weight

$$w(T) = \sum_{(u,v)\in T} w(u, v) \tag{6.13}$$

is sought. In this situation T represents a tree which is referred to as a minimum spanning tree [18]. In the same way a maximum spanning tree can be found as an acyclic subset $T \subseteq E$ that maximizes the total weight in (6.13).

A simple algorithm for finding minimum spanning trees is Prim's algorithm, which is easily adapted to the problem of finding a maximum spanning tree. The algorithm is greedy in the sense that it always makes the choice that is best at the moment. This strategy is not generally guaranteed to find globally optimal solutions, but for this problem an optimal solution is obtained [18].

The overall idea of Prim's algorithm is explained in the following, where it has been adapted to finding a maximum spanning tree. The algorithm is initialized by letting $T = \emptyset$. Then as long as T does not form a tree spanning all vertices V, find a safe edge $(u, v) \in (E - T)$ and add it to T. A safe edge is an edge that has maximum weight and connects an isolated vertex in the graph $G_T = (V, T)$ to the tree T. For the first step, an edge $(u, v) \in E$ with maximum weight can be selected as a safe edge. Upon termination, the set of edges in T define a maximum spanning tree of the graph G.

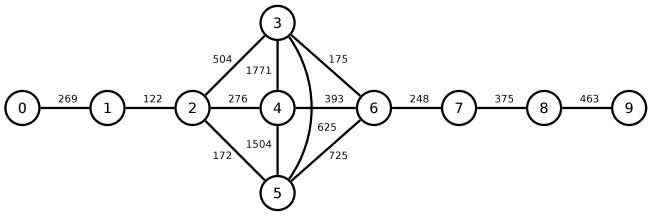


Figure 6.6: The graph $G = (V, E)$ representing a data set consisting of 10 images. The vertices $V = \{0, \ldots, 9\}$ represent the input images, while the edges $(u, v) \in E$ represent matching image pairs. The weight $w(u, v)$ of an edge is the number of keypoint inliers for image pair (u, v).

6.3.2 Building the Cluster

Based on the above clustering can be seen as the task of finding a maximum spanning tree of a graph $G = (V, E)$, where V is the set of input images, E is the set of matching image pairs (u, v) found in the matching step, and the weight $w(u, v)$ of each edge is the number of keypoint inliers for the image pair (u, v). During the process, camera matrices P_j corresponding to the input images and space points $\mathbf{X}_i$ corresponding to the keypoints of the images are recovered. In figure 6.6 the graph G is shown for a data set consisting of 10 images. This data set, which was also used for the examples of recovered structure and motion in figure 6.5, is used for illustration throughout the explanation of the clustering algorithm below.

For efficient implementation, two distinct sets of edges are maintained in the clustering algorithm: a set of free edges F containing all matching image pairs that are not yet part of the cluster, and a set of open edges O containing all matching image pairs which have one image in the cluster. In the following the two main steps of the algorithm, initialization of the cluster and augmenting the cluster, are treated.

Cluster Initialization First all matching image pairs are added to the set of free edges by letting $F = E$ and the set of open edges $O = \emptyset$. Then the free edge $(a, b) \in F$ which has maximum weight is selected as the first edge of the spanning tree, and this edge is removed from F. The selected edge (a, b) corresponds to the best matching image pair, and it is added to the cluster using the method of section 6.1. Now the cluster consists of two cameras defined by P_j, $j \in \{a, b\}$ and a set of triangulated keypoints $\mathbf{X}_i$ with corresponding projections $\mathbf{x}_{i,j}$. As a final step of initialization all matching image pairs that are connected to either image a or b are moved from the set of free edges F into the set of open edges O.

In figure 6.7a a graph illustrating the result of this process is shown, where images from the best matching image pair (3,4) are included in the cluster. As can be seen from the figure, six edges are moved to the set of open edges, and images 2, 5 and 6 are candidates for being added to the cluster next.

Augmenting the Cluster In this step the maximum spanning tree is grown, and it is repeated until all input images are part of the cluster. The open edge $(b, c) \in O$ with maximum weight is selected for augmenting the cluster, and

it is removed from O. Suppose that image b is already part of the cluster, then image c is being added. The edge (b, c) is a *strong* edge, because it represents the matching image pair with most keypoint inliers that connects image c to the cluster. There may be other open edges in O connecting image c to the cluster, but they have less keypoint inliers and are thus *weak* edges.

The new image c is added to the cluster via the strong edge representing image pair (b, c) using the method of section 6.2. In this process the new camera defined by P_c is added to the cluster, any keypoints $\mathbf{X}_i$ already in the cluster that are also inliers of image pair (b, c) are updated with projections $\mathbf{x}_{i,c}$ for bundle adjustment, and the remaining keypoint inliers of image pair (b, c) are triangulated from corresponding projections $\mathbf{x}_{i,b}$ and $\mathbf{x}_{i,c}$ yielding new keypoints $\mathbf{X}_i$ which are added to the cluster.

Any weak edges connecting image c to the cluster are now also removed from the set of open edges O, but see below for how weak edges are utilized. After image c has been added to the cluster, all free edges representing image pairs connected to that image are moved from the set F into the set of open edges O. As long as open edges remain, the cluster is augmented using this method.

Continuing the example from above, a graph illustrating the result of augmenting the cluster with a third image is shown in figure 6.7b. The open edge $(4, 5)$ is selected as the strong edge connecting image 5 to the cluster, but there is also a weak edge $(3, 5)$ which has less keypoint inliers, see below. After image 5 has been added to the cluster, the two edges $(2, 5)$ and $(5, 6)$ are moved to the set of open edges, and image 2 and 6 are candidates for being added to the cluster next.

Utilizing Weak Edges As discussed above, the clustering algorithm should strive to obtain a maximum number of stable keypoints, but without introducing bad measurements. One way to improve stability of keypoints is to add projections from more images when available, but this only helps if the projections added are actually correct.

Consider a cluster initialized from the matching image pair (a, b), which is augmented with a new image c via the strong edge (b, c). Now suppose that there is a weak edge (a, c) that also connects image c to the cluster. Then the keypoint inliers of image pair (a, c) are likely to be correct, and can thus be utilized for updating the keypoints that are already triangulated with more projections for bundle adjustment. To avoid introducing bad measurements, however, no new keypoints are triangulated from weak edges.

Experiments have shown that for most data sets, simply adding projections from all weak edges may lead to introduction of bad measurements. Therefore a parameter W is defined, which limits the number of weak edges that may be utilized when augmenting the cluster with a new image. Then at most the best W weak edges are utilized. The value of W is discussed in the following section.

Continuing the example from above, figure 6.7c shows the graph resulting from augmenting the cluster with image 6 via the strong edge $(5, 6)$ and utilizing weak edges $(4, 6)$ and $(3, 6)$. The previously free edge $(6, 7)$ is now moved to the set of open edges. Figure 6.7d shows one more iteration, and in figure 6.7e the complete graph obtained from clustering this data set is shown. As can be seen in the last figure, the strong edges form a maximum spanning tree of the original

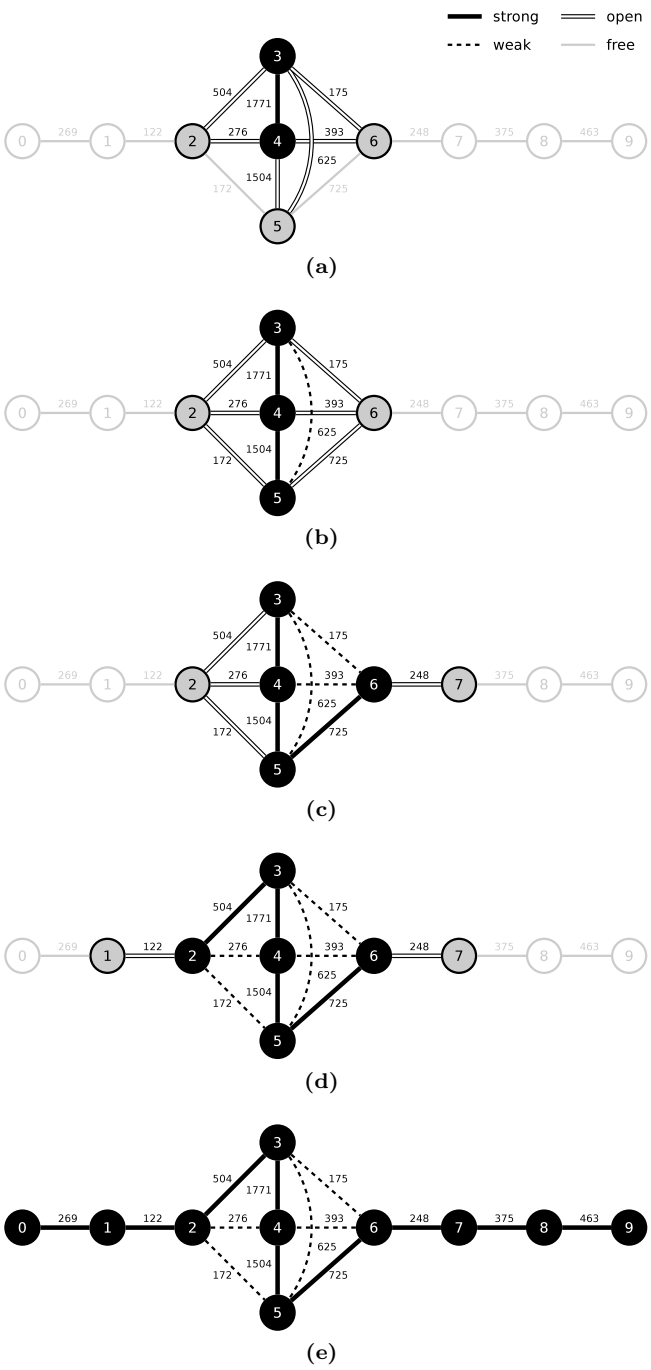


Figure 6.7: Graphs illustrating the progress during clustering for a data set. Vertices marked in black represent images in the cluster, and vertices filled with gray represent candidate images for being added to the cluster next. Figure (a) shows the state after cluster initialization, and figures (b)–(d) show how the cluster is augmented with more images. In figure (e) the final graph is shown, where the strong edges form a maximum spanning tree.

graph $G = (V, E)$ shown in figure 6.6. The weak edges are the remaining edges of the graph, but for other data sets some matching image pairs may be ignored if they are not among the best weak edges in any iteration of the algorithm.

The clustering algorithm terminates when there are no more open edges available in O, and the final result is the set of camera matrices P_j corresponding to all input images, and the set of space points $\mathbf{X}_i$ with associated keypoint locations $\mathbf{x}_{i,j}$ in the input images.

6.3.3 Evaluation

The graphs in figure 6.7 illustrate the progress of clustering for one data set, but the resulting graph in figure 6.7e is representative for most of the data sets used in this project. From this graph it is seen that the strong edges connect the input images in sequence. This is because the images in this data set were captured with movement primarily along one direction, but this is not required by the clustering algorithm. In fact the tree formed by the strong edges can be any kind of tree. In the figure it is also seen that weak edges typically connect images close to each other in the sequence. This is also due to the way the images of this data set are captured, and in general weak edges are most likely to be present for images that are captured from similar perspectives.

For most data sets used in this project maximum of $W = 2$ weak edges utilized when augmenting the cluster with an image works well, and adding projections from the weak edges increases the number of stable keypoints obtained. But for one data set, using more than one weak edge leads to a significant decrease in the number of keypoints obtained, and thus for this data set a value of $W = 1$ was chosen. Evaluation of the structure and motion results obtained using the clustering algorithm treated here is postponed to section 6.5.1.

6.4 Bundle Adjustment

Each time the cluster is augmented with an image, bundle adjustment is applied to the recovered structure and motion as part of an optimization step. In this section, an overview of bundle adjustment is given. For a more in-depth treatment refer to [23] and [26].

Given a set of images depicting a number of 3D points from different viewpoints, bundle adjustment solves the problem of simultaneously refining the positions of these points and the calibration parameters of the cameras that captured the images, based on observed 2D projections of the points in the images. Bundle adjustment provides a statistically optimal solution to this problem, and under the assumption that the error of observed projections is zero-mean Gaussian, bundle adjustment is the maximum likelihood estimator [26]. The name refers to the bundles of light rays originating from each point in the scene and converging in the optical centre of each camera, which are adjusted optimally with respect to both structure and motion parameters.

Essentially bundle adjustment amounts to minimizing the reprojection error between observed and predicted image points. This error is expressed as a sum of squares of a large number of nonlinear, real-valued functions, and thus nonlinear least-squares algorithms are employed for minimization. For bundle adjustment the Levenberg-Marquardt algorithm is typically used, because it

has the ability of converging quickly while still being tolerant to a wide range of initial guesses [26].

More formally, the problem of bundle adjustment can be stated as follows. Assume that n points in 3-space are observed in m images, and denote by $\mathbf{x}_{i,j}$ the homogeneous 3-vector representing the observed projection of the i^{th} point on the j^{th} image. Then bundle adjustment amounts to refining the m camera matrices P_j and the n points represented by homogeneous 4-vectors $\mathbf{X}_i$, such that the observed projection $\mathbf{x}_{i,j}$ is closely approximated by the predicted projection $\hat{\mathbf{x}}_{i,j} = \mathrm{P}_j\mathbf{X}_i$. Here it is assumed that an initial estimate of the camera matrices P_j and points $\mathbf{X}_i$ are available, as is the case when bundle adjustment is used in combination with the clustering algorithm treated above.

Typically each 3D point i is observed only in a subset of the m images. Therefore let $v_{i,j}$ denote binary variables that equal 1 if point i is visible in image j and 0 otherwise. Now by parameterizing each camera j by a vector $\mathbf{a}_j$ and each 3D point i by a vector $\mathbf{b}_i$ as explained in the following section, bundle adjustment minimizes the reprojection error with respect to all 3D points and camera parameters. Specifically

$$\arg\min_{\mathbf{a}_j,\mathbf{b}_i} \sum_{i=1}^{n}\sum_{j=1}^{m} v_{i,j}\, d\big(\mathbf{Q}(\mathbf{a}_j,\mathbf{b}_i),\ \mathbf{x}_{i,j}\big)^2, \tag{6.14}$$

where $\mathbf{Q}(\mathbf{a}_j,\mathbf{b}_i)$ is the predicted projection $\hat{\mathbf{x}}_{i,j}$ of point i on image j, and $d(\mathbf{x},\mathbf{y})$ denotes the Euclidean distance between image points represented by the vectors $\mathbf{x}$ and $\mathbf{y}$. From (6.14) it is clear that bundle adjustment minimizes a physically meaningful criterion, namely the geometric reprojection error, as opposed to algebraic approaches [26].

6.4.1 Sparse Bundle Adjustment

If bundle adjustment is implemented naively the complexity of the problem is very large due to the high number of minimization parameters. In the case of Euclidean reconstruction with intrinsically calibrated cameras as in this project, each 3D point gives rise to 3 parameters, and each camera gives rise to 6 parameters as explained below. Thus the total number of minimization parameters becomes $3n + 6m$. For instance when applying bundle adjustment in the last step of clustering for the data set illustrated by the graph in figure 6.7e, the minimization involves 4426 points in 10 images leading to total of 13338 variables. Fortunately it is possible to take advantage of the fact that there is no interaction among parameters for different 3D points and cameras. When solving the minimization problem of bundle adjustment, the normal equations involved have a sparse block structure due to this lack of interaction, and algorithms for sparse bundle adjustment taking advantage of this are available. In this project SBA[1], which is an open source library providing a C/C++ implementation of generic sparse bundle adjustment [26], is used.

The algorithm implemented in the SBA library is generic in the sense that cameras and points can be parameterized in different ways depending on the specific problem at hand. As discussed above, in this project bundle adjustment is applied to the problem of Euclidean reconstruction with intrinsically

[1] `http://www.ics.forth.gr/~lourakis/sba/`

(a) **(b)**

Figure 6.8: Example of the inversion of camera and point positions that might happen during bundle adjustment. a) Correct structure and motion. b) The cluster has been inverted, and points appear behind the cameras by which they are observed.

calibrated cameras. In this configuration the n 3D points are trivially represented using 3-dimensional vectors $\mathbf{b}_i$. The m cameras, however, are defined by normalized camera matrices P_j. Instead of including all elements of the matrices $\mathrm{P}_j = [\mathrm{R}_j \ \mathbf{t}_j]$ in bundle adjustment, the cameras are parameterized using 6-dimensional vectors $\mathbf{a}_j$ which capture the 6 degrees of freedom for each camera, i.e. 3 for rotation and 3 for translation. The results of applying bundle adjustment to the recovered structure and motion are evaluated in section 6.5.1.

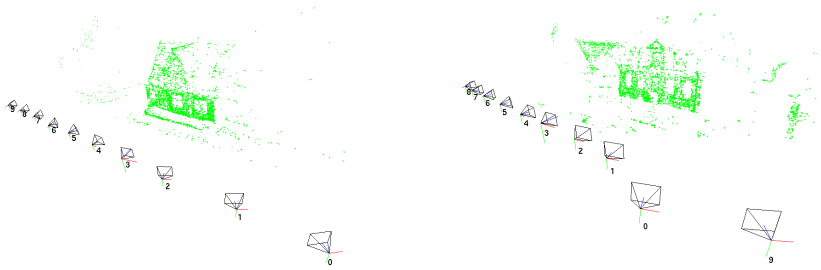
6.5 Optimization and Robustness

In this final section on clustering, the optimization step that is applied each time an image has been added to the cluster is treated. The primary reasons for performing optimization throughout the clustering process are, as discussed in the chapter introduction, that this minimizes the risk of a bad initial guess preventing recovery of structure and motion, and that simpler methods can be employed without affecting the end result. But this strategy also leads to an improvement in the obtained structure and motion estimates.

Optimization is performed for the current state of the cluster, and the first step is to apply bundle adjustment to the recovered structure and motion using the method described in the previous section. That is, the camera matrices P_j and the points $\mathbf{X}_i$ currently in the cluster are updated with the statistically optimal parameters given the observed projections $\mathbf{x}_{i,j}$.

As discussed in section 5.3.2 in general least-squares methods are sensitive to outliers, and this also applies to the bundle adjustment method described in the previous section. Although most keypoint outliers are discarded during matching, previously undetected outliers may manifest themselves during clustering. Therefore after bundle adjustment has been applied, any points that have a maximum reprojection error above some threshold are removed from the cluster to improve robustness. This prevents these points from having negative impact during successive iterations of clustering. The value of the threshold and the effect of pruning such points is discussed in the following section.

During development it was discovered, that on rare occasions the positions of cameras and points in the cluster were inverted during bundle adjustment. The effect of this inversion is that all points suddenly appear behind the cameras



(a) **(b)**

Figure 6.9: Examples of the final recovered structure and motion for two data sets. a) Cluster consisting of 10 images and 4426 points. b) Cluster consisting of 10 images and 4087 points.

by which they are observed. This situation is illustrated in figure 6.8. Such inversion might happen, because the SBA library does not enforce the constraint that observed points must be in front of cameras. Mathematically the inverse solution is equally good, but when augmenting the cluster with another image, this inversion causes problems and therefore it needs to be detected and corrected for clustering to be robust.

The last task of the optimization step is to normalize the cluster, such that the distance between the cameras of the first image pair added to the cluster is equal to 1. This normalization is not strictly necessary, because the recovered structure and motion is always subject to an arbitrary similarity transform, but it makes the results of clustering more consistent.

In addition to the optimization and robustness measures discussed above, which are applied after augmenting the cluster, a couple of precautionary measures are applied during clustering itself. Specifically, when triangulating new points, any points that end up behind the cameras by which they are observed are discarded. And when updating a point with additional projections, if it is discovered that the point matches different keypoints it is deemed a mismatch, and the point is discarded and any associated keypoints are excluded from further processing.

6.5.1 Evaluation

In figure 6.9 two examples of the final structure and motion recovered using the clustering method developed in this chapter are shown. Figure 6.9a shows the cluster corresponding to the graph in figure 6.7e, and in the following the results of clustering for this data set is used for discussion.

The cluster consists of 10 images and 4426 points with a total of 10196 observed projections, i.e. corresponding keypoint locations, in the images. Thus on average each stable keypoint has projections in 2.3 images, see the histogram in figure 6.10a. The RMS reprojection error for points in the cluster is 0.22 pixels, which is very good. In figure 6.10b a histogram of the reprojection error for observed projections is shown, and as can be seen virtually all keypoints have reprojection errors below one pixel. This supports that the structure and motion is recovered with high accuracy. In figure 6.11a a box plot of the reprojection errors for points in the cluster with different number of projections

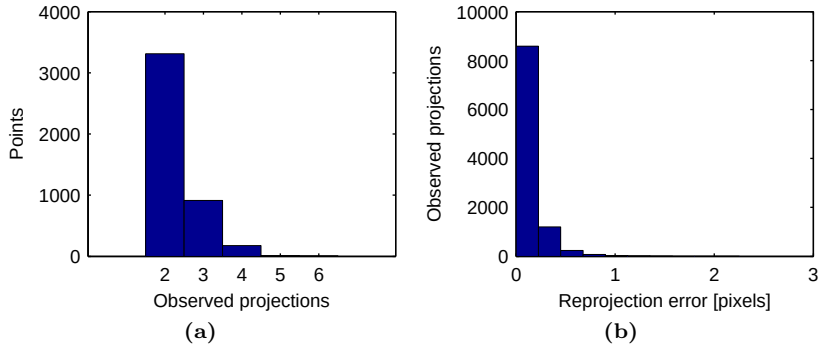


Figure 6.10: a) Histogram of the number of projections for points in the cluster. b) Histogram of the reprojection error for observed projections.

is shown. From this figure it is seen that there is a trend that the reprojection error becomes larger as the number of projections increases for a point. This is to be expected, as points with more projections are more constrained.

One of the tasks during the optimization step is, as discussed above, to remove points that after bundle adjustment have a reprojection error above some threshold. By analyzing the reprojection errors obtained for the data sets used in this project without removing such points, it was determined that the majority of points had reprojection errors below 2.5 pixels, and therefore this value was chosen for the threshold. When removing bad points, the cluster in figure 6.9a has a RMS reprojection error of 0.22 pixels, but if bad points are not removed this error becomes 3.1 pixels, and the maximum reprojection error for the points in this case becomes 195 pixels. Thus it is clear that removing points with high reprojection error leads to better results.

Finally the performance of bundle adjustment is evaluated. A scatter plot of the RMS reprojection error before and after applying bundle adjustment is shown in figure 6.11b. This plot is based on all iterations of clustering for all data sets used in the project, and as can be seen from the figure, bundle adjustment consistently minimizes the reprojection error. The large errors before bundle adjustment are typically caused by the utilization of weak edges when augmenting the cluster, because the added projections do not match with the current parameters of the cameras in the cluster.

6.6 Conclusion

In this chapter the clustering step, which is the last step of the proposed reconstruction method dealing with recovery of structure and motion, has been treated in detail. In sections 6.1 and 6.2 the two basic steps of clustering were treated, namely initialization of the cluster from the best matching image pair, and augmenting the cluster with a third image. Based on this, the developed clustering algorithm was described in section 6.3. The algorithm builds the cluster iteratively by growing a maximum spanning tree in a graph representing the input images and the matches between them. This approach is taken to obtain a maximum number of stable keypoints, and after each iteration of

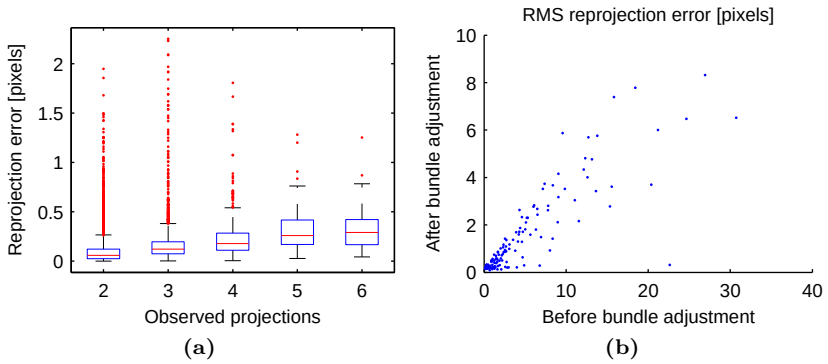


Figure 6.11: a) Box plot of reprojection errors grouped by the number of projections for each point. The boxes indicate the lower quartile, median, and upper quartile values. b) RMS pixel reprojection error before and after bundle adjustment. The graph includes all iterations of clustering for all data sets used in this project.

clustering, the recovered structure and motion is optimized. The optimization consists of bundle adjustment, of which an overview was given in section 6.4, and a number of robustness measures as explained in section 6.5. The structure and motion obtained using the methods covered in this chapter and the two previous chapters provide a solid foundation for reconstructing a textured mesh of the building, which is the topic of the following two chapters.

Chapter 7

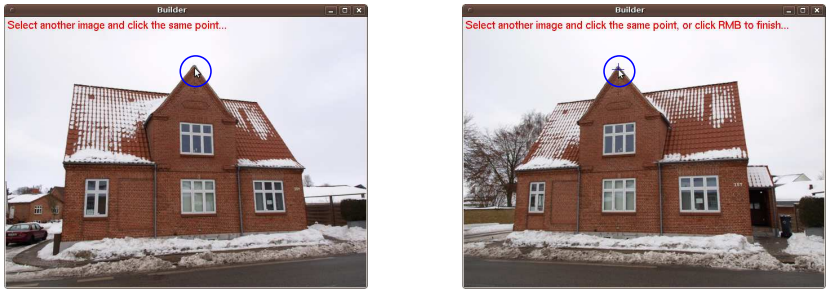
Coarse Model Reconstruction

In this chapter, the coarse model reconstruction step of the proposed reconstruction method is documented. As discussed in section 3.1.1, this is the only step requiring user interaction, and the purpose is to obtain a coarse model of the building to reconstruct. The coarse model is a textured mesh with polygons representing large planar surfaces of the building such as walls and roof surfaces etc., see figure 3.3 for an example. That is, the model should not contain details such as recessed windows and doors, as these are automatically added in the automatic façade reconstruction step. The user is only responsible for defining the shape of the coarse model, and this process is explained in the following section. The appearance of the model, that is textures for the individial polygons, is then automatically extracted from the input images using the method explained in section 7.2.

7.1 User Assisted Reconstruction

The reconstruction of a coarse model is done interactively using an application with a simple user interface that has been developed for the purpose. Among other things, this application allows the user to preview the input images, to navigate the structure and motion recovered during clustering, and to build a coarse model as explained in the following.

A simple approach to coarse model reconstruction has been chosen in this project, because the primary focus of the project regarding model reconstruction is automatic façade reconstruction. The method for user assisted model reconstruction implemented in this project consists of two steps, and it is inspired by the methods analyzed in section 2.2.3. First step is to define a set of locators, which represent 3D positions of selected features of the building, e.g. the corners of a wall. The second step is to define polygons that span the planar surfaces of the building, using the defined locators as vertices of the polygons. The two steps can be performed in any order, as long as all locators needed by a particular polygon are defined before the polygon itself. The steps are described in the two following sections.



(a) **(b)**

Figure 7.1: Interactively defining a locator. a) The user has clicked a feature in one image, highlighted with a blue circle. b) The user has switched to another image and clicked the same feature. From the given locations a new locator is triangulated.

7.1.1 Defining Locators

In the user interface it is possible to switch between the input images of the cluster, and a locator is defined using the mouse simply by clicking the same feature of the building in two or more of the input images. This process is illustrated in figure 7.1.

The 3D position of the locator is then estimated using the camera calibration parameters available from the preprocessing and clustering steps. Specifically, denote by $\mathbf{l}_{i,j}$ the user defined location in pixel coordinates of locator i in image j. Given two such points $\mathbf{l}_{i,a}$ and $\mathbf{l}_{i,b}$ corresponding to the same feature in images a and b respectively, an estimate of the 3D position of the locator $\mathbf{L}_i$ can be obtained using the triangulation method described in section 6.1.1. For triangulation, the camera matrices P_a and P_b from the cluster and the two image points expressed in normalized coordinates are used. As in section 6.1.1, normalized coordinates are obtained by applying the inverse camera calibration matrix K_j^{-1} to the pixel coordinates. For simplicity, the locator position is estimated using only two of the user defined locations, see the discussion in section 7.1.3.

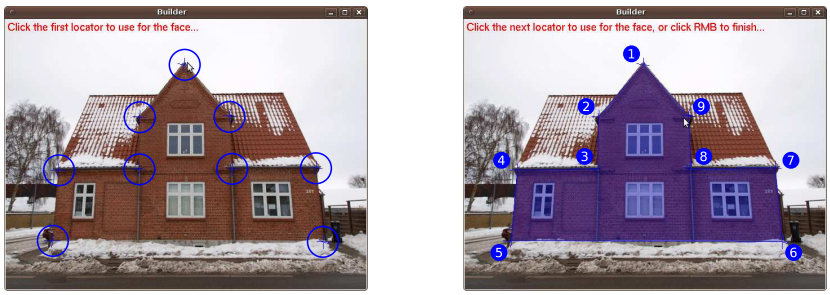
7.1.2 Defining Polygons

In the user interface, a polygon is defined simply by clicking the locators to use as vertices in counterclockwise order when the polygon is seen from the front, and this is illustrated in figure 7.2. Each polygon of the coarse model is described by the sequence of locator indices obtained this way.

When defining a polygon, the partially constructed polygon is shown interactively, and approximate texture extraction is performed in real-time, such that the user gets immediate visual feedback. When the user has finished defining a polygon of the coarse model, a refined texture is extracted for that polygon as explained in section 7.2.

7.1.3 Evaluation

As discussed above, the user is responsible only for defining the shape of the coarse model, i.e. selecting features in the input images to define locators, and



(a) **(b)**

Figure 7.2: Interactively defining a polygon. a) The user has initiated polygon creation after having defined the 9 locators highlighted with blue circles. b) The locators are clicked in the indicated order, and the created polygon is overlaid on the image.

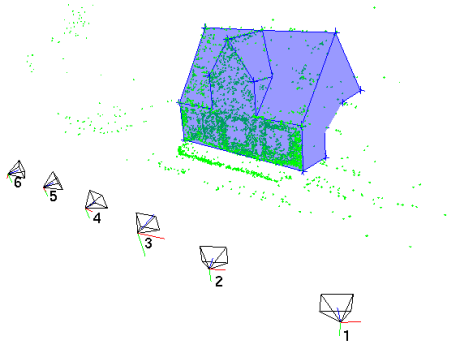


Figure 7.3: Example of the geometry of a reconstructed coarse model. In addition to the model, part of the recovered structure and motion is shown.

defining polygons based on these locators. In figure 7.3 the geometry of a coarse model reconstructed using this method is shown together with the recovered structure and motion.

At this point it is not enforced that the vertices of a polygon in the coarse model, i.e. the positions of a subset of the defined locators, lie in the same plane. It is simply assumed that the vertices of defined polygons are close to being coplanar. Whether this assumption holds, however, depends largely on the accuracy of the locations defined by the user. For the coarse models reconstructed from the data sets used in this project, the impact on the results from this assumption is insignificant.

As mentioned above, the position of locators is estimated from two user defined locations for simplicity. Better accuracy may be obtained by utilizing more than two image points if supplied by the user. One way to optimize locator positions is to use triangulation for obtaining an initial estimate, and then apply bundle adjustment where the camera parameters are kept constant. However, the accuracy obtained using the simple method implemented in this project is sufficient for using the results in development of the automatic façade reconstruction method.

(a) (b)

Figure 7.4: Example of a model with and without rectified textures. a) Without rectification the windows appear skewed. b) With rectification the appearance is correct.

7.2 Texture Extraction

In this section, the developed method for extraction of refined polygon textures is documented. Fortunately the appearance of the building is available in the input images, and knowing the camera calibration parameters corresponding to each input image, mapping from the reconstructed coarse model to the input images is straight forward.

One idea for texture extraction is to use the input image that best views a polygon directly as texture for that polygon. Unfortunately the visual quality of the reconstructed model obtained using this approach is poor. The problem is that the depiction of the building surface in the input image in virtually all cases is subject to a perspective transform. I.e. the building surface does not lie in a plane parallel to the image plane of the input image, and thus appears warped in the image. Therefore if using the best available input image for texturing a polygon without compensating for this, the visual quality becomes inadequate. For correct appearance it is necessary to extract a rectified texture for the polygon. In figure 7.4 an example of a model with and without rectified textures is shown.

The developed method for extraction of rectified polygon textures consists of the following steps: plane fitting, computing plane axes, finding the best image, computing vertex projections, and finally creating the rectified texture. These steps are illustrated in figure 7.5, and each of the steps are explained in the following sections.

7.2.1 Plane Fitting

As discussed in section 7.1.3, the locators used for defining the vertices of a polygon may not lie in the exact same plane, because their positions are estimated from the image points given by the user. Therefore the first step in texture extraction is to find the plane, which best fits the locators defining the polygon. The final rectified texture will contain the appearance of the building surface as if an image was captured by pointing a camera directly towards the surface at a right angle. That is, the image plane of this virtual camera would be parallel with the plane found in this step.

In the following suppose that a single polygon of the coarse model is defined by the sequence of locator indices from 1 to n. Then the vertex positions of this

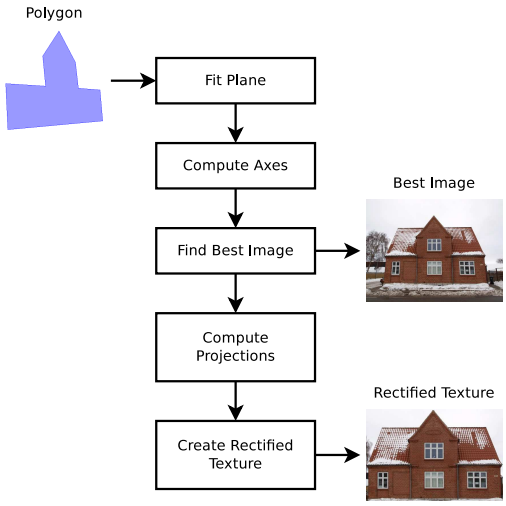


Figure 7.5: Overview of the steps involved in extraction of a rectified texture for a polygon of the coarse model.

polygon are the estimated locator positions $\mathbf{L}_i$, $i = 1, \ldots, n$. Plane fitting is now the problem of finding the plane that minimizes the orthogonal distances from these locators to the plane. The centroid $\mathbf{m}$ of the locators lies in this plane, and can be computed as

$$\mathbf{m} = \frac{1}{n} \sum_{i=1}^{n} \mathbf{L}_i. \tag{7.1}$$

Denoting by $\mathbf{n}$ the unit normal vector of the fitted plane, then the orthogonal distance from locator $\mathbf{L}_i$ to the plane becomes $(\mathbf{L}_i - \mathbf{m}) \cdot \mathbf{n}$. The fitted plane is defined by the known centroid $\mathbf{m}$ and the normal $\mathbf{n}$ which is to be found. A least-squares solution is sought, and this problem can be stated as finding the normal $\mathbf{n}$ which minimizes the sum

$$\sum_{i=1}^{n} \left((\mathbf{L}_i - \mathbf{m}) \cdot \mathbf{n} \right)^2. \tag{7.2}$$

By expressing the problem using n equations on the form $(\mathbf{L}_i - \mathbf{m}) \cdot \mathbf{n} = 0$, and building the corresponding $n \times 3$ matrix

$$\mathrm{A} = \begin{bmatrix} \mathbf{L}_1^\mathsf{T} - \mathbf{m}^\mathsf{T} \\ \vdots \\ \mathbf{L}_n^\mathsf{T} - \mathbf{m}^\mathsf{T} \end{bmatrix}, \tag{7.3}$$

a solution can be obtained using SVD, see appendix B for details. Letting $\mathrm{A} = \mathrm{UDV}^\mathsf{T}$, the sought normal $\mathbf{n}$ is the third column of V, and thus the best fitting plane has been determined.

7.2.2 Computing Plane Axes

The best fitting plane found above is completely described by the centroid $\mathbf{m}$ and the unit normal $\mathbf{n}$. However, to create a rectified texture, it is necessary in

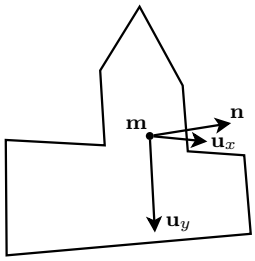


Figure 7.6: Example of the axes $\mathbf{u}_x$ and $\mathbf{u}_y$ computed for a plane defined by the centroid $\mathbf{m}$ and the normal $\mathbf{n}$. The x-axis of the plane has the same direction as the longest edge of the polygon.

addition that two axes perpendicular to each other and lying in the plane are defined. These axes define the x- and y-axis of the rectified texture. There is an infinite number of choices for the axes as they can rotate freely around the plane normal, and therefore it is necessary to select a specific orientation.

In this project, the x-axis is simply selected such that it is parallel with the longest edge of the polygon. For instance, the longest edge of the polygon defined in figure 7.2b is the edge between locators 5 and 6, and thus the x-axis of this polygon becomes parallel with that edge. This approach may not be optimal in all situations, however, as discussed in section 7.2.6.

More specifically, denote by $\mathbf{u}_x$ and $\mathbf{u}_y$ the perpendicular unit vectors defining the x- and y-axis in space of the plane respectively. First $\mathbf{u}_x$ is computed as the unit vector having the same direction as the longest polygon edge projected onto the plane. Then the y-axis is computed as $\mathbf{u}_y = \mathbf{u}_x \times \mathbf{n}$, and thus the vectors $\mathbf{u}_x$, $\mathbf{u}_y$, and $\mathbf{n}$ form an orthonormal basis. An example of the plane axes obtained for a polygon is shown in figure 7.6.

7.2.3 Finding the Best Image

To create a rectified texture for a polygon, it is necessary to determine which of the input images that best views the polygon. In fact there may not be a single image which contains a non-occluded view of the whole polygon, so it may be necessary to merge multiple input images to avoid this problem. In this project, however, it is assumed that it is sufficient to use a single input image for the polygon texture, and thus this step reduces to selecting the best image available.

One approach to solving this problem is to select the image, which most directly views the whole polygon. That is, selecting the input image corresponding to the camera in the cluster whose line of sight has the smallest angle to the normal vector of the plane of the polygon, see figure 7.7a. It was discovered, however, that often the view of a polygon in the image selected this way is occluded by other parts of the building, even though an input image with no such occlusions is part of the cluster.

To avoid such occlusions in more situations, the input image corresponding to the camera with the smallest angle between the normal of the plane and the direction from the centroid to the camera is used instead, see figure 7.7b. The idea is that this approach favours cameras that are closer to the polygon.

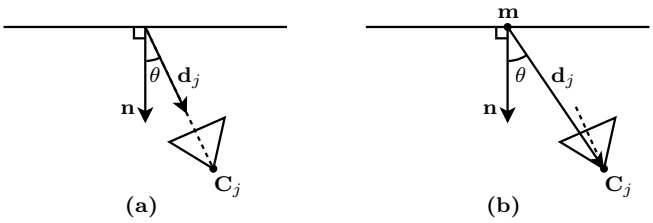


Figure 7.7: Illustration of two approaches for selecting the best image. a) Use the angle between the plane normal and the camera line of sight. b) Use the angle between the plane normal and the vector from the centroid to the camera.

Specifically, the direction from the centroid $\mathbf{m}$ to the camera defined by P_j is represented by the vector $\mathbf{d}_j = \mathbf{C}_j - \mathbf{m}$, where $\mathbf{C}_j$ is the position of the camera computed as in (6.10). Now the angle θ between the vector $\mathbf{d}_j$ and the normal $\mathbf{n}$ of the plane can be computed from

$$\cos\theta = \frac{\mathbf{d}_j \cdot \mathbf{n}}{||\mathbf{d}_j||}. \tag{7.4}$$

It is only possible that camera j views the polygon when $\cos\theta > 0$, and as this value goes towards 0 the quality of the obtained rectified texture decreases. Therefore for the selected image it is required that $\cos\theta > \tau > 0$, where the threshold τ is set to $\cos 80°$ in this project. Furthermore it is required that the reprojections of all locators $\mathbf{L}_i$ used as vertices for the polygon lie within the boundary of the selected image, because otherwise a texture for the whole polygon can not be extracted.

In the following sections let a denote the index of the image that best views the polygon. Then using the method developed above, this image is found as the image for which (7.4) attains the maximum value being at least τ, with the additional constraint that all locators are visible in the image. If an image satisfying these requirements is not available in the cluster, a rectified texture can not be extracted for the polygon, and a dummy texture is used instead.

7.2.4 Computing Vertex Projections

Now knowing the plane of the polygon, two perpendicular axes lying in this plane, and the index of the best available input image, a mapping between the polygon of the coarse model and the input image can be established. For each 3D vertex of the polygon, two 2D projections are computed: a projection into the plane of the polygon, and a projection into the input image.

The vertices of the polygon are defined by the locator positions $\mathbf{L}_i$, and the projection of each vertex into the plane is computed using the plane axes $\mathbf{u}_x$ and $\mathbf{u}_y$ found above, where the coordinates are relative to the centroid $\mathbf{m}$. First the centroid is subtracted to obtain the local vertex coordinates $\mathbf{V}'_i = \mathbf{L}_i - \mathbf{m}$, which are then projected onto the plane to obtain the 3D coordinates

$$\mathbf{V}_i = \mathbf{V}'_i - (\mathbf{V}'_i \cdot \mathbf{n})\mathbf{n}. \tag{7.5}$$

Now to compute the 2D projections of the vertices, each $\mathbf{V}_i$ is projected onto

the plane axes to obtain the vector

$$\hat{\mathbf{v}}_i = \begin{bmatrix} \mathbf{V}_i \cdot \mathbf{u}_x \\ \mathbf{V}_i \cdot \mathbf{u}_y \\ 1 \end{bmatrix}, \tag{7.6}$$

which is the homogeneous 3-vector representing the 2D projection of locator i into the plane of the polygon.

The projection of each vertex into the input image is simply achieved by projecting the locator position $\mathbf{L}_i$ using the camera calibration information available for image a. That is the projection into image a of the locator with position $\mathbf{L}_i$ is computed as the homogeneous 3-vector

$$\mathbf{v}'_i = \mathrm{K}_a \mathrm{P}_a \mathbf{L}_i, \tag{7.7}$$

where the camera calibration matrix K_a is applied such that $\mathbf{v}'_i$ is expressed in pixel coordinates. The mapping between the polygon and the input image is now represented by the two corresponding sets of 2D projections: the set in the plane of the polygon defined by $\hat{\mathbf{v}}_i$, and the set in image a defined by $\mathbf{v}'_i$.

7.2.5 Creating the Rectified Texture

The final step of texture extraction is to actually create the rectified texture. Rectification of the input image is achieved by applying a perspective transform to the image. Such transformation is described by a homography, which is a 3×3 matrix that can be computed given a set of corresponding 2D points represented by homogeneous 3-vectors. In general for corresponding points $\mathbf{x}_i$ and $\mathbf{x}'_i$, the homography H transforming each $\mathbf{x}'_i$ to $\mathbf{x}_i$ is defined [23] by the relation

$$\mathbf{x}_i = \mathrm{H}\mathbf{x}'_i. \tag{7.8}$$

Thus to compute a homography for rectification of the input image a set of corresponding 2D points is needed.

From the previous step, the projections $\mathbf{v}'_i$ of the locators in image a are available, and they are expressed in pixel coordinates. The corresponding pixel coordinates $\mathbf{v}_i$ in the rectified texture to be extracted are needed, and these can be derived from the projections $\hat{\mathbf{v}}_i$ in the plane obtained in the previous step. But as the length of the computed plane axes do not correspond to the size of a pixel in the rectified texture, the projections need to be transformed. Let T be a 3×3 matrix representing a transformation consisting of isotropic scaling and translation, then

$$\mathbf{v}_i = \mathrm{T}\hat{\mathbf{v}}_i, \tag{7.9}$$

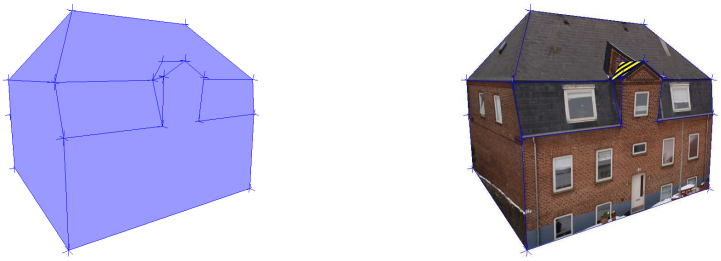
where the scale factor is chosen such that the size of a pixel in the rectified texture becomes approximately the same size as in the input image, and the translation is chosen such that the rectified texture is cropped to the polygon but leaving a small border.

In this project OpenCV[1], which is an open source library providing C/C++ implementations of algorithms for real-time computer vision, is used for both computing the homography transforming each $\mathbf{v}'_i$ in the input image to the corresponding $\mathbf{v}_i$ in the rectified texture, and for applying the perspective transform to the input image to obtain the final rectified texture.

[1] `http://opencv.willowgarage.com/wiki/`

(a) (b)

Figure 7.8: Rectification of the best input image viewing a polygon of a coarse model. a) The best input image. b) The resulting rectified texture. Note that only the region of the rectified image representing the polygon is used for texturing.



(a) (b)

Figure 7.9: An example of a reconstructed coarse model. a) Geometric model. b) Textured model.

7.2.6 Evaluation

In figure 7.8a the input image that best views the polygon defined in figure 7.2b is shown, and this image is selected according to the method described in section 7.2.3. Figure 7.8b shows the rectified texture extracted from this image using the method developed above. As can be seen from this figure, the perspective has been corrected in the rectified texture, and lines that are parallel in the plane of the polygon also appear parallel in the rectified texture.

An example of applying the developed method for extraction of refined textures to all polygons of a coarse model was shown in figure 3.3 on page 27, and in figure 7.9 another example of a textured coarse model is shown. As seen in both figures, for most polygons of the coarse models, the developed method successfully finds the input images that best view the polygons and extracts rectified textures. However, for one polygon in the roof of the coarse model shown in figure 7.9b, no suitable input image could be found. This is indicated by the black and yellow dummy texture that is used for this polygon.

Also note the occlusions from the front of the building that are present in the texture of the roof just above the polygon with the dummy texture. Such occlusions can be avoided e.g. by splitting the problematic roof polygon into smaller polygons, for which better suited images can be found, or by merging different input images as discussed in section 7.2.3.

Figure 7.10: Example of the rectified texture for a polygon for which the longest edge is not suited for defining the x-axis of the plane.

As mentioned in section 7.2.2 the approach used for selecting the x-axis of the plane of the polygon is not optimal in all situations. The approach works well for the polygon defined in figure 7.2b, because the x-axis of the rectified texture in figure 7.8b is aligned with the horizontal lines of the building. In figure 7.10, however, an example of the rectified texture for a polygon for which the longest edge is not suited for defining the x-axis is shown. The choice of x-axis for this polygon leads to reduced visual quality of the model, because e.g. the frames of the windows appear jagged due to the rotation. Using another approach for selecting the x-axis of the plane might improve the visual quality in such cases.

7.3 Conclusion

In this chapter, the coarse model reconstruction step of the proposed reconstruction method has been treated. In this step the user is responsible for defining the shape of the coarse model, and this is achieved using an application that has been developed for the purpose. The user assisted reconstruction process was described in section 7.1, and the resulting model consists of a set of user defined locators, which represent estimated 3D positions of features of the building, and a set of polygons whose vertices are defined by the locators. The appearance of the reconstructed model is then automatically extracted by creating rectified textures for the polygons of the model from the best suited input images using the method developed in section 7.2. The textured coarse model reconstructed in this step is used as the basis for automatic façade reconstruction, which is the topic of the following chapter.

Chapter 8

Automatic Façade Reconstruction

In this chapter, the final step of the proposed reconstruction method, namely automatic façade reconstruction, is documented. Given the reconstructed coarse model obtained in the previous step, the purpose of this step is, as discussed in section 3.1.1, to refine the coarse model by automatically adding façade details such as recessed windows and doors. See figure 3.4 for an example of a refined model. As discussed in section 2.3.2 of the problem analysis, the addition of such features improves the appearance of the model by making it look more realistic. In the analysis it was discovered, however, that with existing user assisted reconstruction methods, significant time is spent adding these details. Therefore in this project, a novel method for automatically adding façade details to a coarse model is proposed.

The overall idea in the proposed automatic façade reconstruction method is as follows. As part of the user assisted reconstruction process treated in the previous chapter, the user marks a subset of polygons in the coarse model as façades for which automatic refinement is wanted. Then during automatic façade reconstruction, each of the marked polygons is analyzed to identify regions of the façade with similar depths. In particular, regions representing recessed windows and doors are sought, because manually modeling these details is repetitive and time-consuming. Finally, the coarse model is updated by replacing the marked façade polygons with automatically reconstructed façades, which have recessed regions where windows and doors have been detected.

The developed method for automatic façade reconstruction, which is applied for each marked polygon in the coarse model, consists of the following steps: façade segmentation, finding region contours, and refining the model. These steps are illustrated in figure 8.1, and each of the steps are explained in the following sections.

8.1 Façade Segmentation

The façade features of interest are, as discussed above, recessed windows and doors. Based on the observation that façades of buildings typically consist of wall regions with large areas of similar color and texture, and smaller regions

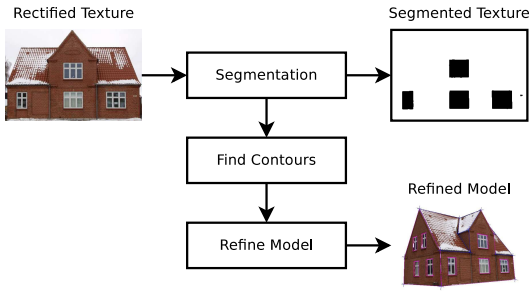


Figure 8.1: Overview of the steps involved in automatic façade reconstruction for a single polygon of the coarse model.

inside the wall regions, which represent façade features such as windows and doors, that deviate in appearance from the wall, it is deduced that the features of interest can be detected from the appearance of the façade. Thus for identifying regions in the façade of similar depth, an approach based on analysis of the rectified texture obtained for the façade polygon has been chosen.

Based on this, the purpose of this step is to segment the rectified texture of the façade polygon into regions representing wall and regions representing windows and doors. This problem can be addressed in several ways, and in the method proposed here this segmentation is achieved using a combination of different image processing algorithms as explained in the following.

8.1.1 Preprocessing

In figure 8.2a the rectified texture for a polygon that has been marked as a façade in a coarse model is shown. This is the polygon that was defined by the user in figure 7.2b. From the figure it is seen that the texture also contains information about pixels that are outside the region representing the polygon, e.g. the roof surface. Only the region of the rectified texture representing the polygon should be considered during segmentation, and therefore a binary mask indicating the region of interest is needed. Creating this mask is a matter of rasterizing the polygon defined by the pixel coordinates $\mathbf{v}_i$ computed for the rectified texture as explained in section 7.2.5. In figure 8.2b the mask obtained for the polygon is shown.

As discussed above, the wall regions of façades typically have similar color and texture. Often a wall appears primarily in one color with no significant texture, e.g. a painted wall, or it is a brick wall consisting of bricks of similar color with joints between them, as is the case for the façade in figure 8.2a. In the case of a plain colored wall, identifying the whole wall as one region is simple. For brick walls, however, it is more difficult to identify the bricks and joints between them as representing the same region of the façade, due to the high contrast between bricks and joints.

An approach to reducing this problem, is to blur the image such that brick walls appear more like plain colored walls. Therefore to even out details such as the joints between bricks and other high frequency elements of the image, it is proposed to apply Gaussian blur to the rectified texture before segmentation. This filtering is achieved by convolving the image with a kernel based on the

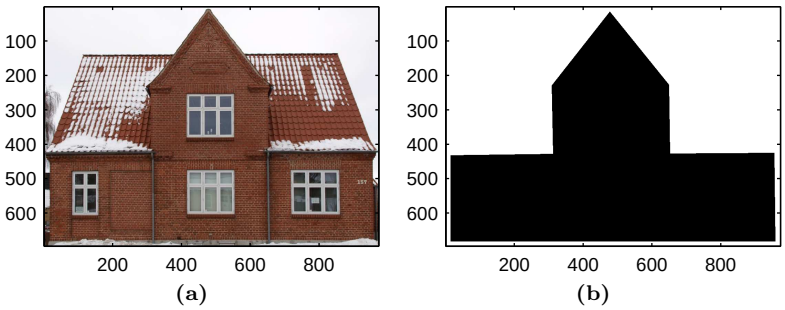


Figure 8.2: a) Rectified texture for the façade polygon. b) Binary mask with black pixels indicating the region of interest.

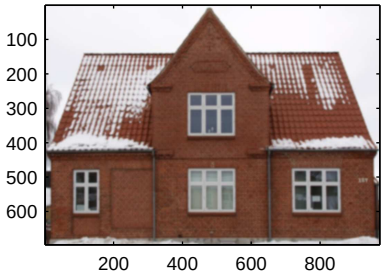


Figure 8.3: The result of applying Gaussian blur to the rectified texture. This evens out details such as the joints between bricks.

Gaussian function

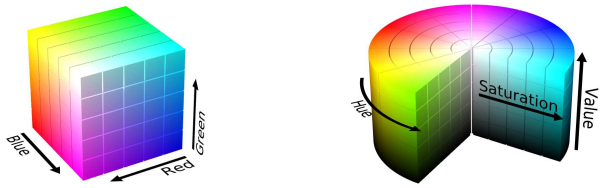
$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}, \tag{8.1}$$

where σ is the standard deviation of the Gaussian distribution [46]. In figure 8.3 the result of applying Gaussian blur to the rectified texture is shown. The choice of the value for σ is discussed in section 8.1.5.

The blurred rectified texture, referred to simply as the blurred texture in the following, is represented in the RGB color space, see figure 8.4a. This representation is not suited for segmenting the image into the different regions of the façade, however, because color and light intensity information is encoded together in the RGB values. Therefore as a final step before segmentation, the blurred image is transformed into another color space which separates color and light intensity information. Here it is proposed to use the Hue, Saturation, and Value (HSV) color space, see figure 8.4b, because this provides good separation of the colors in typical images of façades, see the following section.

8.1.2 Clustering and Classification

The next task in façade segmentation is to classify the pixels of the blurred texture as either representing wall regions or non-wall regions of the façade. Whereas the blurring and color space transformation discussed above is applied to the whole image, this classification is performed only for the pixels in the region of interest.



(a) **(b)**

Figure 8.4: Illustration of two different color spaces [47]. a) The RGB color space. b) The HSV color space.

From experiments it was concluded that an approach based on simple thresholding of the HSV values of the pixels was infeasible for automatically obtaining a correct classification. Therefore methods for unsupervised learning from the field of pattern recognition were investigated for applicability. The general idea of these methods is that a set of observations are assigned into subsets referred to as clusters, such that observations in the same cluster are similar with respect to some similarity measure. In this case the observations are HSV values of pixels in the blurred texture, and the resulting clusters contain subsets of the observations with similar HSV values. Based on the previous discussion, it is assumed that the pixels in wall regions of the façade in the blurred texture have similar color, and thus similar HSV values. Therefore, one of the clusters is likely to represent the colors of the wall, and this can be utilized for classification of the pixels.

Clustering Two methods for unsupervised clustering, namely k-means clustering, and the nearest-neighbour algorithm [20], were studied. The overall idea in k-means clustering is that the set of observations is divided into k clusters characterized by their means. The algorithm typically starts from a random guess and then iterates until it converges. To use this algorithm it is necessary to select an appropriate value for k, and experiments showed that $k = 3$ performed well for façade segmentation in many situations. However, this value, which is somewhat arbitrary, needed adjustments for some façades, and hence the method is not robust. In fact for segmentation it is sufficient to determine a single cluster which represents the colors of the wall, as discussed above, and thus there is no need to assign all observations to a cluster as is done using k-means clustering.

The solution to this problem is using a hierarchical clustering algorithm, specifically here it is proposed to use the nearest-neighbour algorithm. As opposed to k-means which uses a divisive approach for clustering, this algorithm is agglomerative which means that it starts from clusters consisting of a single observation, and then iteratively merges the two clusters that have highest similarity [20]. For the nearest-neighbour algorithm, the similarity measure is based on the distance between the closest observations of the two clusters, i.e. shorter distance means higher similarity. Specifically, let D_i and D_j be the subsets of observations in two candidate clusters for merging, then the distance between

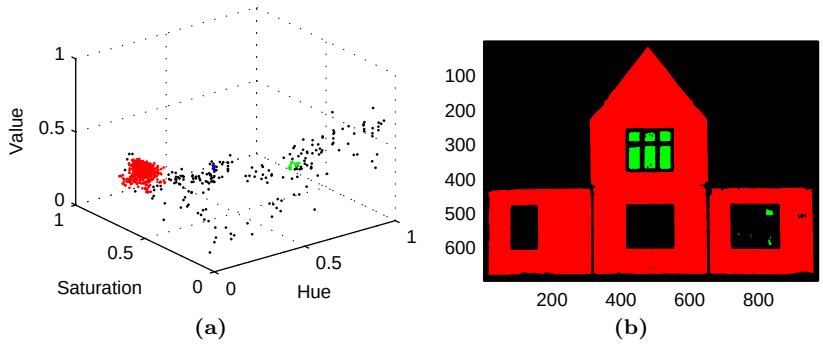


Figure 8.5: a) The single-linkage algorithm applied to the random sample of pixels from the blurred texture. The three largest clusters are marked in red, green, and blue, with red being largest. b) Classification of all pixels in the region of interest of the blurred texture. The colors correspond to the clusters of figure (a), and thus it is seen that the largest cluster (red) represents the colors of the wall regions.

the clusters is given by

$$d_{\min}(D_i, D_j) = \min_{\substack{\mathbf{x}\in D_i \\ \mathbf{x}'\in D_j}} ||\mathbf{x} - \mathbf{x}'||. \tag{8.2}$$

The algorithm may be terminated when the distance between the nearest clusters exceeds some threshold, in which case it is referred to as the single-linkage algorithm [20].

To apply the single-linkage algorithm in the context of façade segmentation, first a set of observations is obtained from the HSV values of a random sample of pixels in the region of interest of the blurred texture. Denote by S the size of this sample. Then the single-linkage algorithm is applied to this sample, and the algorithm is terminated when $d_{\min}$ for the nearest clusters exceeds a cutoff threshold c. The choice of values S and c is discussed in section 8.1.5, and in figure 8.5a an example of clustering using this method is shown for the blurred texture of figure 8.3. As is evident from this figure, the HSV values are well spread, and this makes the HSV color space suited for separating the colors.

Classification Each of the clusters resulting from the above represents observations of similar color in the random sample. Based on the observation that the total area of wall regions in façades typically is larger than the area of any other similarly colored façade region, it is assumed that the largest of the clusters, i.e. the cluster containing most observations, represents the colors of the wall. Segmentation is now a matter of classifying all pixels in the region of interest of the blurred texture based on whether they belong to the cluster representing the wall or not. For classification another representation of the wall cluster is necessary, and here it is proposed to use a multivariate Gaussian distribution with mean $\boldsymbol{\mu}$ and covariance matrix Σ computed from the observations in the wall cluster.

Classification is now performed for all pixels in the region of interest of the blurred texture by computing the Mahalanobis distance from the HSV value of each pixel to the cluster. Denoting the HSV value of the pixel by the vector $\mathbf{x}$,

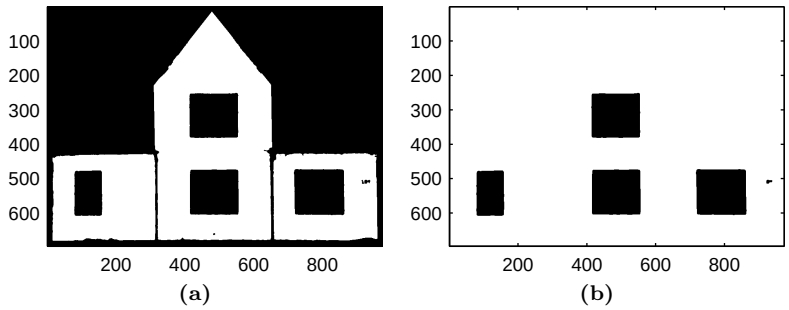


Figure 8.6: a) Binary segmentation of pixels, where zeros (white) represent wall regions, and ones (black) represent potential façade features. Note that pixels outside the region of interest initially are classified as non-wall. b) The binary segmentation after noise removal.

the squared Mahalanobis distance [20] to the cluster is given by

$$r^2 = (\mathbf{x} - \boldsymbol{\mu})^\mathsf{T} \Sigma^{-1} (\mathbf{x} - \boldsymbol{\mu}). \tag{8.3}$$

Any pixels for which the Mahalanobis distance r is below a threshold τ are marked as wall pixels, otherwise they are marked as potential façade features. In figure 8.5b the result of classifying the pixels of the blurred texture using this method is shown. In the figure classification is shown for the three largest clusters, but as discussed above only the largest cluster is considered as representing wall. The value of the threshold τ is discussed in section 8.1.5.

8.1.3 Noise Removal

By only considering whether pixels in the blurred texture belong to the wall cluster or not, a binary image is obtained where wall regions are represented by zeros, and potential façade features are represented by ones. In figure 8.6a an example of this binary segmentation is shown.

It was discovered that noise and elements occluding the polygon surface in the input images often result in noise near the border of the polygon in this binary image. In addition, occluding elements may result in partitioning of the wall region in the binary image. The two vertical drainpipes seen in the rectified texture in figure 8.2a is an example of this, which results in partitioning of the wall region as seen in figure 8.6a. Other examples of occluding elements are lampposts, trees etc., and to disregard such elements, a flood fill of the exterior of the polygon with zeros is performed.

Noise may still be present in the binary image, however, and to prevent over-segmentation morphology is applied. Specifically the image is closed, to fill holes of the non-wall regions, and then opened, to remove noise in the wall regions. Closing and opening is performed using a simple square structuring element of size 5×5 pixels. For more details on these morphology operations see [30]. In figure 8.6b the binary segmentation resulting from noise removal as explained in this section is shown.

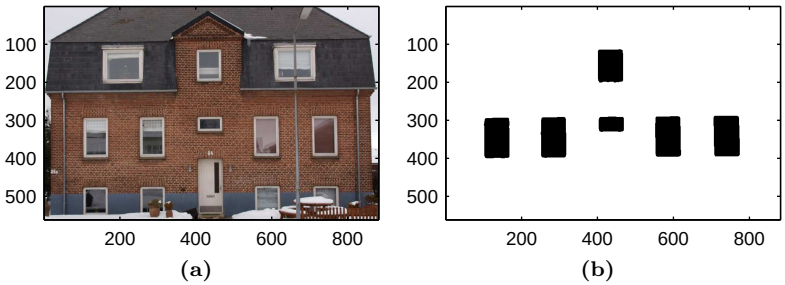


Figure 8.7: An example of façade segmentation. a) Rectified texture for the façade polygon. Note that the top left and right windows are not part of the façade polygon. b) The resulting set of BLOBs representing detected regions of similar depth.

8.1.4 BLOB Analysis

The binary segmentation obtained can be interpreted as a set of BLOBs[1] that represent potential façade features. Hopefully the detected BLOBs include the recessed windows and doors that are sought, but other features may also be present. E.g. in figure 8.6b there is a small BLOB representing the number of the house. In order to remove any BLOBs that are unlikely to represent recessed windows and doors, BLOB analysis [30] is employed. In this project a simple filtering is performed by discarding BLOBs with an area below some threshold, but other factors such as compactness, circularity, aspect ratio, position etc. may be taken into account to further minimize the risk of false positives.

The final result of façade segmentation is the set of BLOBs that are not discarded, and these BLOBs represent regions of similar depth in the façade which are likely to represent recessed windows and doors.

8.1.5 Evaluation

In figure 8.7, the result of applying the façade segmentation method developed in the above sections is shown for another façade polygon. It is seen that six of the recessed windows in the façade are correctly identified, but the door and the bottom four windows are not detected. The reason that segmentation fails for these features is partly that they are occluded by other elements, and partly that they extend beyond the border of the polygon, and thus are removed by the flood fill during noise removal. Thus for the developed algorithm to correctly detect windows and doors, they must be completely inside the façade polygon and not be occluded by other elements.

In the following the choice of the different parameters for the façade segmentation method is discussed. Although there are lots of possibilities for fine-tuning these parameters, during development a set of parameters which lead to good results for the majority of data sets used in this project was determined.

In section 8.1.1 a Gaussian blur is applied to the rectified texture to remove high frequency elements such as joints between bricks. The amount of blur is determined by the standard deviation σ, and for the data sets used in this project, the value $\sigma = 2.2$ pixels performs well. This is a trade-off between

[1] Binary Large OBject (BLOB).

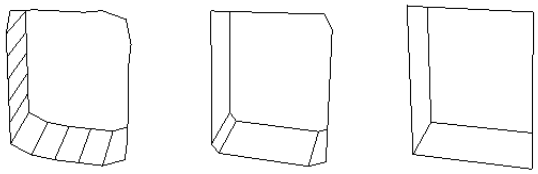


Figure 8.8: Different approaches for approximating region contours. Wireframe rendering of a recessed window with exaggerated depth is used for illustration. a) Contour sampling. b) Polygon approximation. c) Minimum-area rectangle.

smoothing to obtain similarly colored wall regions, and keeping enough details to accurately determine the boundary of other façade regions.

For clustering and classification, which was treated in section 8.1.2, the results are determined by the size S of the random sample of HSV values, and the cutoff threshold c. The value chosen for S influences the choice of c, and vice versa, because e.g. decreasing the sample size S likely increases the distance between the sampled HSV values representing colors of the same region of the façade. For the data sets used in this project, the values $S = 1000$ and $c = 0.025$ were chosen. Here c is expressed as Euclidean distance in the HSV color space spanned by a unit cube, see figure 8.5a.

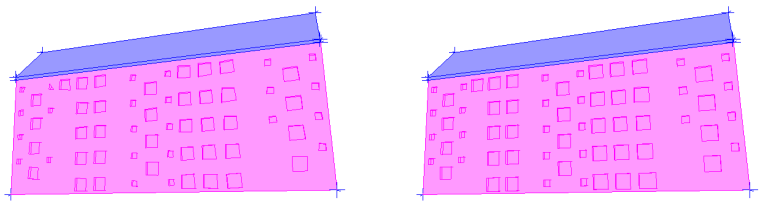
For classification of pixels in the blurred texture as either wall or not wall, the Mahalanobis distance r from the HSV value of the pixel to the wall cluster is used. Pixels for which $r < \tau$ are classified as representing wall, and for the tested data sets the value $\tau = 6$ performs well.

There is a potential pitfall regarding clustering and classification based on HSV values. In the HSV color space, the hue values are circular, see figure 8.4b, but the clustering is performed without taking this into consideration. Even though this was not a problem for the data sets used, the consequence is that hues which are only slightly different may end up in separate clusters. Therefore a robust solution should take this into account.

8.2 Finding Region Contours

Each of the BLOBs resulting from façade segmentation is represented as a subset of pixels in a binary image. For refining the coarse model, however, it is necessary to transform the BLOBs into polygons that represent the contour of the detected façade regions. Three approaches for approximating the region contours have been investigated. These are are illustrated in figure 8.8 and analyzed in the following.

Contour Sampling For each BLOB a sequence of coordinates for the pixels defining the outline of the BLOB is extracted. Forming a polygon from the entire sequence results in a region contour with excessively high resolution, and to obtain a simpler polygon only some of the pixel coordinates are used. That is, the polygon defining the region contour is obtained by sampling e.g. every



(a) (b)

Figure 8.9: Comparison of two approaches for finding region contours. a) Polygon approximation. b) Minimum-area rectangle.

10^{th} pixel coordinate in the sequence. An example of the result of this approach is shown in figure 8.8a.

Polygon Approximation As seen in figure 8.8a the polygon obtained from contour sampling represents the nearly straight sides of the region using several vertices. By applying a polygon approximation technique such as the Douglas-Peucker algorithm [52] to the sequence of coordinates, a simpler polygon is obtained. This algorithm finds the simplest polygon that approximates the original within a specified tolerance. An example of employing the Douglas-Peucker algorithm is shown in figure 8.8b.

Minimum-Area Rectangle As discussed above, the purpose of façade segmentation is to identify regions representing recessed windows and doors. Based on the observation that most windows and doors are rectangular, an approach that results in an even simpler region contour is to identify the oriented rectangle with minimum area that encloses the BLOB. An example of using the minimum-area rectangle [48] for the region contour is shown in figure 8.8c.

Even though the minimum-area rectangle may not approximate the shape of the BLOB as closely as the alternative approaches, the results obtained using this approach are significantly better for the rectangular façade features that are of interest here, see figure 8.9 for a comparison. As seen in the figure, the region contours obtained using minimum-area rectangles are oriented more accurately, and they are rectangular as the real windows of the building. Therefore it is proposed to use minimum-area rectangles for defining the contours of the detected façade regions.

The final result of finding region contours is that all BLOBs from façade segmentation now are represented by their minimum-area rectangles. Each of these oriented rectangles are described by its centroid, an angle relative to the x-axis of the rectified texture, and the dimensions of the rectangle. Further evaluation of the method developed in this section postponed to section 8.3.1.

8.3 Refining the Model

The final step of automatic façade reconstruction is to update the model with a refined version of the original façade polygon in the coarse model. This refined façade has recessed regions where windows and doors have been detected, and

the reconstruction is achieved by cutting the contours defined by the oriented rectangles from the last step out of the original façade polygon, and then inserting new polygons representing the recessed parts of the façade. This process is explained in the following.

Cutting Region Contours As mentioned in section 7.1.3, during coarse model reconstruction it is not enforced that polygons of the model are planar. However, for the reconstruction method proposed here this condition must be met. Therefore the outer contour polygon of the reconstructed façade is defined by the coplanar 3D coordinates $\mathbf{V}_i$ found as explained in section 7.2.4.

In order to cut out the contours defined by the oriented rectangles, for each of the rectangles four vertices $\mathbf{W}_j$ defining the corners are needed, and these vertices must be coplanar with the vertices $\mathbf{V}_i$. Denote by the homogeneous 3-vectors $\mathbf{w}_j$ the pixel coordinates of the rectangle corners in the rectified texture. Then by applying the inverse transformation T^{-1}, where T is the transformation applied in (7.9), the coordinates $\hat{\mathbf{w}}_j$ in the plane are obtained. That is

$$\hat{\mathbf{w}}_j = \mathrm{T}^{-1}\mathbf{w}_j. \tag{8.4}$$

Normalizing such that $\hat{\mathbf{w}}_j = [\hat{x}_j \ \hat{y}_j \ 1]^\mathsf{T}$, then the 3D coordinates $\mathbf{W}_j$ are computed as

$$\mathbf{W}_j = \mathbf{m} + \hat{x}_j\mathbf{u}_x + \hat{y}_j\mathbf{u}_y, \tag{8.5}$$

where $\mathbf{m}$ is the centroid of the polygon, and $\mathbf{u}_x$ and $\mathbf{u}_y$ are the axes of the plane computed as explained in sections 7.2.1 and 7.2.2 respectively.

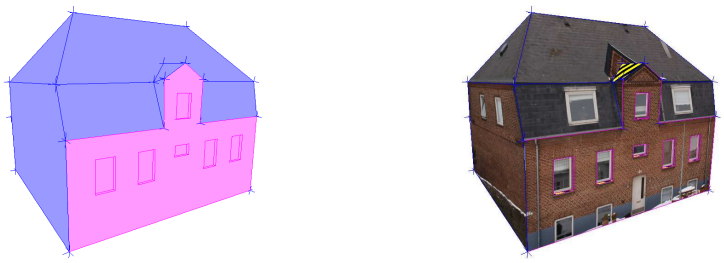
In this project, the tessellation capabilities of GLUT[2] are utilized for cutting the contours for all oriented rectangles out of the outer contour polygon of the reconstructed façade. The result of this process is a new façade polygon, which has rectangular holes for all detected windows and doors.

Extruding Detected Regions The last step of reconstructing the façade is to create new polygons that represent the recessed windows and doors, and this is achieved using a simple extrusion. A new polygon is created for each of the contours defined by the oriented rectangles. This polygon is parallel with the façade polygon obtained in the previous step, but it is slightly offset along the negative direction of the plane normal, such that it represents the surface of the recessed window or door. In this project, a predefined offset is used for simplicity, see the discussion in the following section. In addition to the recessed polygon, four polygons representing the edges of the wall surrounding the recessed window or door are created. Textures for the new polygons are extracted from the input images using the same method as for polygons of the coarse model, see section 7.2.

8.3.1 Evaluation

An example of applying the developed method for automatic façade reconstruction was shown in figure 3.4 on page 28, and in figure 8.10 another example of a refined model is shown. As is seen from both figures, the detected regions correctly correspond to recessed windows of the two buildings. In the model

[2]The OpenGL Utility Toolkit.



(a) (b)

Figure 8.10: An example of a coarse model that has been refined using the proposed method for automatic façade reconstruction. The user has marked a single polygon (magenta) in the coarse model, for which automatic reconstruction is performed. a) Geometric model. b) Textured model.

of figure 8.10, however, the bottom windows and the door is not detected, as discussed in section 8.1.5.

As mentioned above, in this project a predefined offset is used for extrusion of the recessed regions of the façade. This offset is specified by the user and for the refined model to appear correctly, the offset must correspond to the depth of the recessed façade regions of the building. For façade reconstruction to be fully automatic, however, this offset should be automatically estimated for each of the detected regions.

Although the method for finding region contours proposed in section 8.2 performs well for rectangular façade features, the set of BLOBs resulting from façade segmentation may represent other shapes as well. By detecting whether a BLOB closely resembles a rectangle or not, it may be possible to switch between finding region contours using polygon approximation and minimum-area rectangles to obtain good results for both rectangles and other polygonal façade features.

8.4 Conclusion

In this chapter, the last step of the proposed reconstruction method, namely automatic façade reconstruction, has been treated, and a novel method for automatically refining a coarse model by adding façade details such as recessed windows and doors to selected polygons has been developed. In section 8.1 it was deduced that recessed windows and doors can be identified from the appearance of the façade, and a method for segmentation based on image processing of the rectified textures obtained for façade polygons was proposed. The result of this segmentation is a set of BLOBs representing recessed regions of the façade, and in section 8.2 it was proposed to approximate the contours of these regions by the minimum-area rectangles enclosing the BLOBs. Finally, in section 8.3 a method for refining the façades of the coarse model, based on extrusion of the detected regions, was proposed. This chapter concludes the method part of the report.

Part III

Results and Discussion

Chapter 9

System Evaluation

In this part of the report, the results of testing the developed proof of concept system are documented and discussed, and the overall conclusion of the project is provided. This chapter serves as an introduction to the following three chapters, which cover the results obtained for the three main parts of the system, namely recovery of structure and motion, coarse model reconstruction, and automatic façade reconstruction. The purpose of testing is to evaluate the developed system with respect to the problem formulation specified in chapter 3.

The proof of concept system is implemented as two separate applications, autosm and builder. The console application autosm implements automatic recovery of structure and motion from a set of input images as documented in chapters 4 through 6, and builder implements user assisted coarse model reconstruction and automatic façade reconstruction based on the results from autosm as documented in chapters 7 and 8 respectively.

9.1 Test Data

In total 11 data sets were used for evaluating the system. Each of the data sets consists of an unordered set of images of a building captured from different viewpoints. In figure 9.1 all images in the data set hadsundvej-a are shown as an example. The number of images in the data sets vary from 6 to 10.

All images in the data sets have been downscaled to 1280×960 pixels. This resolution was chosen as a balance between visual quality of the reconstructed models, computational complexity, the number of keypoints detected in the images, and the ability to accurately segment façade textures. The images contain the EXIF data that was stored with the image by the camera when capturing, see appendix A for an example. In addition, the images have been corrected for lens distortion before being supplied to the system, cf. the discussion in section 4.1.

In the following three chapters, the results of testing the developed system with the images of these data sets as input are documented and discussed. Finally, chapter 13 contains the project conclusion and perspectvies.

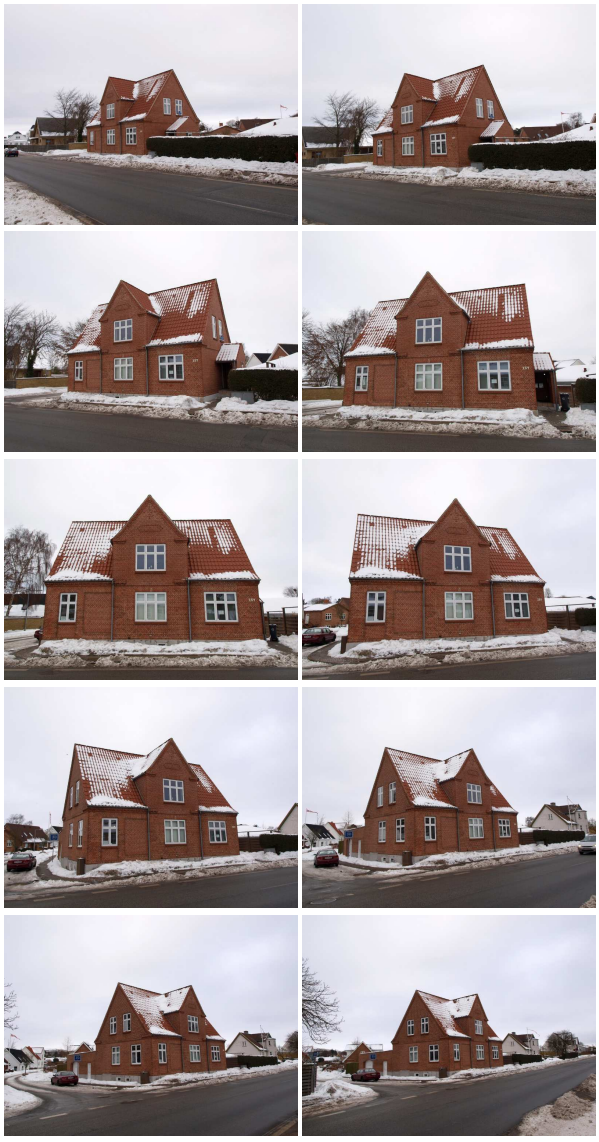


Figure 9.1: All 10 images in the data set hadsundvej-a.

Chapter 10

Structure from Motion

In this chapter the results of recovering structure and motion for the data sets are documented and discussed. The results were obtained using autosm, and the process corresponds to the preprocessing, matching, and clustering steps of the proposed reconstruction method, see figure 3.1.

10.1 Results

In table 10.1 the results of the preprocessing step are shown. The table lists the number of images in each of the data sets, and the average number of keypoints detected in the images. The results of estimating the intrinsic calibration parameters for the images were discussed in section 4.2.

The results of the matching step are shown in table 10.2. The total number of image pairs in a data set equals the binomial coefficient $\binom{m}{2}$, where m is the number of images in the data set. For an image pair to be considered an acceptable match, the number of keypoint inliers must be at least 100, as discussed in section 5.3.3.

Finally, in table 10.3 the results of the clustering step for each of the data sets are listed, and in figure 10.1 the recovered structure and motion is shown for four of the data sets.

10.2 Discussion

The preprocessing, matching, and clustering steps were evaluated in the respective chapters, and here focus is on the overall results obtained.

From table 10.1 it is seen that in general the number of keypoints detected in the input images is high. The number of detected keypoints depends on the contents of the image, and for the images used in this project, the high numbers are partly due to the joints of the brick walls. However, as seen in table 10.2 the average number of keypoint matches between images is significantly lower than the number of keypoints in each of the images. As discussed in section 5.2.1, the detected keypoints are less distinctive due to the repeated patterns, and therefore a conservative threshold has been employed for keypoint matching. By comparing the number of keypoint matches and inliers in table 10.2, it is seen that the used threshold leads to few outliers.

Data Set	Images	Keypoints
bernstorffsgade	10	6,204
bjoernoegade	7	5,839
hadsundvej-a	10	8,589
hadsundvej-b	8	4,758
hadsundvej-c	7	4,597
hadsundvej-d	7	4,398
oestrealle-a	9	7,300
oestrealle-b	6	9,419
oestrealle-c	6	6,333
riishoejsvej-a	8	5,518
riishoejsvej-b	7	7,827

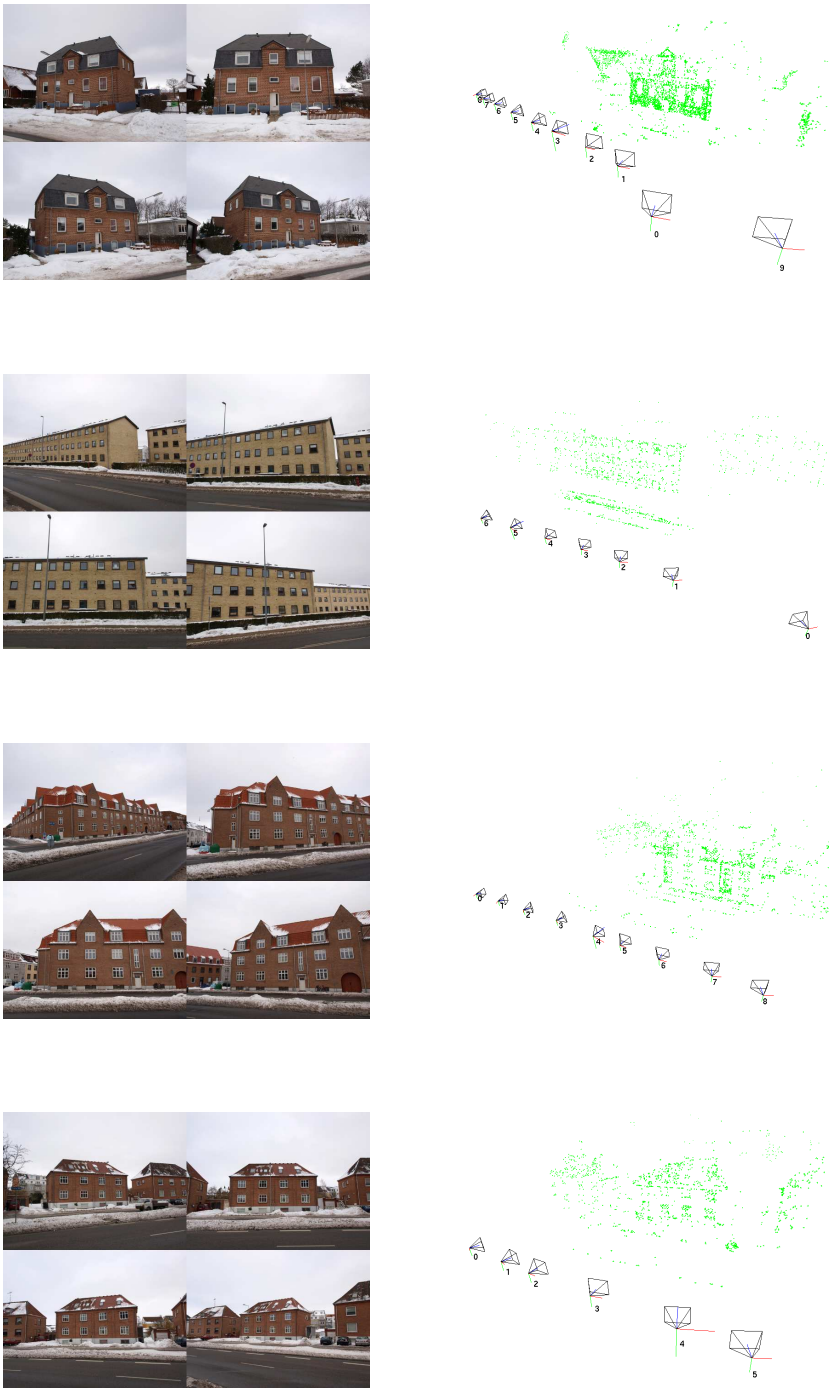
Table 10.1: Results of preprocessing. The keypoints column lists the average number of keypoints per image.

Data Set	Matching Pairs	Keypoint Matches	Keypoint Inliers
bernstorffsgade	20 (45)	440	427
bjoernoegade	10 (21)	285	250
hadsundvej-a	14 (45)	557	544
hadsundvej-b	7 (28)	200	192
hadsundvej-c	5 (21)	240	229
hadsundvej-d	12 (21)	312	304
oestrealle-a	13 (36)	340	316
oestrealle-b	6 (15)	267	258
oestrealle-c	5 (15)	374	363
riishoejsvej-a	9 (28)	189	180
riishoejsvej-b	21 (21)	960	949

Table 10.2: Results of matching. The matching pairs column lists the number of matching image pairs, and in parentheses the total number of image pairs. Keypoint matches and inliers are average numbers for all matching image pairs.

Data Set	Cameras	Points	Projections	Error
bernstorffsgade	10	4,087	9,790 (2.4)	0.20
bjoernoegade	7	875	1,786 (2.0)	0.22
hadsundvej-a	10	4,426	10,196 (2.3)	0.22
hadsundvej-b	8	821	1,752 (2.1)	0.32
hadsundvej-c	6	755	1,561 (2.1)	0.27
hadsundvej-d	7	1,690	4,103 (2.4)	0.30
oestrealle-a	9	2,238	4,989 (2.2)	0.31
oestrealle-b	6	1,091	2,279 (2.1)	0.21
oestrealle-c	6	1,397	2,993 (2.1)	0.22
riishoejsvej-a	7	870	2,180 (2.5)	0.29
riishoejsvej-b	7	3,754	12,214 (3.3)	0.22

Table 10.3: Results of clustering. In the projections column, the average number of projections per point is listed in parentheses. The error column lists the RMS reprojection error for the cluster in pixels.



(a) **(b)**

Figure 10.1: Structure from motion results for data sets bernstorffsgade, hadsundvej-d, oestrealle-a, and oestrealle-c (from the top). a) Four of the input images. b) Recovered structure and motion.

Comparing the number of images in each data set, see table 10.1, with the number of cameras in the obtained cluster, see table 10.3, it is seen that for most data sets the clustering algorithm successfully clusters all input images. For hadsundvej-c and riishoejsvej-a, however, one input image is missing in the clusters. The maximum number of keypoint matches found for image pairs including these images are 106 and 84 respectively, and in both cases this leads to less than 100 keypoint inliers as is required for the image pairs to be considered an acceptable match.

Lowering this threshold to 70 keypoint inliers, for hadsundvej-c the missing image is also clustered leading to 800 points and a slightly larger RMS reprojection error of 0.28 pixels. With the lower threshold the missing image is also clustered for riishoejsvej-a, leading to 754 points and a slightly smaller RMS reprojection error of 0.28 pixels. From this it is seen how adding an image may influence the number of points in the cluster. For hadsundvej-c the number of points increases, while for riishoejsvej-a the number of points decreases. The reprojection errors, however, are almost unchanged, and thus the accuracy of the recovered structure and motion remains.

In general the RMS reprojection errors for the points in the clusters are low, with the maximum being 0.32 pixels for hadsundvej-b as seen in table 10.3. The level of these reprojection errors indicate that structure and motion is recovered accurately. A contributory factor to this is the removal of points with reprojection errors above 2.5 pixels during optimization, as discussed in section 6.5.1.

Although removing points with high reprojection errors improves robustness, using a low threshold may lead to removal of points that are seen in more images. Points that are observed in more than two images are more constrained, and are thus likely to have higher reprojection errors. To ensure correctness, however, points that are observed in more than two images are necessary, because otherwise errors may accumulate during clustering. The projections column in table 10.3 lists the average number of projections per point in parentheses, and the average value for all data sets is 2.3. But for bjoernoegade the average projections per point is only 2.0. By allowing larger reprojection errors for this data set, more points that are observed in more than two images may be retained.

From the results and discussion in this chapter it is concluded that the methods for preprocessing, matching, and clustering developed and implemented in autosm are suitable for accurately recovering structure and motion for the data sets. However, some parameters may need additional fine-tuning in order to obtain optimum results.

Chapter 11

Coarse Model Reconstruction

In this chapter the results of user assisted reconstruction of coarse models for the data sets are documented and discussed. The results were obtained using builder as described in chapter 7, and the process corresponds to the coarse model reconstruction step of the proposed reconstruction method, see figure 3.1.

The purpose of the tests leading to the results covered in this and the previous chapter, is to evaluate the developed system regarding the first part of the problem formulation. Specifically, the developed system is evaluated with respect to reconstruction of textured polygon mesh models of real world buildings from unordered sets of images. In this chapter the focus is on the user assisted model reconstruction part.

11.1 Results

For each of the data sets a coarse model was reconstructed using builder, based on the structure and motion results treated in the previous chapter. In table 11.1 statistics for each of the coarse models are listed, and in figure 11.1 the reconstructed models are shown for six of the data sets. In addition, the coarse models for data sets hadsundvej-a and bernstorffsgade were shown in figure 3.3 and 7.9 respectively.

11.2 Discussion

The coarse model reconstruction step consists of two main parts, namely user assisted model reconstruction and texture extraction. Both of these were evaluated in chapter 7, and in the following focus is on the overall results obtained.

As seen from figure 11.1, the reconstructed coarse models consist of polygons representing large planar surfaces of the buildings. The user completely decides the level of detail, and here a minimalistic approach has been used. More details such as eaves and gutters on the roof could be added to enhance realism, but to benefit from automatic façade reconstruction the coarse models should still represent façades using few large polygons.

Data Set	Locators	Polygons
bernstorffsgade	22	12
bjoernoegade	8	3
hadsundvej-a	20	9
hadsundvej-b	20	13
hadsundvej-c	15	7
hadsundvej-d	7	2
oestrealle-a	33	21
oestrealle-b	8	3
oestrealle-c	11	4
riishoejsvej-a	9	2
riishoejsvej-b	10	4

Table 11.1: Statistics for the reconstructed coarse models.

The coarse models reconstructed for the data sets are not complete models of the buildings, e.g. the backs of the buildings are not modeled. The reason is that no images of the buildings from these viewpoints were captured for the data sets, because the primary focus of this project regarding model reconstruction is automatic refinement of façades. With images covering a whole building and properly recovered structure and motion, however, it should be possible to fully reconstruct a building using the implemented method. As illustrated by the missing roof for riishoejsvej-a in figure 11.1, it may be difficult to capture images of some parts of a building though.

For the models shown in figure 11.1, refined textures were successfully extracted for all polygons. However, for some polygons in a couple of models no suitable input images could be found. Capturing additional images of the building for problematic areas is a solution to this problem.

Occlusions from other parts of the building in the images that lead to artifacts in textures for some polygons were discussed in section 7.2.6. Often occlusions from non-modeled features are also present in the textures for a model, and the tree, which occludes the façade and roof of the building in oestrealle-b in figure 11.1, is a good example. In addition to the solutions proposed in section 7.2.6, letting the user manually select the image to use may be a solution to this problem.

Finally, the quality of the rectified textures that are extracted is limited by the available input images. The best quality is obtained form images that view the polygon at a right angle, and quality decreases with the amount of perspective distortion that must be corrected during rectification.

Based on the results and discussion in this chapter it is concluded that the method for coarse model reconstruction developed and implemented in builder can be used for reconstructing textured polygon mesh models of real world buildings.

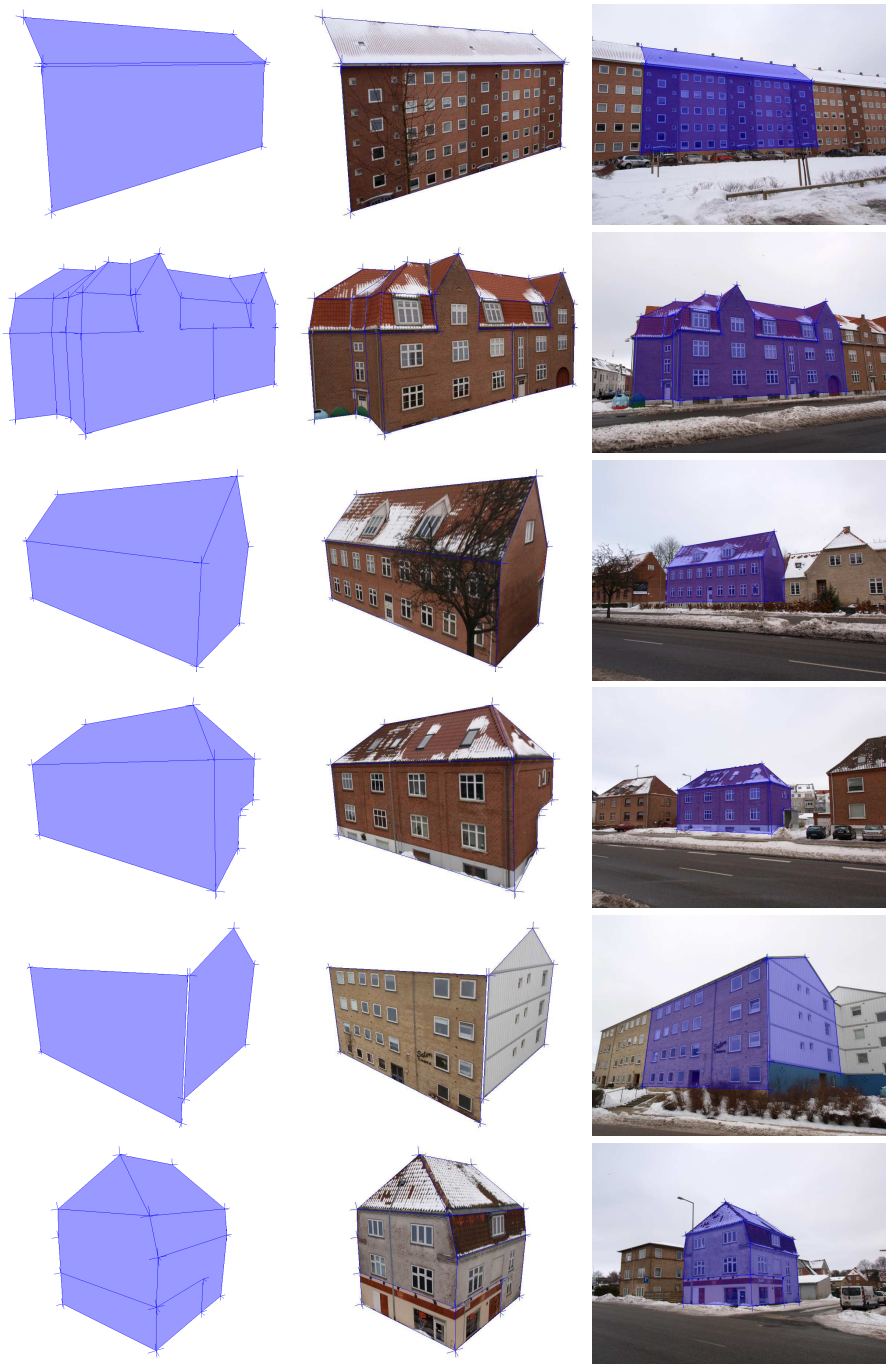


Figure 11.1: Results of coarse model reconstruction for data sets bjoernoegade, oestrealle-a, oestrealle-b, oestrealle-c, riishoejsvej-a, and hadsundvej-c (from the top). a) Geometric model. b) Textured model. c) Geometric model overlaid on image.

Chapter 12

Automatic Façade Reconstruction

In this chapter the results of automatic façade reconstruction for the data sets are documented and discussed. The results were obtained using builder, and the process corresponds to the last step of the proposed reconstruction method, see figure 3.1.

The purpose of the tests leading to the results covered in this chapter, is to evaluate the developed system regarding the last part of the problem formulation. Specifically, the developed system is evaluated with respect to automating the process of adding façade details to the models.

12.1 Results

For testing automatic façade reconstruction, in each of the reconstructed coarse models presented in the previous chapter, a subset of polygons were marked as façades for which to perform automatic refinement. In table 12.1 the number of marked façade polygons and results of the performed façade reconstruction is listed for each of the data sets.

The table lists the number of correctly detected façade features, i.e. recessed windows and doors, and the ground truth for the building. As discussed in section 8.1.5, the developed algorithm is only able to detect windows and doors that are completely inside the façade polygon, and therefore only those features are included in the ground truth value. The cases where incorrect features were detected are treated in the following discussion.

In figure 12.1 the results of automatic façade reconstruction for the coarse models are shown for six of the data sets. In addition, refined models for data sets hadsundvej-a and bernstorffsgade were shown in figure 3.4 and 8.10 respectively.

12.2 Discussion

The automatic façade reconstruction step was evaluated in chapter 8, and in the following focus is on the overall results obtained.

Data Set	Façade Polygons	Detected Features	Ground Truth
bernstorffsgade	1	6	6
bjoernoegade	1	63	82
hadsundvej-a	2	8	8
hadsundvej-b	1	2	2
hadsundvej-c	4	6	7
hadsundvej-d	1	18	21
oestrealle-a	2	12	12
oestrealle-b	1	11	17
oestrealle-c	1	8	8
riishoejsvej-a	2	24	36
riishoejsvej-b	1	8	8

Table 12.1: Results of automatic façade reconstruction. The detected features column lists the number of correctly detected features, and the ground truth column lists the true number of features completely inside the façades of the building.

From table 12.1 it is seen that for 6 of the 11 data sets, the number of correctly detected façade features equals the ground truth. For the remaining data sets, the percentage of correctly detected features is between 65% and 86%, and for all data sets on average 89% of the ground truth features were correctly detected. This is supported by figure 12.1, in which it is seen that in general most features are correctly detected.

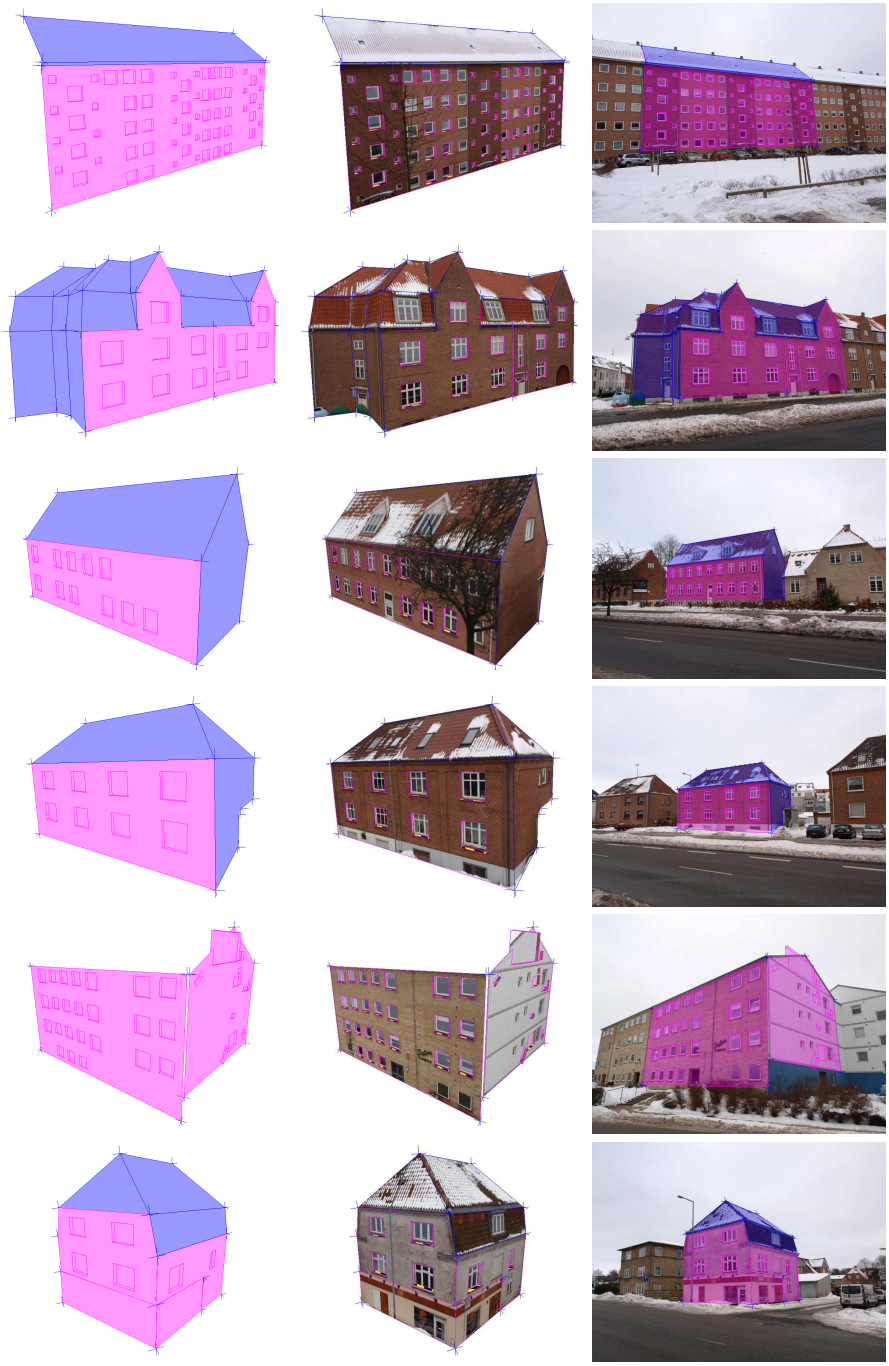
In most of the cases where not all features are detected, this is due to occlusions. Two examples of this are the trees present in bjoernoegade and oestrealle-b in figure 12.1, and in these cases the ideas previously presented for avoiding occlusions in the extracted textures likely also solve the problem of undetected façade features.

As discussed in section 8.1.5, the problem of undetected features is prevalent near the border of façade polygons, and examples of this are the missing windows in the two bottom façade polygons of hadsundvej-c, see figure 12.1. These windows touch the polygon border, and are thus discarded by the flood fill during noise removal in façade segmentation. This is also the reason that no doors were detected in any of the data sets, as all doors extend beyond the polygon borders. Properly detecting and handling this situation could improve performance of automatic façade reconstruction.

For two of the data sets, false positives were detected during automatic façade reconstruction. The worst case is the right façade polygon of riishoejsvej-a shown in figure 12.1. For that particular polygon, 11 false positives were detected and no correct features were found. It is deemed that the similarity of colors in this façade causes segmentation to fail. The other data set is oestrealle-a, where the canopy over the door to the right is incorrectly detected as a recessed façade feature.

As most of the images in the data sets capture the buildings slightly from below, textures are missing for many of the polygons representing the bottom edges of recessed windows. This problem can be solved by capturing images from more viewpoints if possible, or by manually texturing problematic areas.

In section 2.3.2 of the problem analysis, it was discussed that adding façade details such as recessed windows and doors to a reconstructed model makes it look more realistic. This was illustrated in figure 2.9 by three images of a window captured from different viewpoints. To evaluate whether the developed method



(a) (b) (c)

Figure 12.1: Results of automatic façade reconstruction for data sets bjoernoegade, oestrealle-a, oestrealle-b, oestrealle-c, riishoejsvej-a, and hadsundvej-c (from the top). Polygons marked as façades are shown in magenta. a) Geometric model. b) Textured model. c) Geometric model overlaid on image.

Figure 12.2: Three views of a reconstructed window in bernstorffsgade. The window appears flat in the coarse model (top), but in the automatically refined model (bottom), the recessed window appears correctly with the edges of the window frame properly hidden behind the wall in (a) and (c).

for automatic façade reconstruction actually leads to more realistic models, in figure 12.2 the same window is shown for both the reconstructed coarse model and the automatically refined model. Comparing this figure with the real images shown in figure 2.9, it is clear that automatic façade reconstruction makes the model look more realistic.

During model refinement, a user defined depth is used for extruding the recessed regions of the façade. As discussed in section 8.3.1, this depth must correspond to the true depth of the recessed regions of the building in order for the model to appear correctly. In the comparison above, the correct depth was used for the refined model in figure 12.2. However, in order to fully automate façade reconstruction, a method for estimating the depth of recessed regions is needed. Observing that the refined model in figure 12.2 looks more like the captured images in figure 2.9 than the coarse model does, an idea for how depths could be automatically estimated is to search for the depth which makes the refined model appear most like the input images.

Based on the results and discussion in this chapter it is concluded that the method for automatic façade reconstruction developed and implemented in builder does provide a way to automate the process of adding façade details to reconstructed models. The method fails to detect façade features at the border of polygons and in presence of occlusions, but in general most recessed windows are successfully detected and reconstructed.

Chapter 13

Conclusion

In this project the problem of reconstructing 3D models of real world buildings from unordered sets of images has been treated. The motivation for the project is that potentially real estate marketing can be improved by providing interactive visualizations, e.g. on a website, of properties for sale. In order to achieve this, however, simpler methods for 3D reconstruction are needed.

Based on the problem analysis in chapter 2, in which different methods for reconstruction were analyzed, it was deduced that using photogrammetry in combination with user assisted reconstruction was the best choice. In existing methods using this approach, the user spends most of the time adding façade details such as recessed windows and doors to the reconstructed model, and therefore it was decided to investigate how such details could be automatically added. In chapter 3 this led to the problem formulation of the project, which is repeated here:

> *Using photogrammetry and user assisted model reconstruction, how is a system for reconstructing a textured polygon mesh model of a real world building from an unordered set of images developed, and how can the process of adding façade details to the model be automated?*

In order to answer this question, an overall system concept introducing the steps of the proposed reconstruction method was developed, and in figure 13.1 the overview of these steps is shown again. In this project all five steps of the proposed reconstruction method have been treated, and a proof of concept system covering the whole reconstruction process has been designed and implemented. The primary purpose of the developed system is to demonstrate the feasibility of the proposed reconstruction method, and to investigate how existing methods can be improved, in particular with respect to automatic reconstruction of façade details.

From the results and discussion in chapters 10 through 12 it is concluded that using the developed system it is indeed possible to reconstruct textured polygon mesh models of real world buildings from unordered sets of images. Additionally, the novel method for automatic façade reconstruction that has been developed, provides an effective way to automate the process of adding façade details to the models. In the following various aspects of the project are discussed further.

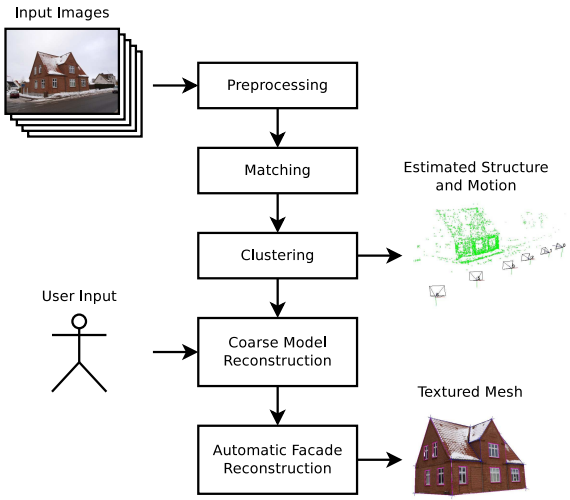


Figure 13.1: Overview of the steps in the proposed reconstruction method which are all implemented in the developed proof of concept system.

13.1 Discussion

The proposed method for automatic façade reconstruction is the primary contribution in the project, and from the obtained results it is clear that it provides a simple and effective method for adding recessed windows to reconstructed coarse models. In the discussion of the results it was concluded that the realism of the models in fact was increased by the automatic reconstruction of recessed windows. However, some work remains for the method to be fully automatic. In particular the depth of recessed regions should be automatically determined, and an idea was proposed in the discussion.

Another interesting aspect of the developed method is that it may be possible to use it for reconstructing other types of building features, which can be detected from their appearance. For instance roof windows, which typically protrude from the roof, could be reconstructed automatically with only small parameter adjustments.

Based on the results, it is deemed that the developed method is able to detect and reconstruct recessed windows of most buildings similar to the ones tested. The method is based on some basic assumptions about façades, e.g. that the walls are similarly colored and that windows are rectangular, and when these are satisfied, the developed method leads to a reduction of the time spent during model refinement compared to other user assisted reconstruction methods. Even if not all windows of a façade are detected, the method may still speed up the reconstruction process. An idea for further development is that the detected windows may be used for interactively guiding the user to manually place any remaining windows.

In the above discussion it is argued that the developed method for automatic façade reconstruction reduces the level of user interaction during refinement of the reconstructed model. In this project a simple approach to user assisted reconstruction of the coarse model has been taken, however. An overall better

solution for reconstruction would be to use e.g. the method from [38] for user assisted reconstruction of a coarse model, and then apply the developed method for automatic façade reconstruction to selected polygons of the model.

The overall outcome of this project is gained experience with 3D reconstruction of buildings from images. The developed proof of concept system has proved useful for reconstructing textured polygon mesh models of real world buildings, and it is believed that the developed method for automatic façade reconstruction leads to an improvement compared to existing reconstruction methods, as it reduces the time spent on model refinement.

13.2 Perspectives

As mentioned in the problem analysis, the ultimate goal regarding 3D reconstruction of buildings is a system allowing any real estate agent to easily reconstruct 3D models of buildings for interactive visualization. As is the case for existing reconstruction methods, the proof of concept system developed in this project is far from reaching this goal.

In particular substantial work on the user interface and the general flow of the reconstruction process is necessary. Additionally, lots of parameters in the system may need adjustments to work in other scenarios, and to achieve a truly easy to use system, these parameters must be determined automatically. The issues presented here are potential areas for further research.

The general problem of 3D reconstruction from images is an interesting subject, and it is the hope that this project can contribute to the further development of this field.

Bibliography

[1] Autodesk, Inc. website. http://usa.autodesk.com/.

[2] EDC-gruppen A/S website. http://www.edc.dk/.

[3] Holomatix Ltd. website. http://www.holomatix.com/.

[4] Microsoft Photosynth website. http://photosynth.net/.

[5] New Dimension Systems Co., Ltd. website. http://www.newdimchina.com/expertise/reverse_engineering.html.

[6] NextEngine website. http://www.nextengine.com/.

[7] Olympus Europa website. http://www.olympus-europa.com/.

[8] Rapidform, Inc. website. http://www.rapidform.com/.

[9] SolidWorks Corp. website. http://www.solidworks.com/.

[10] Telmore A/S website. http://www.telmore.dk/.

[11] Sameer Agarwal, Noah Snavely, Ian Simon, Steven M. Seitz, and Richard Szeliski. Building Rome in a Day. *International Conference on Computer Vision*, 2009.

[12] Ákos Balázs, Michael Guthe, and Reinhard Klein. Efficient trimmed NURBS tessellation. *Journal of WSCG*, 12(1):27–33, 2004.

[13] Johannes Bauer, Niko Sünderhauf, and Peter Protzel. Comparing several implementations of two recently published feature detectors. 2007.

[14] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. SURF: Speeded Up Robust Features. *Computer Vision and Image Understanding (CVIU)*, 110(3):346–359, 2008.

[15] Fausto Bernardini, Joshua Mittleman, Holly Rushmeier, Cláudio Silva, and Gabriel Taubin. The Ball-Pivoting Algorithm for Surface Reconstruction. *IEEE Transactions on Visualization and Computer Graphics*, 5(4):349–359, 1999.

[16] Fausto Bernardini and Holly Rushmeier. The 3D Model Acquisition Pipeline. *Computer Graphics Forum*, 21(2):149–172, 2002.

[17] Jean-Yves Bouguet. Camera Calibration Toolbox for Matlab, 2008. http://www.vision.caltech.edu/bouguetj/calib_doc/.

[18] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, second edition, 2001.

[19] S. Cornou, M. Dhome, and P. Sayd. Architectural Reconstruction with Multiple Views and Geometric Constraints. 2003.

[20] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification*. Wiley-Interscience, second edition, 2001.

[21] Martin A. Fischler and Robert C. Bolles. Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[22] Leica Geosystems. Leica ScanStation 2 on eBay. `http://shop.ebay.com/230398405010`.

[23] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521540518, second edition, 2004.

[24] Richard I. Hartley. In Defence of the 8-point Algorithm. *Computer Vision, IEEE International Conference on*, page 1064, 1995.

[25] William Lorensen and Harvey Cline. Marching cubes: A high resolution 3D surface construction algorithm. *ACM SIGGRAPH Computer Graphics*, 21(4):163–169, 1987.

[26] Manolis I. A. Lourakis and Antonis A. Argyros. SBA: A Software Package for Generic Sparse Bundle Adjustment. *ACM Transactions of Mathematical Software*, 36(1):Article 2, 2009.

[27] David G. Lowe. Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[28] M. Vergauwen F. Verbiest K. Cornelis J. Tops R. Koch M. Pollefeys, L. Van Gool. Visual modeling with a hand-held camera. *International Journal of Computer Vision*, 53(3):207–232, 2004.

[29] Yazid Malek. Choosing The Best Profile In SolidWorks, 2009. `http://www.yamatot.yama-designing.com/choosing-the-best-profile-in-solidworks/`.

[30] Thomas B. Moeslund. *Image and Video Processing*. First edition, 2008.

[31] David Nistér. An Efficient Solution to the Five-Point Relative Pose Problem. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 26(6):756–770, 2004.

[32] A. Akbarzadeh B. Clipp C. Engels D. Gallup P. Merrell C. Salmi S. Sinha B. Talton L. Wang Q. Yang H. Stewénius H. Towles G. Welch R. Yang M. Pollefeys P. Mordohai, J.-M. Frahm and D. Nistér. Real-Time Video-Based Reconstruction of Urban Environments. *3D-ARCH'2007: 3D Virtual Reconstruction and Visualization of Complex Architectures*, 2007.

[33] Q. Pan, G. Reitmayr, and T. Drummond. ProFORMA: Probabilistic Feature-based On-line Rapid Model Acquisition. *Proc. 20th British Machine Vision Conference (BMVC)*, 2009.

[34] Marc Pollefeys. Seven-point algorithm, 2002. `http://www.cs.unc.edu/~marc/tutorial/node55.html`.

[35] Marc Pollefeys, Reinhard Koch, and Luc Van Gool. Self-Calibration and Metric Reconstruction in spite of Varying and Unknown Internal Camera Parameters. *Sixth International Conference on Computer Vision*, pages 90–95, 1998.

[36] Marc Pollefeys and Luc Van Gool. From Images to 3D Models. *Communications of the ACM*, 45(7):50–55, 2002.

[37] Christopher Schwartz and Reinhard Klein. Improving Initial Estimations for Structure from Motion Methods. *The 13th Central European Seminar on Computer Graphics (CESCG 2009)*, 2009.

[38] Sudipta N. Sinha, Drew Steedly, Richard Szeliski, Maneesh Agrawala, and Marc Pollefeys. Interactive 3D Architectural Modeling from Unordered Photo Collections. *SIGGRAPH Asia 2008*, 2008.

[39] Christoffer Valgren and Achim J. Lilienthal. SIFT, SURF and seasons: Appearance-based long-term localization in outdoor environments. *Robotics and Autonomous Systems*, 58(2):149–156, 2010.

[40] Wikipedia. 135 film, 2010. `http://en.wikipedia.org/wiki/135_film`.

[41] Wikipedia. 3D scanner, 2010. `http://en.wikipedia.org/wiki/3D_scanner`.

[42] Wikipedia. Computer-aided design, 2010. `http://en.wikipedia.org/wiki/Computer-aided_design`.

[43] Wikipedia. Constructive solid geometry, 2010. `http://en.wikipedia.org/wiki/Constructive_Solid_Geometry`.

[44] Wikipedia. Crop factor, 2010. `http://en.wikipedia.org/wiki/Crop_factor`.

[45] Wikipedia. Exchangeable image file format, 2010. `http://en.wikipedia.org/wiki/Exchangeable_image_file_format`.

[46] Wikipedia. Gaussian blur, 2010. `http://en.wikipedia.org/wiki/Gaussian_blur`.

[47] Wikipedia. HSL and HSV, 2010. `http://en.wikipedia.org/wiki/HSL_and_HSV`.

[48] Wikipedia. Minimum bounding box algorithms, 2010. `http://en.wikipedia.org/wiki/Minimum_bounding_box_algorithms`.

[49] Wikipedia. Non-uniform rational B-spline, 2010. `http://en.wikipedia.org/wiki/Non-uniform_rational_B-spline`.

[50] Wikipedia. Photogrammetry, 2010. `http://en.wikipedia.org/wiki/Photogrammetry`.

[51] Wikipedia. Polygon mesh, 2010. http://en.wikipedia.org/wiki/Polygon_mesh.

[52] Wikipedia. Ramer–Douglas–Peucker algorithm, 2010. http://en.wikipedia.org/wiki/Ramer-Douglas-Peucker_algorithm.

[53] Wikipedia. Range imaging, 2010. http://en.wikipedia.org/wiki/Range_imaging.

[54] Wikipedia. Structured-light 3D scanner, 2010. http://en.wikipedia.org/wiki/Structured-light_3D_scanner.

[55] Song Zhang and Peisen Huang. High-Resolution, Real-time 3D Shape Acquisition. *Optical Engineering*, 45(12), 2006.

Part IV

Appendices

Appendix A

Exchangeable Image File Format (EXIF)

EXIF is a standard for storing metadata in images files, and it is most commonly used in image files using JPEG compression. The format is part of the Design rule for Camera File system (DCF) standard created by JEITA to encourage interoperability between imaging devices [45]. When capturing images, most digital cameras store additional information using EXIF, and the stored information includes camera settings such as aperture, shutter speed, and focal length.

Below follows the output of the program jhead, which supports listing and modification of EXIF data in image files, for an image in one of the data sets used in this project.

```
File name    : hadsundvej-a0.jpg
File size    : 306629 bytes
File date    : 2010:05:14 19:15:53
Camera make  : OLYMPUS IMAGING CORP.
Camera model : E-520
Date/Time    : 2010:02:12 12:24:09
Resolution   : 1280 x 960
Flash used   : No (auto)
Focal length : 14.0mm
Exposure time: 0.017 s  (1/60)
Aperture     : f/10.0
ISO equiv.   : 100
Whitebalance : Auto
Metering Mode: matrix
Exposure     : aperture priority (semi-auto)
======= IPTC data: =======
Record vers. : 19990
Caption      : OLYMPUS DIGITAL CAMERA
```

Appendix B

Least-Squares Solution of Homogeneous Equations

A problem frequently occurring in this project is that of solving a set of homogeneous equations in the least-squares sense. That is solving a system of equations on the form $\mathrm{A}\mathbf{x} = \mathbf{0}$, where A is an $m \times n$ matrix. If $\mathbf{x}$ is a solution to the set of equations, so is $k\mathbf{x}$ for any scalar k. The trivial solution $\mathbf{x} = \mathbf{0}$ is not of interest, and therefore the additional constraint that $||\mathbf{x}|| = 1$ is added.

An exact solution to such system of equations exists only if the matrix is rank deficient, i.e. $\mathrm{rank(A)} < n$, which is generally not the case. Therefore the least-squares solution is sought, and the problem may be stated as follows:

> Given an $m \times n$ matrix A with $m \geq n$, find $\mathbf{x}$ which minimizes $||\mathrm{A}\mathbf{x}||$ subject to $||\mathbf{x}|| = 1$.

This problem can be solved using Singular Value Decomposition (SVD), and the method is described in [23].

The SVD of matrix A is the factorization $\mathrm{A} = \mathrm{UDV}^\mathsf{T}$, where U and V are orthogonal matrices, and D is a diagonal matrix with non-negative entries. The factorization is normally carried out such that the diagonal entries of D are in descending order. In most applications, the matrix A has at least the same number of rows as columns, i.e. $m \geq n$. In this case U is an $m \times n$ matrix with orthogonal columns, D is an $n \times n$ diagonal matrix, and V is an $n \times n$ orthogonal matrix. Additionally U has the norm-preserving property that $||\mathrm{U}\mathbf{x}|| = ||\mathbf{x}||$.

Returning to the above problem, let $\mathrm{A} = \mathrm{UDV}^\mathsf{T}$. The problem now involves minimizing $||\mathrm{UDV}^\mathsf{T}\mathbf{x}||$. Observe that $||\mathrm{UDV}^\mathsf{T}\mathbf{x}|| = ||\mathrm{DV}^\mathsf{T}\mathbf{x}||$, and $||\mathbf{x}|| = ||\mathrm{V}^\mathsf{T}\mathbf{x}||$. Hence the problem is to minimize $||\mathrm{DV}^\mathsf{T}\mathbf{x}||$ subject to the condition $||\mathrm{V}^\mathsf{T}\mathbf{x}|| = 1$. Now by writing $\mathbf{y} = \mathrm{V}^\mathsf{T}\mathbf{x}$, the problem reduces to minimizing $||\mathrm{D}\mathbf{y}||$ subject to $||\mathbf{y}|| = 1$. As D is a diagonal matrix with its entries in descending order, it immdeiately follows that the solution to this problem is $\mathbf{y} = [0, 0, \ldots, 1]^\mathsf{T}$. Finally, $\mathbf{x} = \mathrm{V}\mathbf{y}$ is simply the last column of V. That is, the solution to the problem stated above is as follows:

> The $\mathbf{x}$, which minimizes $||\mathrm{A}\mathbf{x}||$ subject to $||\mathbf{x}|| = 1$, is the last column of V, where $\mathrm{A} = \mathrm{UDV}^\mathsf{T}$ is the SVD of A.