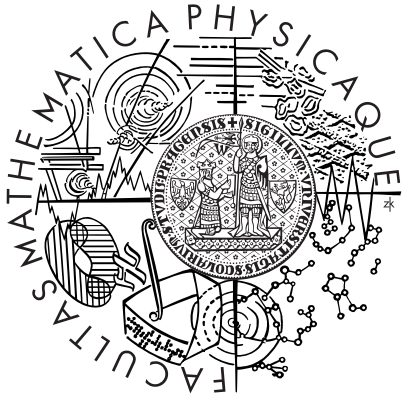
Charles University in Prague
Faculty of Mathematics and Physics

BACHELOR THESIS



Lukáš Mach

Systém pro polo-automatickou 3D rekonstrukci scén

Semi-automatic system for reconstruction of 3D scenes

Department of Software and Computer Science Education

Supervisor: Ing. Jan Buriánek,
Study programme: General Computer Science

2009

I would like to thank my supervisor Ing. Jan Buriánek for his advice and the inspiring atmosphere he created during the DLM project. My thanks also belong to Mgr. Martin Mareš, PhD., for great technical advice he gave me as the advisor of the individual software project, which became part of this thesis.

I'd like to thank Visual Connection, a.s. and The Museum of the City of Prague for the opportunity to participate in a unique digitization project and for providing data and photographs for my work, especially to PhDr. Zuzana Strnadová, Mgr. Pavla Státníková, PhDr. Kateřina Bečková, JUDr. Kateřina Krylová and Prof. Ing. Jiří Žára, CSc. from The Museum of the City of Prague and Bc. Tomáš Petrů and Ing. Jakub Vaněk from Visual Connection, a.s.

I would like to give a special thanks to my friends and former colleagues Ing. David Sedláček, Mgr. Hedvika Peroutková, Mgr. Matěj Cáha and Ing. Zuzana Kútna. Finally, my deepest thanks belong to my family for their love and encouragement during my studies.

I declare that I wrote my bachelor thesis independently and exclusively with the use of the cited sources. I agree with lending this thesis.

In Prague, 1. 8. 2009 Lukáš Mach

Contents

Název práce: Systém pro polo-automatickou 3D rekonstrukci scén
Autor: Lukáš Mach
Katedra (stav): Kabinet software a výuky informatiky
Vedoucí bakalářská práce: Ing. Jan Buriánek
e-mail vedoucího: jabu@seznam.cz

Abstrakt: Modelování na základě fotografií umožuje vytvořit přesné 3D rekonstrukce existujících scén podle několika jejich fotografií. V této práci popisujeme nejdůležitejší věty a výsledky z projektivní geometrie a počítačového vidění nutné k provedení těchto rekonstrukcí, zkoumáme jak mohou být ke zrychlení tvorby scény použity algoritmy na získávání řídkých korespondencí a implementujeme uplné open source softwarové řešení tohoto problému.

Klíčová slova: modelování na základě fotografií, řídká korespondence, structure from motion

Title: Semi-automatic system for reconstruction of 3D scenes
Author: Lukáš Mach
Department: Department of Software and Computer Science Education
Supervisor: Ing. Jan Burinek
Supervisor's e-mail address: jabu@seznam.cz

Abstract: Image-based modeling offers a way to create precise 3D reconstructions of existing scenes based on their photographs. We describe essential results from projective geometry and computer vision necessary to perform such reconstructions, investigate how sparse feature matching algorithms can be used to help the user in creating the scene more easily and implement a fully functional open source software solution.

Keywords: image-based modeling, sparse feature matching, structure from motion

Chapter 1

Introduction

Completely automatic polygonal reconstruction of a 3D scene from a set of its photographs is generally considered to be the holy grail of computer vision. In recent years, new algorithms and new insights into the problem of matching images and computing camera calibration have been gained. However, the goal of automatically obtaining simplified polygonal model that authentically captures main features of the scene still seems to be out of reach.

On the other hand, wide availability of consumer digital cameras and the attractiveness of 3D visualizations underscore the need for software packages that would be able to create 3D models based on digital images. Accurate reconstructions can be also used to document historic artifacts or during criminal investigations[1]. Currently, there are several commercial solutions such as Autodesk ImageModeler [2] or Eos PhotoModeler [5], but they rely on heavy interaction with the user, who basically has to mark every model vertex visible on submitted photographs. When sufficient amount of point correspondences between the images has been entered, the software calculates the camera calibration, which also yields 3D positions of marked vertices. Polygonal model is then obtained by joining these vertices into polygons.

The aim of this work is to find main bottlenecks of these rather manual approaches and to investigate how image processing techniques like sparse feature matching and robust estimation can be used to ease the process of 3D reconstruction. Complete software solution, capable of full metric reconstruction with export of the 3D models into standardized format, has been created and is released under the GPL license. At this time, it is the only such package released into the public domain.

Methods and algorithms presented in this work were also applied in a large scale reconstruction project – digitization of the Langweil's model of Prague (see figures 1.1 and 1.2). During this project, the famous paper model was reconstructed from approximately 300 thousand photographs. Such amount of data clearly calls for significant automatization.

[1] Most prominently, methods of photogrammetry were used during the investigation of the JFK assassination [7].

This photo is temporarily removed.

Figure 1.1: Photo of a part of Langweil's model of Prague. Note the large number of details and small structures like chimneys. Courtesy of the City of Prague Museum. Used with permission.

This photo is temporarily removed. (Don't worry, no further figures in this document were deleted.)

Figure 1.2: A detail view of a narrow street in the model. Courtesy of the City of Prague Museum. Used with permission.

Chapter 2

Overview

In this chapter, we describe in detail the input used by software packages that carry out 3D reconstruction from images, investigate currently available software solutions, describe their user interface and the qualities of the model the user can expect. Let us begin by commenting on how image-based modeling effectively differs from the classic 3D modeling approach.

2.1 Comparison with typical modeling software

When using typical 3D modeling software solutions, artists usually construct models in 3D, for example by starting with scene primitives like spheres or cubes, merging or deforming them, applying various operators, etc. until the modeled object starts to resemble the one the artist has in mind. Currently, the industry offers wide selection of 3D modeling products and artists are thus limited virtually only by their imagination.

However, this approach often fails or becomes cumbersome when we want to precisely recreate existing scenes. The freedom the artist has becomes a disadvantage – for example, nothing prevents the artist from modeling the windows of a building a little wider than they are in reality. Perhaps if the artist had been given a photo, on which the windows can be seen from perpendicular direction, she would have gotten their proportions quite right even with traditional 3D modeling application. But we can't expect that we'll always have suitable photo from which every geometric property (all lengths, angles, ...) can be directly and easily determined.

Furthermore, if the object has been modeled in this way (e.g., by basically having the artist to *guess* its shape), it's hard to statistically evaluate achieved precision and the models can't be used in applications like car accident investigation that require extremely realistic 3D models with verifiable precision.

Final objection to classic 3D modeling approach could be that it requires skilled 3D artist. Using complex products like Maya [3] is demanding and significant amount of training is necessary. Perhaps modeling could be much easier for the user if she was guided by the photos of the object that is being constructed.

2.2 Image-based modeling workflow

When working with *image-based* modeling application, the user starts by submiting photographs of the object that is being reconstructed. She then proceeds to mark places on the images where important 3D vertices are visible. We also say that she marks the position (or *parameters*) of *projections* of these vertices. The situation is illustrated in figure 2.1.

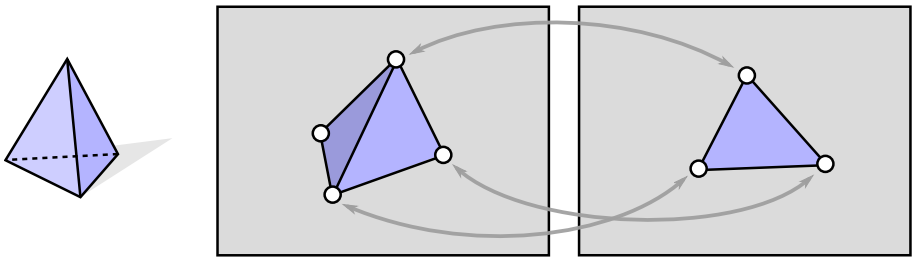


Figure 2.1: Simple scene (tetrahedron) and two views of the scene which might be used for its 3D reconstruction. The user marks points on the images where scene vertices are visible (denoted by circles). Pairs of points in different images corresponding to the same vertex are joined by gray lines.

After marking positions of every vertex on the photos, the application computes *camera calibration* – this includes positions and orientations the camera had when taking each picture and also its internal parameters (focal length, radial distortion, ...). We will give precise definition of each of these camera parameters in section 3.4, when we describe the mathematical model used to approximate real-world cameras. Refer to figure 2.2 for a simple example of a 3D scene with calibrated cameras.

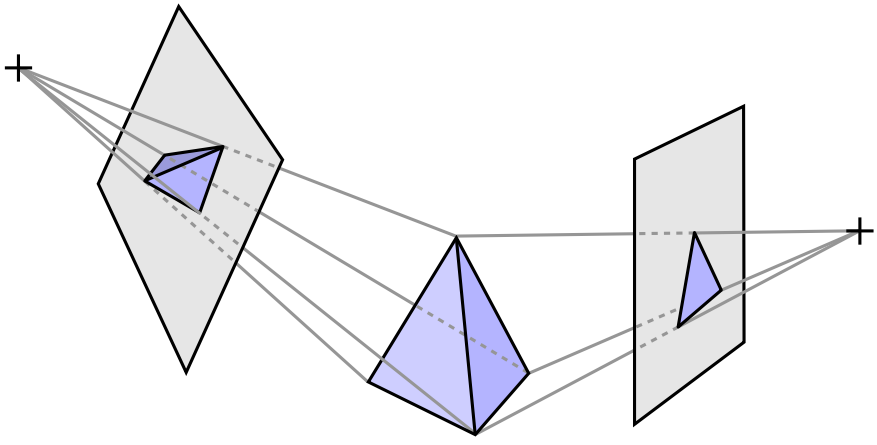


Figure 2.2: 3D scene from the previous figure with calibrated cameras. Camera centers, image planes and rays from camera centers to the vertices visible on each picture are denoted.

When we know the positions and orientations in which the camera was when taking each picture, we can estimate the position of marked vertices (if each vertex is marked on at least two calibrated images). This calculation will be refered to as *triangulation*. After that, the user can easily join the vertices into polygons. Thus, the reconstruction of the 3D polygonal model is completed. Afterwards, textures can be generated for the polygons using the images where the polygon is visible, giving the model more detailed look.

In general, geometric entities more complex than vertices can be considered. For example, the user could specify parameters of projections of second degree curves. However, we mostly restrict ourselves to 3D vertices and their projections, which will be refered to as points. It is important to note that the user never directly enters any 3D information – all she does is submitting 2D data.

2.3 User input and resulting 3D reconstruction

We can now summarise that the user input includes:

- several photos of the scenes,
- coordinates $\mathbf{x}_i^j$ of i-th vertex on j-th image; some vertices might not be visible on all photos and thus some values $\mathbf{x}_i^j$ will be unknown.

The application then uses the projections $\mathbf{x}_i^j$ to solve for camera calibration and vertex coordinates. We now might ask when, if ever, is this input sufficient to uniquely determine the scene cameras and vertices. Several issues have to be considered:

It should be clear that no absolute coordinates can be determined solely from the positions of vertex projections. We simply can't expect that we give the application photos, mark the vertices and it outputs, for example, GPS coordinates of individual vertices of the real scene.[1] The reconstruction is performed in its own coordinate frame which probably differs from the one the user might have in mind by (at least) global translation and rotation of the whole scene.

Similarly, we can come to the conclusion that the overall scale will be also undetermined: when given photos of a building, it is impossible to determine its height only from the projections – it might be ordinary several meters high building, but we might also be given photos of an extremely precise few centimeters high toy replica of the same building taken with a macro camera. These two sets of photos might be nearly undistinguishable even for a human being (assuming that the replica is really detailed and precise). The effects of rescaling the scene and positions of the cameras (and also some of their internal parameters like focal length) cancel each other and result in exactly the same input $\{\mathbf{x}_i^j\}_{i,j}$. But only

[1] Unless, of course, these quantities have been somehow explicitly specified as part of the input – some digital cameras store GPS coordinates of the position where the photo was taken in EXIF data. However, we don't consider EXIF information to be part of the input.

one output can be (deterministically) generated from a given input, leading us to the fact that the overall scale can't be uniquely calculated.

We conclude that the model reconstructed from the above input will differ from the real scene at least by a *similarity transformation* – that is, by composition of translation, rotation and isotropic scaling. In the next chapter, we'll see that given some further (but reasonable) assumptions about the cameras, the scene can really be determined up to similarity transformation – the terms *Euclidean* or *metric reconstruction* are used to refer to this fact.

Another question relating to the user input is how many photos and how many marked points (of how many vertices) are sufficient to calculate the reconstruction. One can easily see that if we mark only one vertex on all photos, the positions of the cameras are not uniquely determined and even their relative orientations can vary. Generally, it can be said that three images with 8 vertices marked on all of the photos are sufficient to reconstruct the scene, although this heavily depends on mathematical models used to represent the cameras. We leave deeper discussion to chapter 3.

What will happen when the user submits *too many* projections of too many vertices? At some point she will submit just enough projections to determine the scene reconstruction (up to an unknown similarity transformation, of course). If she adds another piece of input, one of the following will happen:

She either marks the next projection of a vertex exactly or she marks it imprecisely (for example, she marks a position that is 2.5px away from the true projection). In the former case, this addition to the original input will be compatible with the original reconstruction and it will again generate the same output. In the latter case, however, there won't be a reconstruction that precisely satisfies all constraints since an incompatible one has been added to a set of conditions that already uniquely determine the reconstruction.

Of course, it is nearly impossible to mark position of a vertex projection absolutely precisely. When using digital images, the maximum precision is usually 1px (although in specific cases subpixel precision can be obtained) and even skilled users will mark the vertices off by at least several pixels.

This situation is paradoxical, since one would assume that having more data to work with will make the computation easier. What should we do in these (often heavily) overdetermined situations? No single 3D reconstruction satisfies the entire input. What now makes sense is to try to find among all reconstructions the best one – that is, the one that minimizes some error metric. Most often, the error metric is based on statistical method of maximum likelihood estimation: from the set of all 3D reconstructions, we select the one that is the most probable given the supplied measurements $\{\mathbf{x}_i^j\}_{i,j}$.

In fact, we almost always work with overdetermined input, since this results in much more precise reconstructions. Remarkably, this approach can even be robustified (see section 3.12) so that the output is not affected if part of the input is corrupted (some of the points have been marked completely incorrectly).

Besides the projections of the scene vertices, several other types of input can also used to contrain the reconstruction. Often used are the following:

- the information that all pictures were taken with a camera with the same, but perhaps unknown, focal length (i.e., with the same "zoom"),
- the focal lengths of the cameras used to take each picture,
- affine properties of the scene – e.g., the fact that two scene planes are parallel,
- Euclidean properties of the scene – e.g., the fact that two scene planes are perpendicular.

These constraints, especially the ones concerning internal camera parameters, are of special importance since they can be used to determine scene's metric structure – this is discussed in the section 3.11.

The information about focal length of the camera used to capture a particular photograph can often be retrieved from its EXIF information. We must note however, that focal lengths from EXIF are often imprecise, although they have been used in practice, mainly as an initial guess to be further computationally optimized (for example in Microsoft's Photosynth [26]).

2.4 Currently available software

Before the release of our application, there was no open source image-based modeling tool available. There are several proprietary commercial solutions, though.

The most commonly used are Autodesk ImageModeler [2] and Eos PhotoModeler [5]. Their workflow is very similar and follows the one described in section 2.2. The user creates vertices and marks them on several images. After enough point correspondences are entered, the application computes camera calibration. Once the cameras are calibrated, the application uses this knowledge to guide the user by displaying epipolar lines – this is a line on which the projection of a vertex must lie given it's position on another image (see the section 3.7 for a detailed discussion).

Typically, the user of ImageModeler has several (2 or 4) photos displayed on her screen. A vertex is created by marking it for the first time. She then proceeds to mark the vertex on the remaining images. We argue that this workflow is not optimal – it requires the user to constantly switch her attention between different photos. Often, it can be demanding to figure out from which place the photograph was taken, especially for symmetrical scenes.

In our application, a single photo is displayed on the user's screen and she is encouraged to mark multiple vertices before proceeding to the next one, which leads to much smoother user experience.

A significant disadvantage of ImageModeler and its XML-based `.rzi` file format is that it doesn't save 2D-to-3D correspondence between the constructed

scene and its photos. The vertices estimated by ImageModeler from the image correspondences are called locators. Once they are estimated, the user can use them to create different scene primitives. However, no relation between the scene vertices and the locators from which they were created is retained. This implies that when the user decides to adjust an imprecisely positioned locator (for example by marking its position on another photo), the locator is recalculated, but the position of the vertex created from it is not adjusted. Moreover, the knowledge of where each scene vertex is visible could be valuable when generating textures for individual polygons, since the marked positions are visually more precise then mere reprojections – these two approaches to texturing are compared and implemented in [22].

Another image-based modeling software package is MetaCreations' Canoma [4] (eventually acquired by Adobe and since then discontinued). This now defunct application uses completely different and very user-friendly workflow based on the influential thesis of Paul Debevec [8]. The user interface can be seen in figure 2.3. The user creates the scene by placing various primitives like cubes or cones on it. Parameters of these primitives can be marked on the images – for example, the user can take a cube and specify where one or more of its vertices are visible (the edge incidence is also supported). Relations between the scene primitives can be entered, e.g. the user can stack up two cubes, one on the top of the other.

In this way, Canoma is able to reconstruct a scene even from a single photo. Geometric properties of the primitives are taken into account during the optimization. Thanks to that, a cube reconstructed in Canoma is indeed a perfect cube, whereas the one reconstructed by other image-based modelers will be slightly imperfect since the positions of the cube's vertices are skewed by some amount of noise. On the other hand, Canoma has problems when modeling scenes where its predefined primitives are not present.

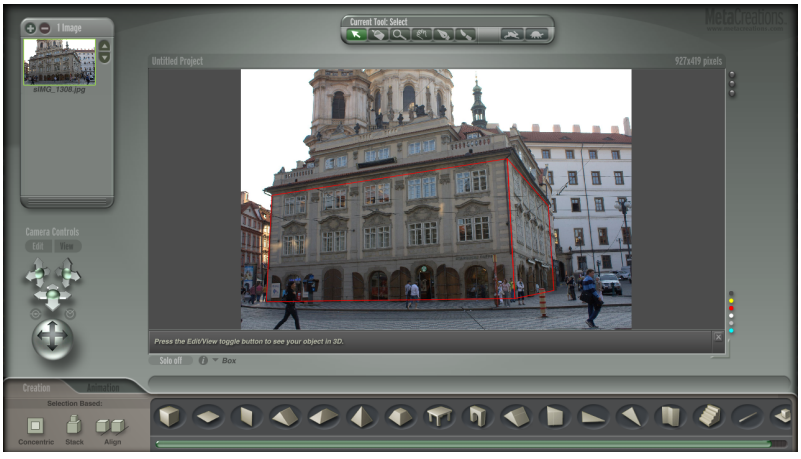


Figure 2.3: Canoma's intuitive UI with a photo and a list of primitives.

2.5 Example

When taking photos to be used for image-based reconstruction, several rules should be remebered to achieve good precision. Refer to appendix A for a list of the most important guidelines. The following input images would be ideal for reconstruction of the building on them:

Figure 2.4: Six images of a building on Lesser Town Square to be used as input for image-based modeling.

Figure 2.5 below shows untextured and textured polygonal model that can be quickly (within 10 minutes) created using the software developed as a part of this thesis. The screenshot on the left also shows automatically reconstructed 3D point cloud of the scene.

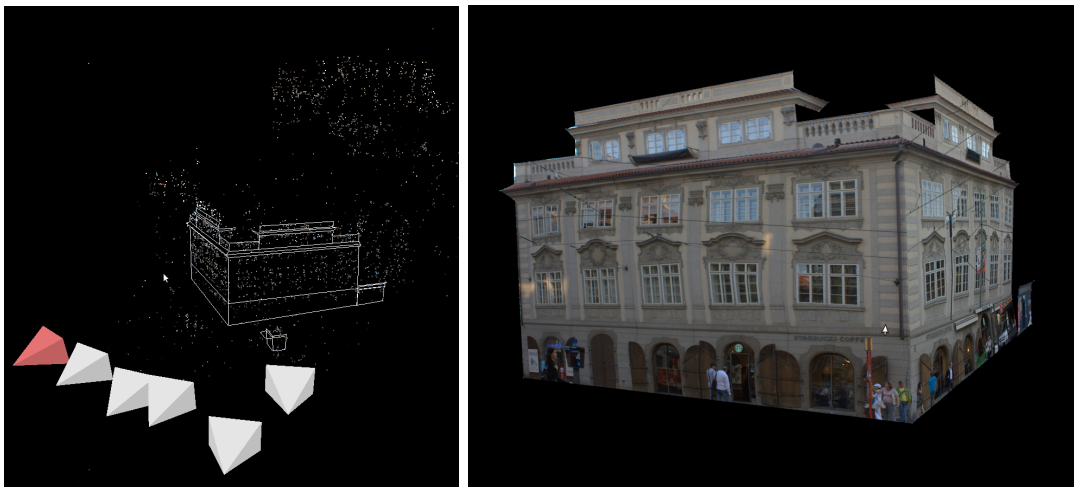
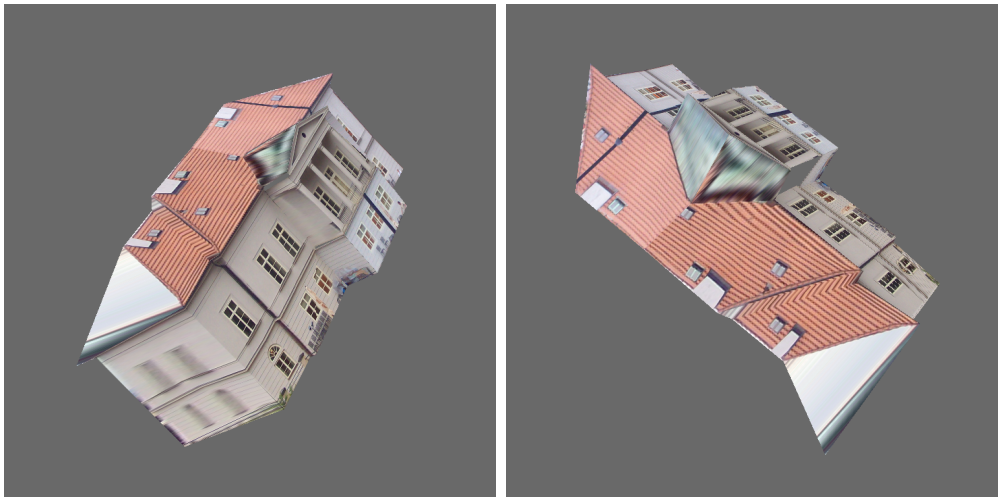


Figure 2.5: Wire frame and texture views of the scene reconstructed using our software package. Pyramids in the left image represent calibrated cameras.



(a) (b)

Figure 2.6: Two views of a polygonal 3D model obtained from 5 of its photos.

In figure 2.6, we can see another example of a 3D reconstruction. Several remarks about its quality can be made. The side view in figure 2.6(a) looks relatively realistically. The top view in figure 2.6(b) reveals two issues. The textures of some of the polygons (e.g., green roof above the entrance) are of low quality. This is caused by the fact that a photo on which these polygons would be visible from acceptable angle wasn't available. We can also see that only one half of the building has been reconstructed. Photos of the other one are hard to obtain from ground because of occlusions by other architecture. Still, polygonal model of the other side might be added using a conventional 3D modeling software by exploiting building's symmetries – see [8] for a detailed treatment of how this can be implemented as a part of an image-based modeler.

We can conclude that while taking photographs might seem to be quick and straightforward, some effort should be put into planning which photos to take so that all scene vertices are clearly visible and distinguishable and all polygons can be seen without significant occlusions.

Chapter 3

Multiple View Geometry

As we have seen in section 2.2, our application will be given projections of vertices on images that were taken by a camera from different viewpoints. From this 2D input, we need to compute 3D coordinates of the vertices and 3D parameters of the cameras (their positions, orientations, ...) used to aquire the images. This problem is sometimes termed *structure from motion*, since the *structure* of the scene (3D vertices) is determined by observing the apparent *motion* of the vertices on images taken from different positions.

This problem has been investigated during the last 150 years by photogrammetric community [7], which mostly aimed to create precise maps from photographs, and later by the computer vision community. Current state-of-art solution uses many results from projective geometry, linear algebra, statistics and nonlinear programming. When considering this problem, several questions might arise about how hard it is:

- Will there be a unique solution? Are coordinates of vertices and parameters of cameras uniquely determined by their projections?
- Is there computationaly effective algorithm to solve the problem?
- Is there numerically stable algorithm?

Answers to these questions are unfortunately negative. There will never be a unique solution, we'll work with parameter family of solutions from which we'll select one as the result of the computation. Current state-of-art solution involves solving instances of problems that are NP-complete. For some of them, no numerically stable algorithm is known.

In this chapter, we list essential concepts from projective geometry and linear algebra, introduce the mathematical model used to approximate real-world cameras and give formal mathematical definition of the problem. We give high-level overview of how the problem can be solved and describe individual algorithms used in the computation. Discussion of our actual implementation is deferred until chapter 5.

3.1 Projective geometry

Projective geometry is heavily used in 3D computer vision. There are several reasons for its popularity. First of all, it is a mature mathematical discipline with plenty of deep results. Its connections with linear algebra allow us to use algebraic expressions (mostly vectors and matrices) to formulate problems and describe solutions. However, the usage of algebraic projective geometry has its drawbacks too [17]. In the following sections we briefly describe the most important definitions, algorithms and results. Details can be found in [14].

Projective geometry can be defined as the study of properties of *projective plane* that are invariant under *projective transformations*.

Definition 3.1.1. *Projective plane consists of a set of points X, set of lines L and incidence relation $\iota \subseteq X \times L$, that satisfy the following projective axioms:*

1. *Given any two distinct points $x_1, x_2 \in X$, there is exactly one line $l \in L$ incident with both of them.*
2. *Given any two distinct lines $l_1, l_2 \in L$, there is exactly one point $x \in X$ incident with both of them.*
3. *There are four points such that no line is incident with more than two of them.*

Definition 3.1.2. *Projective transformation (also called homography) is invertible mapping h from the set of points X of a projective plane to itself such that three points x_1, x_2, x_3 lie on the same line if and only if $h(x_1), h(x_2), h(x_3)$ do.*

We will be mostly interested in the real projective plane, which can be obtained from the classic Euclidean plane $\mathbb{R}^n$ by extending it with points at infinity. The resulting plane will be denoted by $\mathbb{P}^n$. For $n = 3$, we speak about points and planes instead of points and lines.

The points on $\mathbb{P}^n$ can be elegantly defined using homogeneous vectors from $\mathbb{R}^{n+1} \setminus \{\mathbf{0}\}$, which are nonzero vectors defined up to an arbitrary nonzero scale factor. For example, the vectors $\begin{pmatrix}1 & 3 & -2\end{pmatrix}^\mathsf{T}$ and $\begin{pmatrix}2 & 6 & -4\end{pmatrix}^\mathsf{T}$ represent the same point. Likewise, lines in $\mathbb{P}^2$ are represented using nonzero homogeneous vectors from $\mathbb{R}^3$ and planes in $\mathbb{P}^3$ using nonzero homogeneous vectors from $\mathbb{R}^4$ (lines in $\mathbb{P}^3$ are harder to represent, see [14, pg. 68-73] for an overview of possible methods).

The incidence relation is defined as follows: a point $\mathbf{x}$ lies on line $\mathbf{l}$ if and only if $\mathbf{x}^\mathsf{T}\mathbf{l} = 0$. Other entites can be also represented as homogeneous vectors or matrices (i.e., nonzero matrices defined up to an arbitrary nonzero scale factor). For example, conics are represented by homogeneous symmetric matrices $\mathtt{C} \in \mathbb{R}^{3\times3}$ and point $\mathbf{x}$ lies on this conic if and only if $\mathbf{x}^\mathsf{T}\mathtt{C}\mathbf{x} = 0$.

Specifically, in the so called *canonical coordinates*, Eucledian point $\begin{pmatrix}x & y\end{pmatrix}^\mathsf{T}$ is represented by any of the 3-vectors of the form $\begin{pmatrix}kx & ky & k\end{pmatrix}^\mathsf{T}$, $k \in \mathbb{R}\setminus\{0\}$. The

points at infinity are represented by nonzero vectors of the form $\begin{pmatrix} kx & ky & 0 \end{pmatrix}^\mathsf{T}$, $k \in \mathbb{R} \setminus \{0\}$. Note that the line $\begin{pmatrix} 0 & 0 & 1 \end{pmatrix}^\mathsf{T}$ contains all points at infinity. For this reason, it is called *the line at infinity* and denoted $\mathbf{l}_\infty$.

It is this simple extension which makes great simplifications possible. For example, in $\mathbb{P}^2$, every pair of distinct lines has exactly one intersection (see the second axiom in definition 3.1.1). This is not true in Euclidean geometry, because distinct parallel lines have no intersection (in $\mathbb{P}^2$, parallel lines intersect on one of the newly added points at infinity). Similarly, in projective geometry every pair of distinct conics intersects in 4 points. In Euclidean geometry, two distinct elipses intersect in up to 4 points, but two distinct circles form one of the special cases – they intersect in at most two points. In $\mathbb{P}^2$, the additional two intersections are the so called *absolute* or *circular points*, which are points on $\mathbf{l}_\infty$ with canonical coordinates $\begin{pmatrix} 1 & \pm i & 0 \end{pmatrix}^\mathsf{T}$. It can be easily verified that these two complex infinite points lie on *every* circle.

These facts about $\mathbb{P}^2$ can be straightforwardly generalized into $\mathbb{P}^3$. Points of $\mathbb{P}^3$ are represented by nonzero homogeneous vectors $\mathbf{X} \in \mathbb{R}^4 \setminus \{\mathbf{0}\}$. The role of lines is replaced by planes, which are again represented by homogeneous vectors $\pi \in \mathbb{R}^4 \setminus \{\mathbf{0}\}$. Specifically, the line at infinity becomes the plane at infinity, which will be denoted by π_∞ and which has canonical coordinates $\begin{pmatrix} 0 & 0 & 0 & 1 \end{pmatrix}^\mathsf{T}$. Absolute points are replaced by the absolute conic Ω_∞ – more will be said about this entity in the section 3.11. Second order surfaces in projective 3-space are called *quadrics* and are represented by homogeneous symmetric matrices $\mathtt{Q} \in \mathbb{R}^{4\times 4}$. As with conics, a point lies on a quadric $\mathtt{Q}$ if and only if $\mathbf{X}^\mathsf{T}\mathtt{Q}\mathbf{X} = 0$.

3.2 Projective reference frame

It is important to note that the definition of projective plane doesn't distinguish between finite points and points at infinity. If we're given a set of points in projective plane $\mathbb{P}^2$ (e.g., as an output of an algorithm that uses only results from projective geometry), we can't be automatically sure where lies the line at infinity. This means that we can't decide which pairs of lines are parallel and which aren't – in other words, we can't make affine measurements. Similarly, we can't be sure what are the coordinates of the absolute points and thus can't distinguish between circles and (say) ellipses. More precisely, we can't carry out Euclidean measurements such as determining angles between lines. Indeed, all these concepts lie outside of the scope of projective geometry. It can't tell us anything about them, since it studies only the quantities that are unchanged by projective transformations and both parallelism and angles can very well be affected by the application of a general homography.

We first need to decide which line should be interpreted as $\mathbf{l}_\infty$ and which two points on it are the absolute points. After we know their coordinates in current projective frame, we can apply a refining projective transformation that moves them to their canonical form. Affine and Euclidean measurements can then be

easily carried out.

If we would ignore this fact and immediately transformed all points to "canonical" form by simply dividing their vectors by the last coordinate and interpreting the first two as coordinates of a Euclidean point, the result would differ from the right one by an unknown projective transformation since the data we have gotten have been in their own (unknown) projective frame.

3.3 Projective transformations

Homography in $\mathbb{P}^n$ was introduced in definition 3.1.2 as a transformation that maps lines to lines. Algebraically, it can be expressed by homogeneous nonzero nonsingular matrix $\mathtt{H} \in \mathbb{R}^{(n+1)\times(n+1)}$. Under this mapping, point $\mathbf{x}$ is transformed to point $\mathbf{x}' = \mathtt{H}\mathbf{x}$ and line $\mathbf{l}$ is transformed to $\mathbf{l}' = \mathtt{H}^{-\mathsf{T}}\mathbf{l}$. We can indeed verify that if the point $\mathbf{x}$ lies on the line $\mathbf{l}$ (or equivalently, if $\mathbf{x}^\mathsf{T}\mathbf{l} = 0$), then the image of the point $\mathbf{x}$ under the mapping $\mathtt{H}$ lies on the image of the line $\mathbf{l}$:

$$\mathbf{x}'^\mathsf{T}\mathbf{l}' = (\mathtt{H}\mathbf{x})^\mathsf{T}(\mathtt{H}^{-\mathsf{T}}\mathbf{l}) = \mathbf{x}^\mathsf{T}\mathtt{H}^\mathsf{T}\mathtt{H}^{-\mathsf{T}}\mathbf{l} = \mathbf{x}^\mathsf{T}\mathtt{I}\mathbf{l} = \mathbf{x}^\mathsf{T}\mathbf{l} = 0.$$

The set of all homographies forms a group since the inverse of a homography and the composition of two homographies are again homographies. Within this group, two important subgroups can be identified:

- *Affine transformation* is a transformation fixing the line or plane at infinity (as a set, not pointwise). Algebraically, it can be expressed in canonical coordinates by matrix with the last row composed of zeros, except for the bottom right value, which is equal to 1. Thus, for $\mathbb{P}^2$ and $\mathbb{P}^3$, it is represented by matrices:

$$\begin{bmatrix} a_{11} & a_{12} & t_1 \\ a_{21} & a_{22} & t_2 \\ 0 & 0 & 1 \end{bmatrix} \text{ and } \begin{bmatrix} a_{11} & a_{12} & a_{13} & t_1 \\ a_{21} & a_{22} & a_{23} & t_2 \\ a_{31} & a_{32} & a_{33} & t_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \text{respectively.}$$

  After applying affine transformation, parallel lines remain parallel.

- *Similarity transformation* is an affine transformation, for which the upper left submatrix (formed by removing the last row and last column) is orthogonal matrix multiplied by a scale factor $s \in \mathbb{R}$. In $\mathbb{P}^2$ and $\mathbb{P}^3$, it can be equivalently defined as a transformation that fixes the absolute points and the absolute conic, respectively. Angles and ratios of lengths stay invariant after applying a similarity transformation.

Every homography can be decomposed into the following chain of transformations:

$$\mathtt{H} = \mathtt{H}_P\mathtt{H}_A\mathtt{H}_S = \begin{bmatrix} \mathtt{I} & \mathbf{0} \\ \mathbf{v}^\mathsf{T} & 1 \end{bmatrix} \begin{bmatrix} \mathtt{K} & \mathbf{0} \\ \mathbf{0}^\mathsf{T} & 1 \end{bmatrix} \begin{bmatrix} s\mathtt{R} & \mathbf{t} \\ \mathbf{0}^\mathsf{T} & 1 \end{bmatrix},$$

where $\mathtt{H}_s$ is a similarity transformation, $\mathtt{H}_a$ is an affine transformation, $\mathtt{I}$ denotes identity matrix, $\mathbf{0}$ is zero vector and $\mathbf{v}, \mathbf{t}$ are arbitrary vectors. This can be proven and carried out simply by applying the QR decomposition.

3.4 The projective camera

Previous section was concerned with transformations from $\mathbb{P}^n$ to $\mathbb{P}^n$. Similarly, we can introduce transformations from $\mathbb{P}^n$ to $\mathbb{P}^m$, where $m, n \in \mathbb{N}$. Of particular interest will be the mapping from $\mathbb{P}^3$ to $\mathbb{P}^2$, since it is suitable to approximate real-world cameras, which take planar images of three-dimensional scenes. This mapping can be represented by a nonzero homogeneous matrix $\mathtt{P} \in \mathbb{R}^{3\times 4}$ of rank 3. Provided that the left 3×3 submatrix of $\mathtt{P}$ is nonsingular, $\mathtt{P}$ can be decomposed as:

$$\mathtt{P} = \mathtt{K}\begin{bmatrix} s\mathtt{R} & \mathbf{t} \end{bmatrix}.$$

Note that this is essentially the same decomposition as in the previous section. The assumption of nonsingularity is needed to carry out the QR decomposition. However, we'll use a slight variation:

$$\mathtt{P} = \mathtt{KR}\begin{bmatrix} \mathtt{I} & -\widetilde{\mathbf{C}} \end{bmatrix}, \tag{1}$$

where $\widetilde{\mathbf{C}} \in \mathbb{R}^3$. In canonical coordinates (i.e., when we're in Euclidean space), these entities have the following interpretation:

- $\widetilde{\mathbf{C}} = \begin{pmatrix} x & y & z \end{pmatrix}^\mathsf{T}$ is the camera center in inhomogeneous coordinates.
- $\mathtt{R}$ is a rotation matrix defining the orientation of the camera.
- $\mathtt{K}$ is an upper-triangular matrix, which determines internal camera parameters such as the focal length. For this reason, it is called the internal calibration matrix of the camera.

The matrix $\mathtt{P}$ is a homogeneous matrix and so the matrices $\mathtt{P}$ and $\lambda\mathtt{P}$, $\lambda \in \mathbb{R}\setminus\{0\}$, represent the same projection (and *the same camera*). In this decomposition, $\widetilde{\mathbf{C}}$ is inhomogeneous vector and the scale of $\mathtt{R}$ is fixed (since it is a rotation matrix). Thus, the effect of multiplying $\mathtt{P}$ with λ is propagated into $\mathtt{K}$. This ambiguity can be removed by requiring that $\mathtt{K}_{33} = 1$. The equation (1) is then satistied only up to an arbitrary nonzero scale factor. Moreover, the ambiguity of the QR decomposition is removed by requiring $\mathtt{K}_{11} > 0, \mathtt{K}_{22} > 0$.

The matrix $\mathtt{K}$ now has the form:

$$\mathtt{K} = \begin{bmatrix} fm_x & s & x_0 \\ 0 & fm_y & y_0 \\ 0 & 0 & 1 \end{bmatrix}.$$

The interpretation of these values is the following:

- m_x and m_y represent the number of pixels per unit distance in image coordinates in the x and y directions. The ratio $a = m_x/m_y$ is termed *aspect ratio.*

- f is the focal length of the camera in the same units that were used in the definition of m_x and m_y.

- s is the skew parameter. Nonzero values of this parameter indicate that the pixels of the camera are not perfect rectangles, but are effectively skewed.

- x_0 and y_0 are the coordinates of the principal point on the image plane (usually in pixels). This point is the image of the principal ray (the ray going through the camera center perpendicularly to the image plane) in the image coordinate system.

Typical cameras have zero skew (i.e., their pixels are very near to perfect rectangles) and the aspect ratio is usually nearly 1.0 (i.e., the pixels are nearly squares). Principal point is commonly assumed to be in the middle of the image, although this can deviate for cheaper cameras. Aside from pixels being imperfect rectangles, nonzero skew can be also caused by imperfect alignment of the camera lenses. A typical example of the internal calibration matrix for a camera which produces images with the resolution 2848px×2134px with horizontal field of view 60°:

$$\mathtt{K} = \begin{bmatrix} 2466 & 0 & 1424 \\ 0 & 2466 & 1067 \\ 0 & 0 & 1 \end{bmatrix}.$$

Real cameras deviate from the introduced model in several ways. Most importantly, real cameras don't generally project straight lines in scene as straight lines. They appear to be slightly curved in the images. This is caused by barrel or pincushion distortion – a nonlinear distortion of the image. In this work, we ignore this deviation and assume that the photos used as input for the reconstruction either are not significantly deformed or were corrected using a third-party application. Practical results indicate that our implementation produces good results even without nonlinear correction.

3.5 Structure from motion

The problem of *structure from motion* is that of estimating the parameters of 3D geometric entities by observing apparent motion of their projections on several images taken from different viewpoints.

Some properties of the input for this task were discussed in section 2.3. Here we reformulate the problem in terms of algebraic projective geometry. The marked projection of i-th vertex on j-th image will be represented by nonzero

homogeneous 3-vector $\mathbf{x}_i^j$, scene vertices by nonzero homogeneous 4-vectors $\mathbf{X}_i$ and the cameras will be modeled by projection matrices $\mathtt{P}_j$.

To summarise, we have the following meassurements:

- coordinates $\mathbf{x}_i^j$ of images of the unknown vertices projected using the unknown projection matrices.

These values can be submitted manually by the user or obtained using automatic matching algorithms (discussed in the next chapter). Some of the values $\mathbf{x}_i^j$ are left undefined, e.g., as a result of occlusions. From this input, we want to calculate:

- vectors $\mathbf{X}_i$ representing the 3D vertices,
- projection matrices $\mathtt{P}_j$ representing the camera projections.

The projection of the vertex $\mathbf{X}$ using the projection matrix $\mathtt{P}$ is given by $\mathbf{x} = \mathtt{P}\mathbf{X}$. However, points, vertices and projections are all represented by homogeneous vectors and matrices and thus this equation will be satisfied only up to an (unknown) scale factor $\lambda \in \mathbb{R} \setminus \{0\}$. If all of the measurements were exact and all computations done on a computer with infinite precision, then solving the following set of nonlinear equations would lead to an exact solution for $\mathbf{X}_i$ and $\mathtt{P}_j$:

$$\lambda_{ij}\mathbf{x}_i^j = \mathtt{P}_j\mathbf{X}_i, \tag{2}$$

where $\lambda_{ij} \in \mathbb{R} \setminus \{0\}$.

Two issues, informally introduced in section 2.3, have to be considered:

It can be easily verified that the output is not uniquely determined by the input. Indeed, we can take any non-singular 4×4 matrix $\mathtt{H}$ and replace each vertex $\mathbf{X}_i$ by $\mathtt{H}\mathbf{X}_i$ and each projection matrix $\mathtt{P}_j$ by $\mathtt{P}_j\mathtt{H}^{-1}$. We immediately see that if the original matrices satisfied equations (2) then so do the modified ones:

$$\lambda_{ij}\mathbf{x}_i^j = \mathtt{P}_j\mathbf{X}_i = \mathtt{P}_j\mathtt{I}_4\mathbf{X}_i = \mathtt{P}_j\mathtt{H}^{-1}\mathtt{H}\mathbf{X}_i.$$

If we carry out the reconstruction based on this set of equations, then the result will differ from the real one by an unknown homography $\mathtt{H}$. Such reconstruction is called *the projective reconstruction*, since only scene properties that are projective invariants (for example, ratios of areas) can be meassured. This is insufficient for almost all practical applications, since a general homography affects affine and Euclidean properties of the scene like parallelism. Thus, scene planes that should be parallel or perpendicular will likely lie in general positions. Such deformation of the scene is clearly unacceptable for example when using the reconstruction for visualization purposes. To identify the homography $\mathtt{H}$ that establishes the correct projective frame, we'll have to exploit non-projective properties of the scene or cameras.

For example, we could have the user explicitly pick several pairs of planes that are parallel. Since parallel planes intersect on the plane at infinity π_∞,

we could identify the coordinates of π_∞ from these intersections and then apply homography that transforms the plane to its canonical position. Affine properties of the scene would then be preserved.

Another possibility is to derive Euclidean properties of the scene from certain assumptions about the scene cameras. Typical real-world digital cameras usually have approximately square pixels and thus have zero skew and aspect ratio 1.0. This can be used calculate the refining homography that takes the reconstruction into Euclidean projective frame.

Since we aim to create a software package that is as automatic as possible, we choose the second approach over the first one, which requires manual assistance of the user. The second approach, termed *autocalibration* or *self-calibration*, has some disadvantages too. The main problem is that it generally requires to find global minimum of a highly nonlinear function with many local minima [15]. Details about the autocalibration algorithm used in our software package are given in section 3.11.

The second issue is that in real scenarios, the measurements will always have only finite precision and will be corrupted by noise. No single reconstruction then satisfies all equalities (2). Still, we can try to find among all reconstructions the *best* one. One of course has to define what "best" means. Generally accepted method is to seek the maximum likelihood (ML) solution, given some assuption about the probabilistic distribution of the noise. Gaussian distribution is usually used because it is easy to handle analytically.

Formally, we try to find a solution that minimizes some *cost function* defined on the scene parameters $\{\mathbf{X}_i\}_i$ and $\{\mathtt{P}_j\}_j$. This cost function assigns a non-negative real value (cost) to the scene parameters, which is proportionate to how well the parameters match the user input. Under the assumption that the noise follows isotropic mean-zero Gaussian distribution (independent and identical for every measured point), the ML solution is obtained by minimizing the following expression:

$$\min_{\mathbf{X}_j, \mathtt{P}_i} \sum d(\mathtt{P}_i \mathbf{X}_j, \mathbf{x}_i^j)^2, \tag{3}$$

where $d(\mathbf{x}, \mathbf{y})$ denotes the geometric distance between the points $\mathbf{x}$ and $\mathbf{y}$. The points $\{\mathbf{X}_i\}_i$ and cameras $\{\mathtt{P}_j\}_j$ that minimize the sum of squared geometric distances between a measured points $\{\mathbf{x}_i^j\}_{i,j}$ and the projections $\mathtt{P}_i\mathbf{X}_j$ are the most probable reconstruction under the given assumptions. The reader can refer to [14, pg. 633] for an overview of cost functions for various distributions of noise.

Unfortunately, solving such instances of nonlinear programming is theoretically problematic. Nonlinear programming is generally NP-hard and it can be shown [13] that so is this special case. Since the function to be minimized in (3) is differentiable, one approach to find the optimal solution would be to compute all its stationary points and among them find the one that leads to the minimal cost. If the cost function is rational (which is true for cost functions based on most of the noise distributions), then so are the derivatives. Equating them to zero leads to a system of polynomial equations. However, this solution is reason-

able only for problems of minimal size, since the degree of the polynomial to be solved grows cubically [15] and finding roots of such polynomials is numerically problematic. The intractability of using this method in almost all cases has been noted by photogrammetrists early on in the 1920's [7].

The approach used in practice is as follows:

1. find an approximate solution that minimizes a suitable function similar (although not identical) to our cost function,

2. use iterative numerical methods to minimize our cost function using the obtained approximate solution as a starting point.

In the following sections, we describe some of the techniques used to construct the initial approximate solution and in section 3.10 describe an optimization algorithm commonly used to refine the initial solution so that it approaches the optimal one. This optimization is termed *bundle adjustment* in literature. Its main feature is that it corrects the camera parameters and the positions of the vertices simultaneously.

3.6 Construction of the initial solution

Even for a reasonably simplified cost function, there still isn't a single straightforward way to obtain an approximate solution to our problem. It can be however constructed by employing a series of algorithms, each of which carries out a small part of the computation.

The computation can be started by estimating projection matrices $\mathtt{P}_i, \mathtt{P}_j$ for an image pair with at least 8 correspondences. This is done using the fundamental matrix described in the next section. Aimed with the knowledge of projection matrices for an image pair, we can estimate the position of the vertices visible on both of the images using vertex triangulation described in section 3.8. In this way, we can obtain starting point for the computation of the initial solution.

If there is a camera in our dataset that still is not estimated (its projection matrix is unknown), but on which at least 6 already estimated space points are marked, we can apply the camera resection method of section 3.9. Using it, the projection matrix is computed from the known coordinates of the vertices and their marked projections. After it is estimated, triangulation is once again used to calculate the vertices that are visible on at least 2 currently known cameras (some of them might have been already calculated, but the triangulation is done again to take into account the newly gained knowledge).

Of course, it is not evident which image pair should be used to bootstrap the computation and in which order the camera resections and vertex triangulations should be performed to achieve the optimal numerical stability and precision. There even is a danger of computing completely incorrect initial solution (one that leads to approximation far away from the global minimum), when the data are in a degenerate configuration. For example, the projection matrix cannot be

uniquely determined when we perform camera resection from vertices lying on a single scene plane.

Fundamental matrix, resection and triangulation are not the only possible building blocks for construction of the initial solution. For example, three projection matrices can be estimated at once when projections of 6 points are known on all three images using trifocal tensor described in [14, pg. 363-408]. Matrix factorization using the SVD [14, pg. 436] can produce near-optimal results at estimating n projection matrices when the cameras used to obtain the images had long focal lengths (and thus can be approximated by affine cameras – ones that have the projection center located at infinity). We refer the reader to [14] for an indepth discussion of possible techniques.

The strategy used to construct the initial solution in our implementation is described in the section 5.3.

3.7 The fundamental matrix

The relation between two views in projective space $\mathbb{P}^3$ can be neatly expressed using a 3×3 real matrix $\mathtt{F}$ of rank 2 called *the fundamental matrix*.

Consider a point $\mathbf{X} \in \mathbb{P}^3$ and two different cameras $\mathtt{P}_1$ and $\mathtt{P}_2$ (to avoid a degenerate configuration, suppose that $\mathbf{X}$ and the two camera centers are not colinear). Furthermore, denote the image of $\mathbf{X}$ in the first view by $\mathbf{x}$. What can $\mathbf{x}$ tell us about the position of this point's projection in the second view? The fact that we know the projection of $\mathbf{X}$ in the first view restricts its position to a line in $\mathbb{P}^3$ going through the first camera's center. This line is projected onto the second image as line $\mathbf{l}'$. The image of $\mathbf{X}$ in the second photo, denoted by $\mathbf{x}'$, has to be somewhere on it.

The fundamental matrix expresses precisely such mapping between points in one view and lines in another.

Definition 3.7.1. *The* ***fundamental matrix*** $\mathtt{F}$ *for a pair of images is a* 3×3 *homogeneous matrix which satisfies*

$$\mathbf{x}'^{\mathsf{T}}\mathtt{F}\mathbf{x} = 0$$

for all corresponding points $\mathbf{x} \leftrightarrow \mathbf{x}'$.

By multiplying a point $\mathbf{x}$ from left by $\mathtt{F}$ we get a line $\mathbf{l}' = \mathtt{F}\mathbf{x}$ in the second image. The point $\mathbf{x}'$ lies on this line since $\mathbf{x}'^{\mathsf{T}}\mathbf{l}' = \mathbf{x}'^{\mathsf{T}}\mathtt{F}\mathbf{x} = 0$. The line $\mathbf{l}'$ is called the epipolar line.

We now prove the existence of such matrix. This algebraic proof follows [14, pg. 243], where the reader can also find its geometric counterpart.

Theorem 3.7.2. *The fundamental matrix for two images acquired by cameras with non-coincident centers exists and is a uniquely determined homogeneous matrix of rank 2.*

Proof. The set of points in space that map to a given point $\mathbf{x} \in \mathbb{P}^2$ in a camera image is a line in $\mathbb{P}^3$. To identify this line, we find two distinct points in space that lie on it. One of them is the camera center $\widetilde{\mathbf{C}}$, since every projection ray goes through it. Another point can be obtained using pseudo-inverse of $\mathtt{P}$. This is a matrix $\mathtt{P}^+ = \mathtt{P}^\mathsf{T}(\mathtt{P}\mathtt{P}^\mathsf{T})^{-1}$. The pseudo-inverse of $\mathtt{P}$ has the property that $\mathtt{P}\mathtt{P}^+ = \mathtt{I}$ since $\mathtt{P}\mathtt{P}^+ = (\mathtt{P}\mathtt{P}^\mathsf{T})(\mathtt{P}\mathtt{P}^\mathsf{T})^{-1} = \mathtt{I}$. The point $\mathtt{P}^+\mathbf{x}$ also lies on our line. Indeed, it projects to the point $\mathbf{x}$:

$$\mathtt{P}(\mathtt{P}^+\mathbf{x}) = \mathtt{I}\mathbf{x} = \mathbf{x}.$$

By projecting these two points using the second camera, denoted by $\mathtt{P}'$, we get points $\mathtt{P}'\widetilde{\mathbf{C}}$ and $\mathtt{P}'\mathtt{P}^+\mathbf{x}$ in the second image. It can be easily verified that a line going through points $\mathbf{x}$ and $\mathbf{y}$ is $\mathbf{x} \times \mathbf{y}$, where $\times$ denotes the cross product. The epipolar line is thus $\mathbf{l}' = (\mathtt{P}'\widetilde{\mathbf{C}}) \times (\mathtt{P}'\mathtt{P}^+\mathbf{x})$. The cross product can be straightforwardly rewritten as matrix multiplication, which leaves us with $\mathbf{l}' = [\mathtt{P}'\widetilde{\mathbf{C}}]_\times(\mathtt{P}'\mathtt{P}^+)\mathbf{x}$, where $[\mathbf{a}]_\times$ represents the following skew-symmetric matrix:

$$\begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix}.$$

It follows that the matrix $\mathtt{F}$ is equal to $[\mathtt{P}'\widetilde{\mathbf{C}}]_\times(\mathtt{P}'\mathtt{P}^+)$ and its rank is 2 (since non-zero 3×3 skew-symmetric matrix clearly has rank 2). □

Our main motivation for using and computing fundamental matrices is that it allows us to extract the projection matrices for the camera pair. The following result shows how:

Theorem 3.7.3. *Let* $\mathtt{F}$ *be a fundamental matrix and* $\mathtt{S}$ *any skew-symmetric matrix. Then* $\mathtt{F}$ *is the fundamental matrix corresponding to the following camera pair:*

$$\mathtt{P} = \begin{bmatrix} \mathtt{I} & \mathbf{0} \end{bmatrix},$$
$$\mathtt{P}' = \begin{bmatrix} \mathtt{S}\mathtt{F} & \mathbf{e}' \end{bmatrix},$$

where $\mathbf{e}'$ *is nonzero vector such that* $\mathbf{e}'^\mathsf{T}\mathtt{F} = \mathbf{0}$ *(assuming that the matrix* $\mathtt{P}'$ *has rank 3).*

To prove the theorem 3.7.3, we need the following lemma:

Lemma 3.7.4. *A non-zero matrix* $\mathtt{F}$ *is the fundamental matrix corresponding to a pair of camera matrices* $\mathtt{P}$ *and* $\mathtt{P}'$ *if and only if* $\mathtt{P}'^\mathsf{T}\mathtt{F}\mathtt{P}$ *is skew-symmetric.*

Proof. The condition that $\mathtt{P}'^\mathsf{T}\mathtt{F}\mathtt{P}$ is skew-symmetric is equivalent to $\mathbf{X}^\mathsf{T}\mathtt{P}'\mathtt{F}\mathtt{P}\mathbf{X} = 0$ for all $\mathbf{X}$. Letting $\mathbf{x} = \mathtt{P}\mathbf{X}$ and $\mathbf{x}' = \mathtt{P}'\mathbf{X}$ transforms this into $\mathbf{x}'\mathtt{F}\mathbf{x} = 0$, which is the defining equation of the fundamental matrix. □

Now that we've proven the lemma, we can prove the result 3.7.3:

Proof. Using the lemma 3.7.4, we only have to check that the following matrix is skew-symmetric:

$$\begin{bmatrix}\mathtt{SF} & \mathbf{e}'\end{bmatrix}^{\mathsf{T}} \mathtt{F} \begin{bmatrix}\mathtt{I} & \mathbf{0}\end{bmatrix} = \begin{bmatrix}\mathtt{F}^{\mathsf{T}}\mathtt{S}^{\mathsf{T}}\mathtt{F} & \mathbf{0} \\ \mathbf{e}'^{\mathsf{T}}\mathtt{F} & 0\end{bmatrix} = \begin{bmatrix}\mathtt{F}^{\mathsf{T}}\mathtt{S}^{\mathsf{T}}\mathtt{F} & \mathbf{0} \\ \mathbf{0}^{\mathsf{T}} & 0\end{bmatrix},$$

which can be easily verified. □

Thanks to this result, once we calculate the fundamental matrix for an image pair, we can compute the corresponding projection matrices. What now remains is to show how to compute the fundamental matrix from corresponding points.

This can be done using linear algebraic methods when at least 8 point matches are given. Each pair of corresponding points $(\mathbf{x}, \mathbf{x}')$ leads to the following equation constraining the unknown matrix $\mathtt{F}$:

$$\mathbf{x}'^{\mathsf{T}}\mathtt{F}\mathbf{x} = 0.$$

Assuming $\mathbf{x} = (x, y, 1)^{\mathsf{T}}$ and $\mathbf{x}' = (x', y', 1)^{\mathsf{T}}$, it can be rewritten as:

$$x'xf_{11} + x'yf_{12} + x'f_{13} + y'xf_{21} + y'yf_{22} + y'f_{23} + xf_{31} + yf_{32} + f_{33} = 0.$$

A set of n correspondences thus results in the following system of linear equations:

$$\mathtt{A}\mathbf{f} = \begin{bmatrix} x'_1x_1 & x'_1y_1 & x'_1 & y'_1x_1 & y'_1y_1 & y'_1 & x_1 & y_1 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x'_nx_n & x'_ny_n & x'_n & y'_nx_n & y'_ny_n & y'_n & x_n & y_n & 1 \end{bmatrix} \mathbf{f} = \mathbf{0},$$

where $\mathbf{f}$ is a 9-vector made up from the entries of $\mathtt{F}$ in row-major order.

The fundamental matrix $\mathtt{F}$ is a homogeneous matrix, which means that it is defined only up to scale. Thus, even when the number of equations is large ($n > 8$), the matrix $\mathtt{A}$ should have one parameter right null-space whose generator is the nonzero vector $\mathbf{f}$ used to construct the desired matrix $\mathtt{F}$. When we have 9 or more correspondences, it might happen that the matrix $\mathtt{A}$ becomes non-singular due to noise in image coordinates. In such cases there is no nonzero vector $\mathbf{f}$ that satisfies $\mathtt{A}\mathbf{f} = \mathbf{0}$. What makes sense is to find the vector that minimizes $|\mathtt{A}\mathbf{f}|$ subject to the constraint $|\mathbf{f}| = 1$. This vector $\mathbf{f}$ is the singular vector corresponding to the smallest singular value of $\mathtt{A}$ [14, pg. 588].

After finding the unit 9-vector $\mathbf{f}$ minimizing $|\mathtt{A}\mathbf{f}|$ and rewriting $\mathbf{f}$ into 3×3 matrix $\mathtt{F}$, we have to enforce the fact that the fundamental matrix must have rank 2. It could very well happen that our matrix is regular and thus does not represent any valid epipolar geometry. In such cases, we want to find the closest matrix (in Frobenius norm) with rank 2 to the one calculated in the previous step.

The rank property can be easily enforced using the SVD: let $\mathtt{F} = \mathtt{U}\mathtt{D}\mathtt{V}^{\mathsf{T}}$ be a decomposition of the matrix $\mathtt{F}$ into the product of matrix $\mathtt{U}$ with orthogonal columns, diagonal matrix $\mathtt{D}$ (with the elements on the main diagonal being in

decreasing order) and orthogonal matrix $\mathtt{V}$. Furthermore, let $\mathtt{D}'$ be the matrix obtained by taking $\mathtt{D}$ and equating the bottom right element to zero. It follows from the norm preserving property of orthogonal matrices that the closest rank 2 matrix to $\mathtt{F}$ is $\mathtt{U}\mathtt{D}'\mathtt{V}^\mathsf{T}$.

The resulting algorithm to obtain projection matrices $\mathtt{P}_1, \mathtt{P}_2$ for an image pair from a set of its point correspondences thus follows this outline:

1. Given $n \geq 8$ point correspondences, construct the $n \times 9$ matrix $\mathtt{A}$.
2. Find the singular vector $\mathbf{f}$ of $\mathtt{A}$ corresponding to its smallest singular value.
3. Construct the matrix $\mathtt{F}$ from $\mathbf{f}$.
4. Reduce the rank of $\mathtt{F}$ to 2 using SVD as described above.
5. Extract the projection matrices $\mathtt{P}_1, \mathtt{P}_2$ using the result 3.7.3.

3.8 Triangulation of vertices

We now turn to the problem of estimating the position of a point in space from several (at least two) of its known projections. Assume we are given projection matrices $\mathtt{P}_1, \mathtt{P}_2, \ldots, \mathtt{P}_n$ and the coordinates $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$ of projections of this point on respective cameras. We want to find such a point $\mathbf{X}$ in $\mathbb{P}^3$ that best fits these measurements. Such computation is called *triangulation*.

As in section 3.5, we would ideally like to find a point that minimizes the sum of squared reprojection distances:

$$\min_{\mathbf{X}} \sum_{i=1..n} d(\mathtt{P}_i\mathbf{X}, \mathbf{x}_i)^2.$$

This can be viewed as a flavour of the general (and NP-hard) problem to minimize (3) from 3.5, with the provision that the camera matrices $\mathtt{P}_i$ are fixed and only the point $\mathbf{X}$ is allowed to vary. However, this special case is somewhat easier to solve optimally. An efficient algorithm for the case of two images is known since [16], while recent breakthroughs in applications of branch and bound method in Second Order Cone Programming have lead to optimal algorithm for triangulation from a general number of images [18].

In this section, we describe an algorithm based on linear algebra [14, pg. 312]. As such, it doesn't minimize the geometric reprojection error but only its approximation. It has the advantage that it can be readily implemented when we have a library with appropriate linear algebraic functions available. We derive this method for the case of two photos ($n = 2$). Note however that it can be straightforwardly generalised.

Since we are dealing with homogeneous quantities, the projection equation $\mathbf{x}_i = \mathtt{P}_i\mathbf{X}$ is satisfied only up to an unknown scale factor. This can be eliminated by rewriting the equation using cross product: $\mathbf{x}_i \times (\mathtt{P}_i\mathbf{X}) = \mathbf{0}$. Denoting the j-th row of the matrix $\mathtt{P}_i$ by 4-vector $\mathbf{p}_i^{j\mathsf{T}}$ and letting $\mathbf{x}_i = (x_i, y_i, w_i)$ allows us to write this condition as:

$$\begin{aligned} x_1(\mathbf{p}_1^{3\mathsf{T}}\mathbf{X}) - (\mathbf{p}_1^{1\mathsf{T}}\mathbf{X}) &= 0 \\ y_1(\mathbf{p}_1^{3\mathsf{T}}\mathbf{X}) - (\mathbf{p}_1^{2\mathsf{T}}\mathbf{X}) &= 0 \\ x_1(\mathbf{p}_1^{2\mathsf{T}}\mathbf{X}) - y_1(\mathbf{p}_1^{1\mathsf{T}}\mathbf{X}) &= 0 \end{aligned}$$

These equations are linear in $\mathbf{X}$ and only two of them are independent. Constraints from both images thus result in the system of linear equations $\mathtt{A}\mathbf{X} = \mathbf{0}$, where

$$\mathtt{A} = \begin{bmatrix} x_1\mathbf{p}_1^{3\mathsf{T}} - \mathbf{p}_1^{1\mathsf{T}} \\ y_1\mathbf{p}_1^{3\mathsf{T}} - \mathbf{p}_1^{2\mathsf{T}} \\ x_2\mathbf{p}_2^{3\mathsf{T}} - \mathbf{p}_2^{1\mathsf{T}} \\ x_2\mathbf{p}_2^{3\mathsf{T}} - \mathbf{p}_2^{4\mathsf{T}} \end{bmatrix}.$$

In general case, $\mathtt{A}$ will be $2n \times 4$ matrix formed by stacking up constraints from all known projections.

The situation is now similar to the previous section. If all measurements were exact, the matrix $\mathtt{A}$ would have one dimensional right null-space and our desired nonzero vector $\mathbf{X}$ would be its generator. Due to noise in the image coordinates and in the entries of the projection matrices (caused for example by inaccurate input from the user), the matrix might become regular. As in the previous section, we once again take the unit singular vector corresponding to the smallest singular value as our estimate for $\mathbf{X}$, since it minimizes the quantity $|\mathtt{A}\mathbf{X}|$ (subject to the constraint $|\mathbf{X}| = 1$, which prevents $\mathbf{X}$ from approaching $\mathbf{0}$).

The point $\mathbf{X}$ estimated in this way will likely differ from the one minimizing the reprojection error. It should however be close enough to be succesfully refined using nonlinear techniques referenced in section 3.10.

3.9 Camera resection

We now consider a situation when we have space points $\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_n$ estimated and their projections onto an image are known. Camera resectioning refers to the computation of the projection matrix $\mathtt{P}$ for this image. Once again, we describe a linear technique [14, pg. 178] used to provide an estimate to be optimized using nonlinear iterative method. We note that, as in the case of triangulation, efficient globally optimal algorithm can be obtained using Second Order Code Programming [18].

The projection of the point $\mathbf{X}_i$ will be denoted $\mathbf{x}_i$. Each $\mathbf{X}_i \leftrightarrow \mathbf{x}_i$ correspondence adds a constraint $\mathbf{x}_i = \mathtt{P}\mathbf{X}_i$ on $\mathtt{P}$ – again, satisfied only up to an unknown scale factor). As in the previous section, the multiplicative ambiguity is removed

by taking cross product. This transforms the equation to $\mathbf{x}_i \times \mathtt{P}\mathbf{X}_i = \mathbf{0}$, which can be rewritten as:

$$\begin{bmatrix} \mathbf{0}^\mathsf{T} & -w_i\mathbf{X}_i^\mathsf{T} & y_i\mathbf{X}_i^\mathsf{T} \\ w_i\mathbf{X}_i^\mathsf{T} & \mathbf{0}^\mathsf{T} & -x_i\mathbf{X}_i^\mathsf{T} \\ -y_i\mathbf{X}_i^\mathsf{T} & x_i\mathbf{X}_i^\mathsf{T} & \mathbf{0} \end{bmatrix} \begin{pmatrix} \mathtt{P}^1 \\ \mathtt{P}^2 \\ \mathtt{P}^3 \end{pmatrix} = \mathbf{0},$$

where $\mathtt{P}^j$ is the j-th row of $\mathtt{P}$ and the point $\mathbf{x}_i$ is written as $(x_i, y_i, w_i)^\mathsf{T}$. The third equation is a linear combination of the first two, so we gain 2 independent constraints from each correspondence. The projection matrix has 11 degrees of freedom (it has 12 entries, one degree is removed for scale) and thus at least 6 correspondences are necessary to solve for $\mathtt{P}$.

This is done in a way analogous to the previous two sections – we pick the singular vector of the assembled matrix corresponding to the smallest singular value. Again, this doesn't minimize the geometric error, but the algebraic one and should be later refined using iterative nonlinear methods.

3.10 Nonlinear optimization

Using the results from previous sections, we are able to acquire an initial solution which doesn't minimize the geometric error, but should be close enough that the optimal solution can be obtained through local optimization. We now informally introduce iterative method commonly used to achieve this and refer the reader to for example [27] for formal derivation.

Numerical mathematics offers two classic algorithms used to optimize differentiable functions: the Gauss-Newton algorithm and the gradient descent. Both algorithms proceed by taking an initial solution and iteratively improving it using local approximations of the function. After a given number of iterations, the process stops and outputs the improved solution which should be closer to the true minimum.

The Gauss-Newton method proceeds by approximating the function using quadratic Taylor expansion around the current position. Global minimum of this quadratic approximation is chosen as the position for the next iteration. Gradient descent method computes the direction of the steepest descent in each position by evaluating the partial derivatives. A step of certain length is then performed in this direction. If this new position fails to be better than the current one, a smaller step is tried. Since the function is differentiable, a small enough step is guaranteed to improve upon the current iteration. This is the main advantage of the gradient descent method. On the other hand, its convergence can be very slow when a lot of small steps are performed. The Gauss-Newton method usually has a very good performance, but can run into problems when the function does not locally resemble a quadratic one.

This leads us to the Levenberg-Marquardt algorithm, which combines these two approaches. It interpolates between the step taken by the Gauss-Newton

method and the gradient descent, favoring the former one as long as it succeeds to decrease the value of the cost function.

A quick description of the Levenberg-Marquardt algorithm can be found in [9], while [27] offers an in depth treatment and includes a discussion of numerical stability and pre-conditioning. Our software package uses open source implementation of this algorithm, described in technical report [19].

3.11 Autocalibration

As we have already noted in section 3.5, when we perform a reconstruction based only on the projection equations (2), the result will differ from the true metric reconstruction by an unknown general projective transformation. This *projective reconstruction* almost always won't have the plane at infinity π_∞ in its canonical position $(0,0,0,1)^\mathsf{T}$ and thus scene lines and planes that should be parallel will not appear as such. In other words, affine and Euclidean properties of the scene won't be preserved.

Projective geometry doesn't offer us any tools to deal with this problem, since it is only concerned with properties of the projective plane that are invariant under projective transformations. We are going to exploit metric properties of the cameras to obtain the correcting homography $\mathtt{H}$.

In the section 3.4, we discussed how the camera projection matrix can be decomposed into several matrices which have a geometric meaning when the reconstruction is in Euclidean frame. Specifically, we introduced the internal calibration matrix:

$$\mathtt{K} = \begin{bmatrix} fm_x & s & x_0 \\ 0 & fm_y & y_0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Typical consumer cameras have practically perfectly square pixels and correctly aligned lense system. The skew factor s thus should be nearly zero (since the pixels are rectangles) and the aspect ratio m_x/m_y should be nearly 1.0 (since the pixels are squares). Also, the principal point (x_0, y_0) is typically located approximately in the middle of the image. Choosing the image coordinate system so that the image center has coordinates $(0,0)$ results in the following simplified calibration matrix:

$$\mathtt{K} = \begin{bmatrix} fm_x & 0 & 0 \\ 0 & fm_x & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{4}$$

If we would obtain only projective reconstruction and perform decomposition of the camera matrices into $\widetilde{\mathbf{C}}, \mathtt{R}$ and $\mathtt{K}$, the upper-triangular matrix $\mathtt{K}$ would generally not reflect this (for example, the entry in the position $(1,2)$ would not be approximately 0). Based on this, we can reject the incorrect projective reconstructions – the ones that differ from the true one by more than just a similarity transformation.

We now introduce a remarkable geometric entity of $\mathbb{P}^3$ called *the absolute conic*, using which we can factor out the internal calibration matrices $\mathtt{K}_i$ from the camera matrices $\mathtt{P}_i$.

It should be intuitively clear that the image of the plane at infinity π_∞ taken by a camera does not depend on the position of the camera's center – when we move without changing camera's orientation or internal calibration, the points located at infinity stay fixed. Remarkably, there is a set of points on the plane at infinity that is not affected by both translation *and* rotation of the camera. The projection of this set of points is determined only by the camera's internal calibration matrix $\mathtt{K}$.

This set is a conic located on the plane at infinity and consists of purely imaginary points. It is called *the absolute conic* Ω_∞. In the metric frame, the points $\mathbf{X}$ on this conic satisfy the following two equations:

$$\begin{aligned} \mathbf{X}_4 &= 0, \\ \mathbf{X}_1^2 + \mathbf{X}_2^2 + \mathbf{X}_3^2 &= 0. \end{aligned}$$

The first equation restricts the conic to π_∞. The second one can be rewritten as $(\mathbf{X}_1, \mathbf{X}_2, \mathbf{X}_3)\mathtt{I}(\mathbf{X}_1, \mathbf{X}_2, \mathbf{X}_3)^\mathsf{T} = 0$. Since a point lies on a conic $\mathtt{C}$ if and only if $\mathbf{x}^\mathsf{T}\mathtt{C}\mathbf{x} = 0$, we conclude that the absolute conic lies on the plane at infinity where it is represented by the skew-symmetric matrix $\mathtt{C} = \mathtt{I}$.

Both the conic and the plane at infinity can be conveniently represented using *the absolute dual quadric* $\mathtt{Q}_\infty^*$, which we now briefly introduce. For details and proofs of the claims in the next paragraph, refer to [14, pg. 83] and [14, pg. 462].

The absolute dual quadric is algebraically represented using a 4×4 matrix of rank 3. In the metric frame, it has the canonical form:

$$\tilde{\mathtt{I}} = \begin{bmatrix} \mathtt{I} & \mathbf{0} \\ \mathbf{0}^\mathsf{T} & 0 \end{bmatrix}.$$

The plane at infinity π_∞ is a null-vector of $\mathtt{Q}_\infty^*$. Every plane π that satisfies $\pi^\mathsf{T}\mathtt{Q}_\infty^*\pi = 0$ is tangent to Ω_∞. In this way, the absolute dual quadric encapsulates both Ω_∞ and π_∞. Thus, once this quadric is identified and moved to its canonical position, the metric properties of the scene are corrected. The quadric projects (like all dual quadrics) to a dual conic:

$$\omega^* = \mathtt{P}\mathtt{Q}_\infty^*\mathtt{P}^\mathsf{T}.$$

This conic ω^* is dual to the projection of the absolute conic ω and $\omega^* = \omega^{-1}$.

We now prove that the image of the absolute conic depends only on the internal calibration of the camera used to project it:

Theorem 3.11.1. *The image of the absolute conic Ω_∞ projected using camera $\mathtt{P} = \mathtt{K}\mathtt{R}\begin{bmatrix} \mathtt{I} & -\widetilde{\mathbf{C}} \end{bmatrix}$ is the conic $\omega = (\mathtt{K}\mathtt{K}^\mathsf{T})^{-1}$.*

Proof. To calculate the projection of the absolute conic, we first derive formula for projecting the points located on π_∞. Since Ω_∞ is contained within π_∞, we can use this to express its projection.

The points on π_∞ are of the form $\mathbf{X} = (\mathbf{X}_1, \mathbf{X}_2, \mathbf{X}_3, 0)^\mathsf{T}$. Letting $\mathbf{d} = (\mathbf{X}_1, \mathbf{X}_2, \mathbf{X}_3)^\mathsf{T}$ allows us to write the projection of the point $\mathbf{X}$ using a general camera as:

$$\mathbf{x} = \mathtt{P}\mathbf{X} = \mathtt{KR}\begin{bmatrix}\mathtt{I} & -\widetilde{\mathbf{C}}\end{bmatrix}\mathbf{X} = \mathtt{KR}\mathbf{d}.$$

Thus, we see that the mapping between the plane at infinity and the camera image can be represented by homography $\mathtt{H} = \mathtt{KR}$.

Under a mapping $\mathbf{x}' = \mathtt{H}\mathbf{x}$, a conic $\widetilde{\mathbf{C}}$ transforms into $\widetilde{\mathbf{C}}' = \mathtt{H}^{-\mathsf{T}}\widetilde{\mathbf{C}}\mathtt{H}^{-1}$. Indeed, $\mathbf{x}'^\mathsf{T}\widetilde{\mathbf{C}}'\mathbf{x}' = (\mathtt{H}\mathbf{x})^\mathsf{T}\mathtt{H}^{-\mathsf{T}}\widetilde{\mathbf{C}}\mathtt{H}^{-1}\mathtt{H}\mathbf{x} = \mathbf{x}^\mathsf{T}\widetilde{\mathbf{C}}\mathbf{x}$. Thus, the absolute conic Ω_∞ is mapped to $\omega = (\mathtt{KR})^{-\mathsf{T}}\mathtt{I}(\mathtt{KR})^{-1} = \mathtt{K}^{-\mathsf{T}}\mathtt{R}\mathtt{R}^{-1}\mathtt{K}^{-1} = (\mathtt{K}\mathtt{K}^\mathsf{T})^{-1}$. □

Together with the claims about the absolute dual quadric, this result implies

$$\omega^{-1} = \mathtt{K}\mathtt{K}^\mathsf{T} = \mathtt{P}\mathtt{Q}^*_\infty\mathtt{P}^\mathsf{T},$$

which – in the case of simplified internal calibration matrix (4) – is equal to:

$$\mathtt{K}\mathtt{K}^\mathsf{T} = \begin{bmatrix}(fm_x)^2 & 0 & 0\\ 0 & (fm_x)^2 & 0\\ 0 & 0 & 1\end{bmatrix}.$$

In this way, these equalities allow us to introduce constraints on $\mathtt{Q}^*_\infty$ based on our simplified calibration matrix $\mathtt{K}$. In line with [23], we can write 4 linear equations from every projection matrix in our scene:

$$\begin{aligned}(\mathtt{P}\mathtt{Q}^*_\infty\mathtt{P}^\mathsf{T})_{11} - (\mathtt{P}\mathtt{Q}^*_\infty\mathtt{P}^\mathsf{T})_{22} &= 0\\ (\mathtt{P}\mathtt{Q}^*_\infty\mathtt{P}^\mathsf{T})_{12} &= 0\\ (\mathtt{P}\mathtt{Q}^*_\infty\mathtt{P}^\mathsf{T})_{13} &= 0\\ (\mathtt{P}\mathtt{Q}^*_\infty\mathtt{P}^\mathsf{T})_{23} &= 0\end{aligned}$$

This is a system of linear equations that allows us to solve for $\mathtt{Q}^*_\infty$. In the case of an overdetermined solution, this can again be done by taking the singular vector corresponding to the smallest singular value. The rank 3 constaint can be enforced using the SVD, as in section 3.7. Since a dual quadric $\mathtt{Q}^*$ transforms under homography $\mathtt{H}$ as $\mathtt{H}\mathtt{Q}^*\mathtt{H}^\mathsf{T}$, the correcting homography is obtained by decomposing the matrix as $\mathtt{Q}^*_\infty = \mathtt{H}\tilde{\mathtt{I}}\mathtt{H}^\mathsf{T}$, which can be carried out using eigenvalue decomposition (see for example [14, pg. 580]). Discussion of proper preconditioning of the linear system can be found in [24].

3.12 Robust estimation using RANSAC

In the previous sections, we already deal with a certain level of imprecision in the input data by assuming that it is corrupted by Gaussian noise. However,

normal distribution is not heavy-tailed and thus assigns nearly zero probability to completely outlying data. This implies that if the user (or automatic matching algorithm) accidentally marks, for example, a vertex incorrectly by 100px, it will substantially affect the resulting reconstruction since such measurement will be heavily penalised. In contrast to this, we would like our algorithm to be robust against such outliers.

Thankfully, the algorithms used to build the initial solution can be adapted so that they are robust and, as a side effect, even classify parts of the input as inliers or outliers. The general paradigm that can achieve this is called RANSAC, which is a short for *random sample consensus*[1].

To robustly estimate a quantity, for example the fundamental matrix `F`, from a strongly overdetermined set of measurements, we can proceed in the following way:

1. Randomly pick a minimal subset of the input from which the estimated quantity (e.g., `F`) can be determined.
2. Using this sample, estimate the quantity.
3. Go through all measurements and count how many of them approximately satisfy the obtained relation (for example, how many pairs of corresponding points indeed satisfy the epipolar constraint expressed by `F` within some precision).
4. Iterate steps 1-3 enough times and keep track of the sample that resulted in the highest number of inlying measurements.
5. Estimate the quantity once again, this time using all the inliers in the input data. Mark the rest of the input as outliers.

See [14, pg. 117] for a more detailed discussion of RANSAC, which includes a derivation of the number of iterations (step 4) necessary to achieve a certain probability (usually 99%) of picking a subset of the data unaffected by outliers.

[1] And also a word-play on the english word "ransack", which means "to search thoroughly".

Chapter 4

Image processing techniques

The process of manually marking projections of vertices on photographs is demanding and the precision rarely better than several pixels. In this chapter, we describe three methods – SIFT [20], MSER [10] and MSCR [11] – used to extract matches automatically. When applied to suitable photos, these algorithms can produce large number of correspondences, often with subpixel precision. All three techniques are sparse feature matching algorithms. Their typical output, a set of matches for a pair of images, is shown in figure 4.1.

Figure 4.1: Two views of a scene with a sculpture of Ema Destinnová. Corresponding points found using the SIFT algorithm of section 4.1 are connected by line segments.

Generally, to match two images of the same scene, these algorithms first extract features from the photos – for example corners or bright spots. Since the feature detection is highly repeatable, the majority of the features should be detected in both images. Each feature then has a descriptor assigned, typically a high-dimensional real vector. The descriptors are constructed in such a way that

they are (at least to a certain extent) affinely invariant. This implies that looking at a feature from different viewpoints should result in the same (or very similar) descriptor. Thanks to that, we can match pairs of photos by taking every feature from the first photo and finding the feature with the nearest descriptor in the second one.

This approach can be contrasted with dense matching algorithms, such as [25], which try to match every image pixel, typically assuming that the scene has some level of spatial consistency. Such amount of matches is an advantage for example for visualization of the scene. Camera calibration on the other hand requires smaller amount of reliable and precise matches. For this reason, we focus our attention to sparse feature matching algorithms.

4.1 Scale-invariant feature transform

SIFT [20] is a very popular algorithm to generate image matches, commonly used in software packages for panorama stitching, HDR photography, etc. Matches obtained by this algorithm have been successfully used as input for camera calibration in [26] and [6]. It produces very good results when the matched images do not substantially differ in their orientation. This is because SIFT uses descriptors that are invariant to rotation, scale changes, translation, illumination changes and noise, but only partially invariant to a general affine transformation of the image.

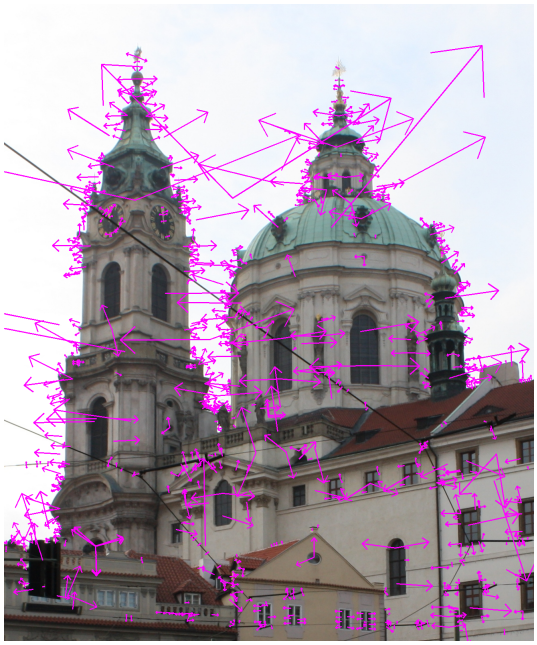


Figure 4.2: Features detected using SIFT. Each one is denoted by an arrow going from feature's location in the direction of its local orientation. The arrow's size is proportionate to feature's local scale.

It proceeds by taking a grayscale version of the image and extracting feature points corresponding to centers of blobs in the photo. Local orientation and scale is then assigned to each feature and invariant descriptors constructed using these local frames. An example of the resulting set of features with their local orientations and scales represented using arrows is in figure 4.2.

The feature detection is done using *scale-space*, which is a series of images constructed by taking the original one and blurring it by subsequently larger and larger Gaussian kernels. Refer to figure 4.3 for an example. At some point, the image will get so blurred that it doesn't make sense to perform the operations in the original resolution. When this happens, we can downsize the image to 50%, which speeds up scale-space generation and the operations to be performed on it.

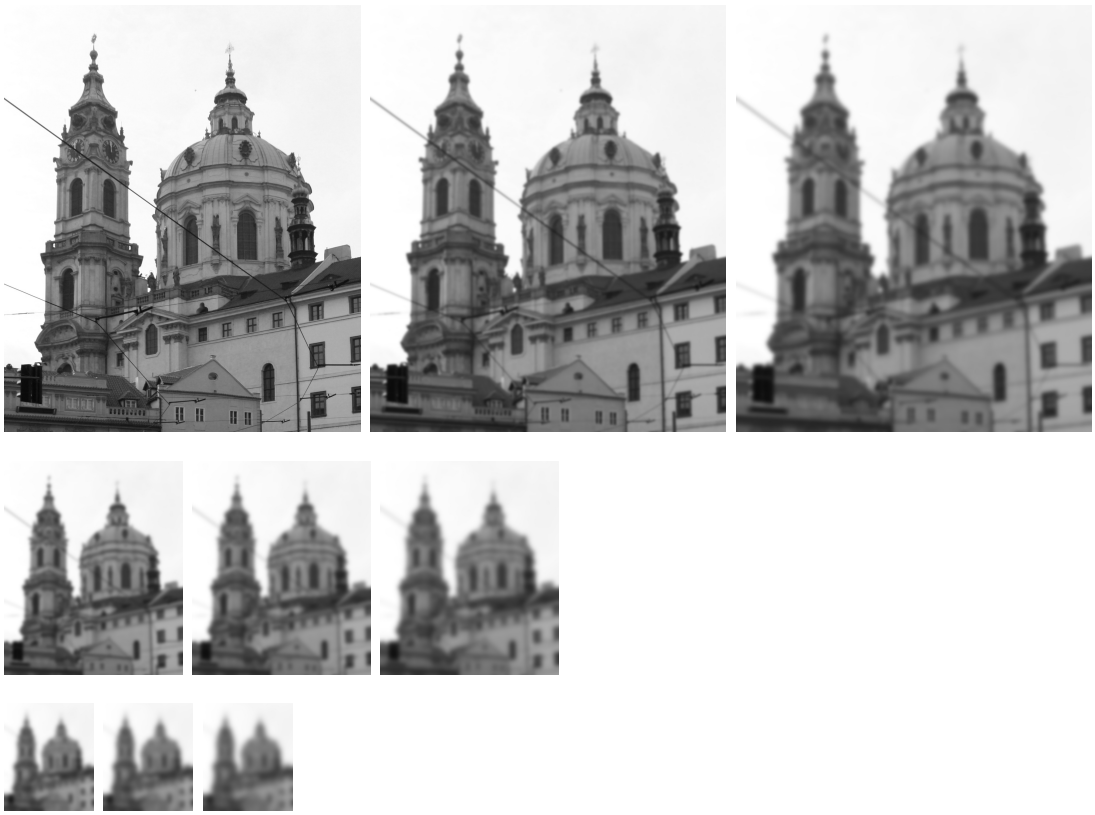


Figure 4.3: Scale-space consists of subsequently more and more blurred images.

Neighbouring images in scale-space are then substracted. See figure 4.4 for an example of how the resulting image could look like. For images blurred by Gaussian kernels σ_1 and σ_2, the substraction results in a difference-of-Gaussian image D:

$$D(x, y) = (G(x, y, \sigma_1) - G(x, y, \sigma_2)) * I(x, y),$$

where $*$ is convolution operator, I the grayscale image and G Gaussian kernel:

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

As is shown in [20], the difference-of-Gaussian is connected to the Laplacian of Gaussian and the difference image can thus be informally thought of as a second derivative of the original image.

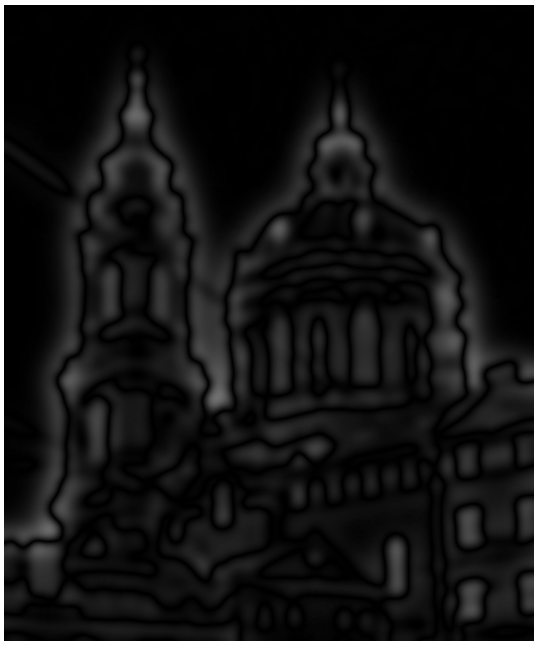


Figure 4.4: The difference of two neighbouring images in scale-space. The resulting image's intensity is proportionate to the absolute value of the difference.

We then take the resulting series of difference images and find local extrema in it – pixel positions, where the difference has a value strictly higher (or lower) than all its neighbouring pixels (and also the corresponding pixels on neighbouring difference images). Observe on figure 4.4 how the difference image has bright spots in places where the original image has blobs of a certain size. This favored size depends on how large was the convolution kernel used to generate the original pair of blurred images: a difference image created by substracting a pair of images from the end of the scale-space has local extrema in places where large blobs can be seen in the original photo. In this way, we obtain points of interest corresponding to blobs of all sizes.

The neighbourhood of each pixel of minimal or maximal difference is then fitted by a quadratic function. By calculating extrema of these quadratic functions, we obtain their positions with subpixel precision. Also, when the corresponding quadratic function is shallow or narrow, the feature point is rejected as unstable. Such features have high probability of being affected by noise (the case of shallow quadratic function) or of corresponding to edges in the scene (narrow function), which we do not want to detect, since their appearance varies with different viewpoints.

When the points of interest are computed, we look at the original image and for every feature calculate the image gradient at its position, which establishes local orientation and scale of the feature.

In the second part of the computation, each feature is assigned a descriptor, which characterises the appearance of the feature's image neighbourhood. This descriptor is constructed using the established feature scale and orientation and thus doesn't change after application of translation, rotation or scaling to its image. Research in biology and psychology lead to the conclusion that image gradients are of special importance for human object recognition. SIFT descriptors are based on this research and basically are just histograms of gradient directions around feature's position.

To create a descriptor for a feature, we first compute gradients for pixels around its location (in the scale nearest to the one of the feature). Magnitudes of these regions are then weighted by a Gaussian window centered on the feature's position. The samples are then divided into 4×4 subregions and for each subregion a histogram of 8 gradient directions is calculated. The descriptor is formed by concatenating values from these histograms into 128-dimensional real vector. See [20] for more details about the descriptor construction, such as achieving invariance to illumination changes.

Matching features of an image pair is typically done by taking each feature in one image and performing a nearest neighbour search (in L_2-norm) through the descriptors of features in the other photo. It is advisable to also consider the feature with the second nearest descriptor. If the distance of this second candidate descriptor is similar to the nearest one, we can't reliably decide which of the two candidates is the match. Thus, a match is typically accepted only if the ratio of the first and second descriptor distances is less than 0.8. This is also elegant way to avoid matching repeating scene features that commonly result in mismatches. According to [20], this technique (called *feature space outlier rejection*) rejects 90% of the false matches while discarding only 5% of the correct ones.

4.2 Maximally stable extremal regions

Maximally stable extremal regions are image regions that are significantly brighter or darker than their neighbourhood. Their detection can be done in a highly repeatable way so that the same regions are detected even when their image is deformed by any affine (or even any continuous) transformation. They were introduced in [10] for grayscale images and eventually generalised for color images as MSCRs [11]. The figure 4.5 shows an input image and the detected regions overlaid over its grayscale version.

Suppose that the grayscale image is represented by a function $I : \Omega \to [0..255]$, where $\Omega = [1..W] \times [1..H]$ is the set of all pixel coordinates. We now describe a watersheddding process used to detect MSERs in I.

For a threshold $t \in [0..255]$, we divide the set of pixel positions into sets $\mathbf{B}$ (black) and $\mathbf{W}$ (white):

$$\mathbf{B} = \left\{\mathbf{x} \in \Omega^2 : I(x) \leq t\right\}, \mathbf{W} = \Omega^2 \setminus \mathbf{B}.$$

Our algorithm starts with $t = 255$. All pixels are thus contained in the set

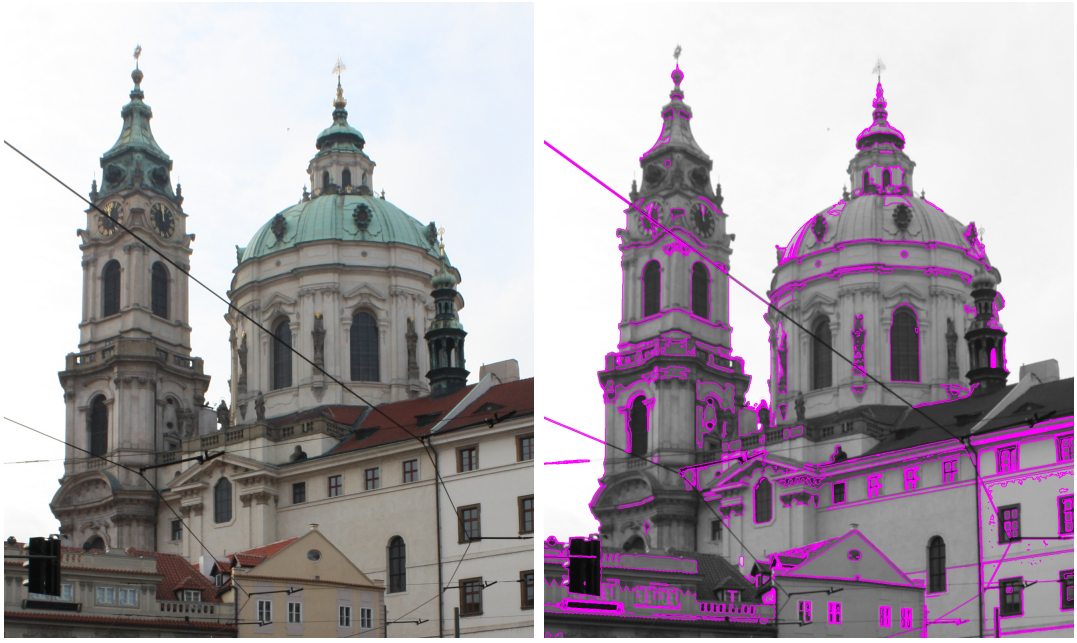


Figure 4.5: Input image for MSER detection (left) and the slightly blurred grayscale version of the image used during the computation with the contours of the resulting set of regions highlighted (right).

B and the set **W** is empty. The corresponding thresholded image is entirely black. When we start to decrease t, white spots start to appear and grow as more and more pixels fall above the threshold level. Individual regions will start to merge and eventually form one white region covering the entire image at the end of the computation. All pixel positions then belong to the set **W**. Figure 4.6 demonstrates this evolution.

White and black regions in the individual thresholded images are called *extremal regions*. From the set of all regions, we want to pick only the ones that have the highest probability of being detected on different views of the scene. MSER algorithm achieves this by choosing the regions that do not substantially change their size across several intensity threshold levels. Specifically, the algorithm looks at the evolution of each region and observes how fast it grows in each step. When it grows less than a given percentage of area for at least a given number of levels, region's contour from the step of minimal growth is saved.

To obtain only the most stable regions, several filtering methods can be applied. The input image is usually blurred using Gaussian kernel to account for noise. Large regions are typically immediately discarded, since they have low probability of being planar, which makes constructing descriptors for them difficult. Finally, long and narrow regions are again rejected, since it is improbable that they can be seen without occlusions on other views of the scene.

Details about MSER algorithm can be found in the original article [10].

While the algorithm from the previous section has a standard generally used

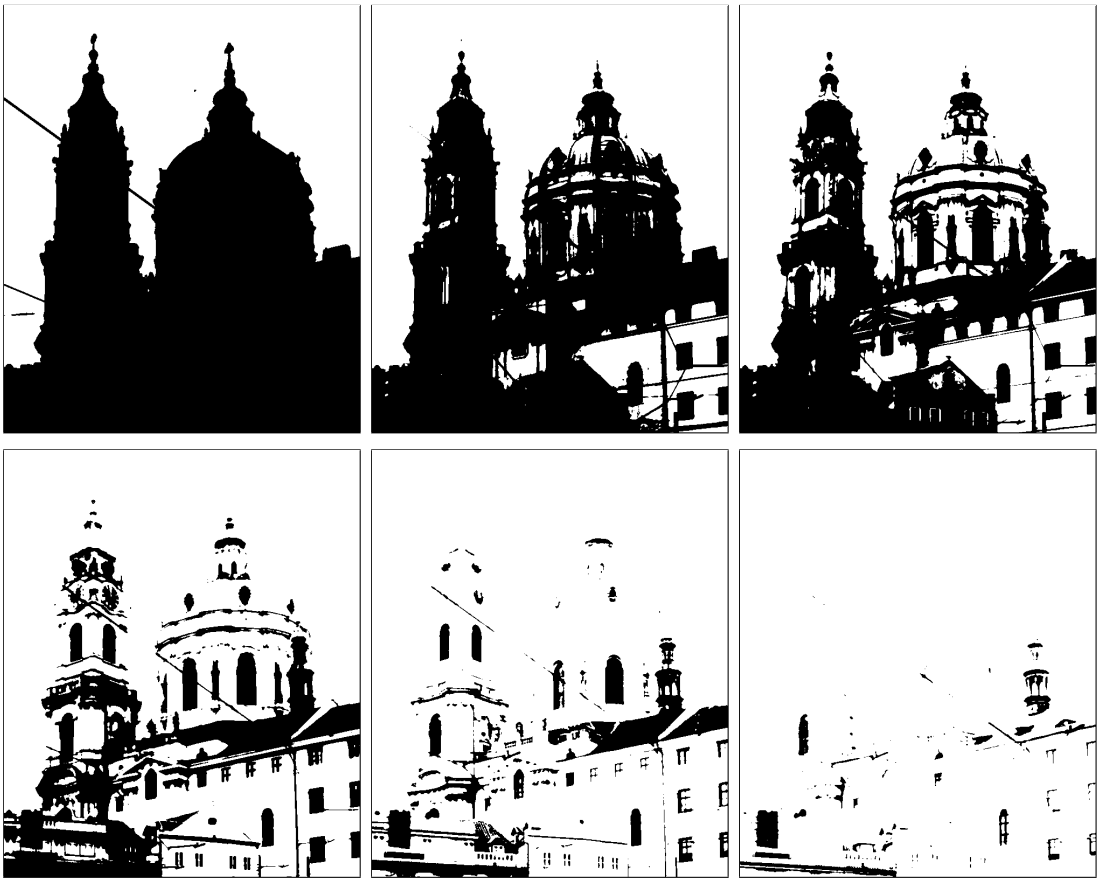


Figure 4.6: The input image from figure 4.5 thresholded by decreasing t. Observe how the building's windows form very stable regions clearly visible on several thresholded images.

descriptor available, the issue of matching MSERs is more complicated. One suggested approach is to reuse the SIFT descriptor [12] to represent the region's shape or texture. The matching method introduced in [21] uses an affine normalization of the region which, coupled with orientation assignment, allows to use standard intensity-normalised cross-correlation for the matching.

4.3 Maximally stable color regions

The MSER regions are detected in a grayscale image which has the disadvantage that regions with different colors (for example, red and green) can be assigned similar intensity and subsequently not being recongnised as separate regions. This leads us to a generalisation of MSER for color images, introduced as MSCR in [11], which we now briefly introduce.

We again take the image function I, now defined as $I : \Omega \to \mathbb{R}^3$. In this way, the image function assignes RGB color values to all pixel positions Ω. We also

define a graph G with image pixels as it's vertices and edges E defined in the following way (note that $\mathbf{x}$ and $\mathbf{y}$ are 2-dimensional vectors):

$$E := \big\{\{\mathbf{x}, \mathbf{y}\} \in \Omega^2 : |\mathbf{x} - \mathbf{y}| = 1\big\}$$

where $|\mathbf{x} - \mathbf{y}|$ is a Manhattan or Euclidean distance of pixel coordinates. The edges thus connect the neighbouring pixels. Finally, we assign a weight $g(\mathbf{x}, \mathbf{y})$ to every edge. In line with [11], we use the *Chi squared measure* to calculate the weight between neighbouring pixels $\mathbf{x}$ and $\mathbf{y}$:

$$g^2(\mathbf{x}, \mathbf{y}) = \sum_{k=1}^{3} \frac{(I_k(\mathbf{x}) - I_k(\mathbf{y}))^2}{I_k(\mathbf{x}) + I_k(\mathbf{y})},$$

where $I_k(\mathbf{x})$ represents the value of the k-th color channel of the pixel $\mathbf{x}$.

The evolution process for color images is demonstrated in figure 4.7 and proceeds in the following way. We consider a series of subgraphs of G with edge sets $E_t \subseteq E$. We denote this subgraph as G_t. The connected components of G_t are referred to as regions. We start with $E_t, t = 0$ and gradually increase t. As we do this, new edges start to appear in the subgraph G_t and regions grow and merge. Analogously to the MSER algorithm, the detected stable regions are the ones nearly unchanged in size across several thresholding levels t.

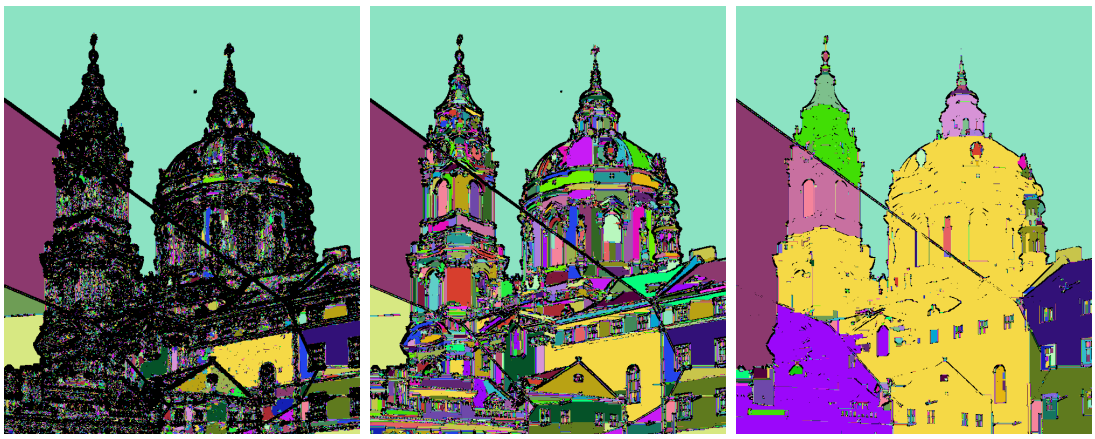


Figure 4.7: MSCR evolution process for the input image from figure 4.5. Regions for increasing edge weight thresholds are depicted using different colors, with trivial single pixel regions being left black.

Because MSCRs are based on color and not only on the combined intensity of all three color channel values, they are less affected by shadows in the scene, which in the case of MSER detection often split the affected region into several. Recommendations for region filtering mentioned at the end of the previous section apply to MSCR regions as well. The same matching methods used for MSERs can be utilized to establish correspondences also between MSCRs.

Chapter 5

Semi-automatic approach

The principal theme of this thesis is using image processing techniques to help the user reconstruct 3D scenes as fast as possible. The most demanding and time-consuming part of the workflow seems to be entering correspondences for camera calibration, since they have to be extremely precise to achieve a quality reconstruction. We thus turn our attention to obtaining these matches automatically and adjusting the camera calibration algorithms so that they are robust against gross outliers which often arise when using automatic image registration. When the matches are available, they can be also used to estimate the positions of objects on photos where they haven't been marked yet. This is demonstrated in section 5.4, where we show that it is possible to automatically place a polygon on photos after it has been marked on at least one of them.

Aa a result of this thesis, a complete software solution for image-based modeling has been created. This open source project, called *insight3d*, is aimed at general users who wish to reconstruct their favorite objects or scenes in 3D from several photos. The program can be freely downloaded from the project's website[1]. Versions for both Windows and Linux are available.

5.1 Matching an image pair

We now descibe details of the method used to obtain matches for an image pair. We choose to use the SIFT algorithm [20] described in section 4.1 because of its well-defined and well-tested feature descriptors. As stated in the previous chapter, SIFT keypoints are assigned a 128-dimensional vector as a descriptor and feature matching is performed using the nearest neighbour search among the features in the second image.

However, a typical 4Mpx image results in roughly 2-15 thousand features and such high-dimensional nearest neighbour search is theoretically problematic. Taking every feature in the first image and performing exaustive search over all descriptors in the second image results in several minutes long computation times

[1] `http://insight3d.sourceforge.net/` – its offline version is located on the attached DVD

(on a typical computer with 2GHz single-core CPU). Since the task is trivially parallelizable, multiple cores and SIMD instruction sets like SSE2 can be easily used to speed up the computation, but the required time is still unreasonably long since we typically want to match every possible pair of photos, which results in quadratic number of matching rounds. General approximate nearest neighbour search (using a library such as ANN [1]) is also problematic, since these techniques generally work effectively only when the vectors have a dimensionality of roughly up to 10. For our 128-dimensional problem, typical approximate algorithms result in unacceptable running times even for large values of the tolerated error. However, as described in [20], approximate search using kd-trees can be made effective by limiting the number of explored kd-tree nodes to, say, 200. This results in sometimes returning a strongly suboptimal vector. Since the algorithms in computer vision are typically robust to small number of outliers, this doesn't cause significant problems. Typically, this brings the time necessary to match an image pair down to roughly 10-20 seconds.

5.2 Guided by fundamental matrices

We found that both the speed of the image-pair matching algorithm and the number of correspondences returned can be significantly improved by appropriately using a technique called *guided matching*.

The main idea is to perform the matching in the typical way and then use the correspondences to estimate the fundamental matrix for the image pair. The matching is then repeated, but this time for each feature from the first image we consider only the points that lie on the estimated epipolar line in the other image as the matching candidates. This results in a higher number of matches, since it is less likely that the match will fail feature space outlier rejection described at the end of section 4.1 (two features with similar descriptor would have to be located on the same epipolar line).

To increase the matching speed, we have implemented this technique in the following way. First, enough correspondences to robustly estimate the fundamenal matrix are obtained using standard matching approach based on kd-trees. We do not try to match all features, since that would take away all the computational time that we try to reduce. The fundamental matrix for this partially matched image pair is then estimated. After this, we divide the second image into *buckets* forming a 2D grid. Each bucket has the features that lie in it assigned. When matching a feature from the first image, we compute its epipolar line and check which buckets lie near it and thus have a chance of containing potential matches. We then iterate through these buckets and try to match the original feature against the ones located inside each of the selected buckets. We have empirically found that buckets of size 110px result in optimal running time for typical 4Mpx photos. This reduces the matching time to several seconds.

5.3 Camera calibration

After the images are matched and tracks extracted, we need to calibrate the cameras. This is potentially difficult, since the input data contain some amount of gross outliers and the computation is inherently numerically unstable.

To achieve certain level of robustness, we arranged the individual algorithms described in sections 3.7 to 3.11 in the following way:

First, we need to bootstrap the construction of the initial solution by calculating projection matrices for a pair of images using the fundamental matrix. We take a look at the image pairs in our dataset and sort them according to the number of correspondences in decreasing order. We then randomly pick one of the top n image pairs (n is a parameter of our procedure). After that, we estimate the fundamental matrix using the linear algorithm from section 3.7. To robustify the computation, the RANSAC paradigm is applied. Finally, we triangulate the vertices visible on both of our images.

If there still are uncalibrated photos in our dataset, we perform camera resection (described in section 3.9). We sort the uncalibrated photos according to the number of vertices with estimated position and randomly pick one of the top n photos. The projection matrix of this photo is then estimated using camera resection.

After each calibration step, we recalculate the positions of the vertices to take into account the newly gained knowledge. Triangulation is performed using the algorithm described in section 3.8, with the provision that the RANSAC paradigm is again applied to robustify the computation. As mentioned in section 3.12, this gives us also the estimate of which measurements are inliers and which outliers. This information is saved and the following steps of the calibration are then performed only using the inlying measurements.

The algorithm continues by resecting new photos and reestimating the vertices until there are no uncalibrated photos. To achieve better stability, several iterations of the nonlinear optimization described in section 3.10 are performed with a certain probability (another parameter of the procedure) after each step. At the end of the computation, the constructed initial solution is refined using nonlinear optimization and projective reconstruction is obtained.

Autocalibration is then carried out. Here, we again apply RANSAC using the assumption that the cameras have principal point located in the middle of the image. This is done by picking three cameras randomly, estimating the metric structure of the scene from the constraints described in section 3.11, applying correcting homography to all cameras and extracting the resulting internal calibration matrix from each corrected projection matrix. The quality of the autocalibration result is then judged by the number of cameras that really do have the principal point located in the middle of the image (with some tolerance, typicaly at most several hundred pixels away from the image center). Such application of RANSAC to autocalibration ensures that the metric structure was not established incorrectly due to incorrectly estimated cameras in the projective reconstruction.

The randomization of the initial solution construction helps when there is a danger that at some point the input data for one of the elementary algorithms are degenerate (see [14] for details about degenerate configurations). In such a case, the result is not properly constrained and the computation results in incorrect output. When this happens, the user still has a chance of obtaining the correct solution by running the calibration algorithm again, hoping that the (probably different) computation path will not lead to a degenerate configuration again.

A demonstration of the precision we are able to obtain using our calibration method can be found in figure 5.1.

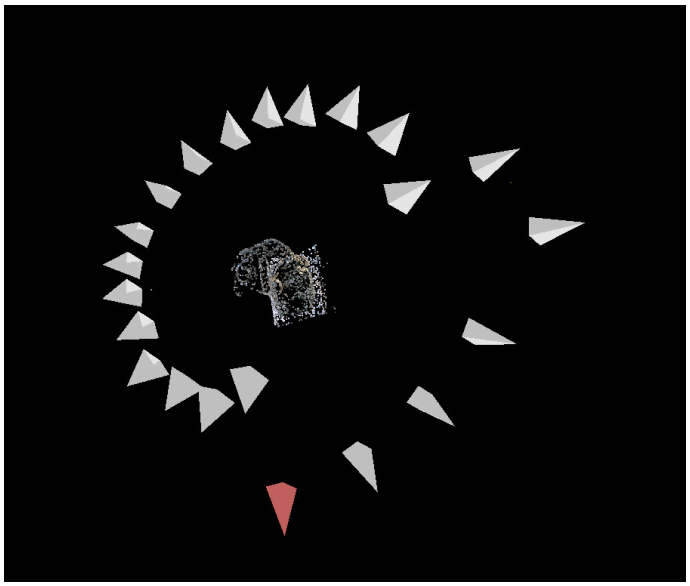


Figure 5.1: Calibration of a scene where camera moving in a circle around a sculpture. Automatically reconstructed point cloud is seen in the middle. The first few photos were taken from greater distance using longer focal length. The circle is properly closed even though the first and the last photo in the sequence were not matched and thus had no correspondences between them. Build-up error thus doesn't significantly affect the calibration.

5.4 Transfer of a polygon using homography

Suppose that the user marked several vertices on one image and joined them into a polygon. Suppose further that the images have been matched and the automatically obtained matches cover more or less the whole photo. This allows us to help the user mark this polygon on other images. We can take the automatically generated points from the inside of the polygon and look where they are located on another image.

Since the polygon should be planar, it should be possible to transfer its point on the other image using a planar homography `H`. The estimation of the homography is analogous to the estimation of the projection matrix during the camera

Figure 5.2: The polygon in the image on the left has been marked manually. When the user of insight3d switches to an image where the polygon is not fully marked and presses *spacebar* key, the application uses robust estimation to calculate the position of the polygon on the current image.

resection (see section 3.9). Once again, we apply RANSAC paradigm to estimate the homography robustly. An example of a polygon and its automatically estimated counterpart is in figure 5.2. If the user already marked one or more of the polygon's vertices on the second image, we enforce the position of these vertices as a strong constraint (thus, their position is not changed).

Chapter 6

Future work

We have described and implemented methods used to reconstruct 3D scenes from images. Sparse feature matching algorithms have been employed and the camera calibration algorithm robustified so that completely automatic calibration and generation of a precise 3D pointcloud of the scene is possible. The software solution was successfully tested on datasets obtained by both compact digital cameras and both APS-C and full-frame DSLRs.

Possible future improvements are twofold:

The camera calibration algorithm can be further improved and robustified. To achieve better precision, projective bundle adjustment and autocalibration should be followed by the Euclidean bundle adjustment – that is, a nonlinear optimization in which the cameras have constraints like zero skew strongly imposed. This might help when the scene is in a near-degenerate configuration (e.g., is nearly planar). Furthermore, degenerate configurations could be detected automatically using the covariance matrix, Monte Carlo estimation or by checking the second smallest singular value in the SVD. Implementing MSER or MSCR detection could help when matching photos of poorly textured scenes or when there are significant rotational changes between views.

A significant improvement in terms of user friendliness would be to implement Canoma-style modeling (described in section 2.4), where the user can add primitives like cubes or pyramids into the scene. Adding objects with known geometry (and Euclidean properties) would have significant advantages for autocalibration, since these properties could be used to estimate the metric structure of the scene. It would also help the user create complete reconstructions of symmetric objects even when they are not visible from all sides. Since this happens very often in practice, implementing the ideas of [8] should be a priority for the future development of the project.

Bibliography

[1] ANN – Approximate Nearest Neighbour Library; retrieved on 2009-08-01. http://www.cs.umd.edu/~mount/ANN/.

[2] Autodesk ImageModeler; retrieved on 2009-08-01. http://www.autodesk.com/imagemodeler.

[3] Autodesk Maya; retrieved on 2009-08-01. http://www.autodesk.com/maya.

[4] Canoma (unofficial website of this discontinued software); retrieved on 2009-08-01. http://www.canoma.com/.

[5] Eos PhotoModeler. http://www.photomodeler.com/.

[6] M. Brown and D. G. Lowe. Unsupervised 3d object recognition and reconstruction in unordered datasets. pages 56–63, 2005.

[7] Robert Burtch. History of photogrammetry; retrieved on 2009-08-01. http://www.ferris.edu/faculty/burtchr/sure340/notes/History.pdf, 2008.

[8] Paul E. Debevec. *Modeling and Rendering Architecture from Photographs*. PhD thesis, University of California at Berkeley, Computer Science Division, Berkeley CA, 1996.

[9] Chris Engels, Henrik Stewénius, and David Nistér. Bundle adjustment rules. 2006.

[10] Maximally Stable Extremal, J. Matas, O. Chum, M. Urban, and T. Pajdla. Robust wide baseline stereo from. In *In British Machine Vision Conference*, pages 384–393, 2002.

[11] Per-Erik Forssén. Maximally stable colour regions for recognition and matching. In *IEEE Conference on Computer Vision and Pattern Recognition*, Minneapolis, USA, June 2007. IEEE Computer Society, IEEE.

[12] Per-Erik Forssén and David Lowe. Shape descriptors for maximally stable extremal regions. In *IEEE International Conference on Computer Vision*, volume CFP07198-CDR, Rio de Janeiro, Brazil, October 2007. IEEE Computer Society.

[13] Roland W. Freund and Florian Jarre. Solving the sum-of-ratios problem by an interior-point method. *J. of Global Optimization*, 19(1):83–102, 2001.

[14] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521540518, second edition, 2004.

[15] Richard I. Hartley and Fredrik Kahl. Optimal algorithms in multiview geometry. In *ACCV (1)*, pages 13–34, 2007.

[16] Richard I. Hartley and Peter Sturm. Triangulation. In *In Proceedings of ARPA Image Understanding Workshop*, pages 957–966, 1994.

[17] Berthold K. P. Horn. Projective geometry considered harmful, 1999.

[18] Fredrik Kahl, Sameer Agarwal, Manmohan Krishna Chandraker, David Kriegman, and Serge Belongie. Practical global optimization for multiview geometry. *Int. J. Comput. Vision*, 79(3):271–284, 2008.

[19] M.I.A. Lourakis and A.A. Argyros. The design and implementation of a generic sparse bundle adjustment software package based on the levenberg-marquardt algorithm. Technical Report 340, Institute of Computer Science - FORTH, Heraklion, Crete, Greece, Aug. 2004. Available from `http://www.ics.forth.gr/~lourakis/sba`.

[20] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60:91–110, 2004.

[21] Jiri Matas, Stepan Obdrzalek, and Ondrej Chum. Local affine frames for wide-baseline stereo, 2002.

[22] Hedvika Peroutková. Phototextures extraction for 3d photometric reconstruction. Master's thesis, MFF UK, Praha, 2008.

[23] Marc Pollefeys, Reinhard Koch, Luc, and Luc Van Gool. Self-calibration and metric reconstruction in spite of varying and unknown intrinsic camera parameters. pages 90–95, 1998.

[24] Marc Pollefeys, Frank Verbiest, and Luc J. Van Gool. Surviving dominant planes in uncalibrated structure and motion recovery. In *ECCV '02: Proceedings of the 7th European Conference on Computer Vision-Part II*, pages 837–851, London, UK, 2002. Springer-Verlag.

[25] Sébastien Roy and Ingemar J. Cox. A maximum-flow formulation of the n-camera stereo correspondence problem. In *ICCV '98: Proceedings of the Sixth International Conference on Computer Vision*, page 492, Washington, DC, USA, 1998. IEEE Computer Society.

[26] N. Snavely, S. M. Seitz, and R. Szeliski. Modeling the world from Internet photo collections. *International Journal of Computer Vision*, 80(2):189–210, November 2008.

[27] Bill Triggs, Philip F. McLauchlan, Richard I. Hartley, and Andrew W. Fitzgibbon. Bundle adjustment - a modern synthesis. In *ICCV '99: Proceedings of the International Workshop on Vision Algorithms*, pages 298–372, London, UK, 2000. Springer-Verlag.

Appendix A

Guidelines for optimal results

The photos to be used in insight3d can be taken from arbitrary positions and there is no need to perform any measurements when making them. The application figures all of that automatically. However, several important rules should be remembered to achieve good results. Photos in figure 2.4 are an example of a dataset that allows for a precise detailed reconstruction.

- **The photos should be focused** (not blurred).
- There should be **large overlaps between neighbouring photos**. Note how most of what can be seen on any of the photos in figure 2.4 can also be seen on the next one.
- **Planar scenes should be avoided.** When everything on the photo lies on a single plane in space, like on the following photo, insight3d can't properly determine the focal length.

- There should be only **limited angle difference** between subsequent photos. When moving around an object, the user shoould take a photo every

15-25 degrees. Again note the photos 2.4 and how with each photo we look at the building from only slightly different angle.

- It is optimal to shoot scenes with lots of unique **details and textures**. Old buildings and sculptures can be processed almost routinely, while blank walls and clean cars have low chance to be automatically matched. But if automatic matching fails, insight3d offers tools to enter matches manually.
- **The photos shouldn't be cropped.** The application needs to know where was the original image center.
- **Rule of 3.** Every part of the scene to be reconstructed should be visible on at least 3 photos.

Appendix B

Software architecture

The algorithms described in this work have been implemented in a software package written in C++, which is available (along with the user documentation) on the attached DVD and can be also downloaded from the internet. The source code is licensed under the GNU AGPL 3.0 license. Below is a brief description of the organization of the code into modules and individual source files.

- **Core**
  `core_*.cpp`
  - Handles basic interaction with the operating system.
  - Provides routines and macros for creating, updating and releasing dynamic structures.
  - Defines additional basic math functions.
  - Error management.
- **Geometry**
  `geometry_*.cpp`
  - Defines structures to store vertices, points, meta-data about images, polygons, calibrations and indexing structures.
  - Provides methods to manipulate these structures and validate the consistency of their content.
- **MVG**
  `mvg_*.cpp`
  - Provides implementation of multiple view geometry algorithms, such as the camera resection, triangulation, finite camera decomposition, data normalization and autocalibration.
  - These libraries have only one dependency – the OpenCV library, which is commonly used in computer vision programs. Moreover, all functions take their arguments in the manner common to generic OpenCV

routines. This part of the source code is thus readily reusable in other computer vision programs.

- **Tool framework**
  `tool_core.cpp`
  - Provides routines for creating tools such as the "Selection and zoom" tool or the "Points creator" tool. This separates the source code of tools from routines provided by the GUI library we decided to use. The resulting code is therefore more readable, shorter and independent of the chosen GUI toolkit.

- **Tools**
  `tool_*.cpp`
  - Every tool available to the user has a separate source file. The tools defined in the current version are: `coordinates` (aligns coordinate system with selected camera), `extrude` (extrudes polygons to automatically detected ground), `matching` (performes automatic sparse matching of images), `plane_extraction` (robustly finds main plane in the pointcloud of the model), `points` (lets the user mark corresponding points on submitted images), `polygons` (used to join vertices into polygons), `resection` (camera resectioning), `selection` (used to select points), `triangulation` (triangulates the position of the vertices from their projections images obtained by calibrated cameras).

- **UI**
  `ui_*.cpp`
  - Provides routines for displaying the content of the data structures in OpenGL window. Separate source files are devoted to rendering the context popup window (which displays thumbnails from other images), selection boxes, symbols for points, point clouds, etc.
  - UI module also handles the events sent by the GUI toolkit and eventually sends the events for example to the tool framework.