Computational Tools for Stereo and Light Field Photography

Inauguraldissertation
der Philosophisch-naturwissenschaftlichen Fakultät
der Universität Bern

vorgelegt von

Daniel Donatsch

von Meilen

Leiter der Arbeit:

Prof. Dr. M. Zwicker
Universität Bern

Prof. Dr. K. Hormann
Università della Svizzera Italiana

Computational Tools for Stereo and Light Field Photography

Inauguraldissertation
der Philosophisch-naturwissenschaftlichen Fakultät
der Universität Bern

vorgelegt von

Daniel Donatsch

von Meilen

Leiter der Arbeit:

Prof. Dr. M. Zwicker
Universität Bern

Prof. Dr. K. Hormann
Università della Svizzera Italiana

Von der Philosophisch-naturwissenschaftlichen Fakultät angenommen.

Bern, 6.10.2015 **Der Dekan:**
Prof. Dr. G. Colangelo

Abstract

This thesis covers a broad part of the field of computational photography, including video stabilization and image warping techniques, introductions to light field photography and the conversion of monocular images and videos into stereoscopic 3D content.

We present a user assisted technique for stereoscopic 3D conversion from 2D images. Our approach exploits the geometric structure of perspective images including vanishing points. We allow a user to indicate lines, planes, and vanishing points in the input image, and directly employ these as guides of an image warp that produces a stereo image pair. Our method is most suitable for scenes with large scale structures such as buildings and is able to skip the step of constructing a depth map.

Further, we propose a method to acquire 3D light fields using a hand-held camera, and describe several computational photography applications facilitated by our approach. As the input we take an image sequence from a camera translating along an approximately linear path with limited camera rotations. Users can acquire such data easily in a few seconds by moving a hand-held camera. We convert the input into a regularly sampled 3D light field by resampling and aligning them in the spatio-temporal domain. We also present a novel technique for high-quality disparity estimation from light fields. Finally, we show applications including digital refocusing and synthetic aperture blur, foreground removal, selective colorization, and others.

Acknowledgements

First of all, I thank my adviser Matthias Zwicker. This Thesis would not have been possible without his guidance whenever I lost track, his ideas whenever I got stuck and his experience when it came to presenting my research. Further I'm deeply thankful that he gave me the chance to work in his lab and finally grew from a mathematician into a computer scientist. I'm very glad I could work with him.

The daily life wouldn't be pleasant without great colleagues. I thank all current and previous CGG members, to make this lab such a nice work place. Ill always keep good memories of all these people. Wan-Yen Lo, Claude Knaus and Sonja Schaer were already in the lab when I arrived and introduced me to the University of Bern and the life as a PhD student. Fabrice Rousselle made each lunch talk way more interessting and funny. Daljit Singh Dhillon never became tired of discussing and explaining structure from motion topics. Peter Bertholet and Marco Manzi joined me for some of the most exciting concerts I've been and showed up irregularly in my indoor cycling class. Shihao Wu joined my indoor cycling class regularly making me proud and happy that I could convince at least one computer scientist that sport is good for the mind, too. Finally, Dragana Esser, who took care of our well-being, found a solution for each administrational problem and served us regularly home made cake and bread.

I had the pleasure to advice several Bachelor and Master theses. From each of the students I learnt something, which made this an exciting part of my work. I'd like to highlight here Daniel Frey and Marcel Zingg, which both built tools which I have used a lot. Further I thank Nico Färber for implementing my ideas which lead to the first paper. The most important student was Siavash Bigdeli. Without his help, which went far beyond coding, my second paper would not have been possible.

I thank Kai Hormann for being the co-referee of this thesis and for his helpful input for finalizing the text you are reading now.

The sys-admin Peppo Brambilla kept not only the machines running, he took me on many runs during lunch breaks which helped to

free memory in my brain.

I thank Felix Frey for his friendship and the images he provided for our first paper.

Michael Amrein spent many hours explaining me queer mathematical formulas and theorems during our math studies. Without him I maybe would not have finished my ETH diploma and the door for a PhD would never have opened.

Last but not least, I thank my girlfriend Yasmin Köller, my parents Andrea and Hansjürg Donatsch and my sister Claudia Donatsch and her husband Thomas Brülisauer for supporting me. It helped a lot to know that they stand behind me and love me. Additionally, I thank my parents for having given me the opportunity to go to the university and for teaching me education as an important value. Finally I would like to thank Yasmin for the many hours she spent for proofreading this thesis.

Contents

1 Introduction **5**
- 1.1 Contributions 8
- 1.2 Thesis Organization 10

2 Video Stabilization **13**
- 2.1 Feature Points 15
  - 2.1.1 Corner Points 16
  - 2.1.2 Kanade–Lucas–Tomasi Feature Tracker 18
  - 2.1.3 SIFT 18
- 2.2 Structure from Motion 21
  - 2.2.1 Projective Geometry 22
  - 2.2.2 Camera Model 28
  - 2.2.3 Two-View Geometry 31
  - 2.2.4 Fundamental Matrix 35
  - 2.2.5 Finding Camera Matrices 38
  - 2.2.6 Projective Reconstruction 39
  - 2.2.7 Finding the Fundamental Matrix 41
  - 2.2.8 n-View Geometry 43
- 2.3 3D Video Stabilization 45
- 2.4 Subspace Constraint 47
- 2.5 Subspace Video Stabilization 51
  - 2.5.1 Filtering with Subspace Constraint 52
  - 2.5.2 Moving Factorization 53
- 2.6 Other Methods 56

2.6.1 Video Stabilization using Epipolar Geometry . 56
2.6.2 Bundled Camera Paths for Video Stabilization 59
2.7 Stereo Video Stabilization 60
2.7.1 3D Stereo Video Stabilization 61
2.7.2 Subspace Stabilization for Stereo Videos 62

3 2D to 3D Conversion 67
3.1 3D Perception 68
3.2 3D Displays 74
3.3 2D to 3D Conversion Pipeline 74
3.4 Depth and Disparity Maps 76
3.4.1 Multi-View Stereo 77
3.4.2 User Input 80
3.4.3 Machine Learning 82
3.5 Creating Output Images 83

4 Image Warps 87
4.1 Basics of Image Warping 88
4.2 Optimization Techniques 94
4.2.1 The Least Squares Problem 95
4.2.2 Linear Least Squares Problems 96
4.2.3 Non-Linear Least Squares Problems 97
4.3 Image Warping Applications 103
4.3.1 New Viewpoint for Given Images 103
4.3.2 Stereo Image Warps 108
4.3.3 Wide-Angle Images and Panoramas 114
4.3.4 Content Enhancement 118
4.3.5 Finite Elements 120

5 Light Fields 125
5.1 Capture and Represent Light Fields 126
5.2 Disparity Estimation 131
5.3 Applications 138

6 3D Conversion Using Vanishing Points and Warping 147
6.1 Related Work 148
6.2 Overview 149
6.3 Constrained Image Warping 150
6.3.1 User interface 150
6.3.2 Mathematical Formulation 151
6.3.3 Optimization 154
6.4 Results 155
6.5 Limitations 157

7 Hand-Held Light Field Photography and Applications 159
7.1 Related Work 161
7.2 Spatio-Temporal 3D Light Field Resampling 163
7.2.1 Feature Trajectory Matrix 164
7.2.2 Factorization 164
7.2.3 Linear Camera Motion 167
7.2.4 Rendering of Output Views 168
7.3 Disparity Map 169
7.3.1 Score Volume Computation 169
7.3.2 Score Volume Filtering 171
7.4 Applications 176
7.4.1 Refocusing using Synthetic Apertures 176
7.4.2 Further Applications 179
7.5 Mobile Application 182
7.6 Limitations and Conclusions 184

8 Conclusions 187

Chapter 1

Introduction

There has always been a difference between real scenes and what we picture with cameras. Before the invention of photography it was impossible to record a visual instance as it is. Painters spent hours or even days and weeks to picture what they have seen or were told. It therefore seems likely that many famous kings, queens, and other royals were not as good looking as their portraits, but they were represented in the most favorable light.

With the invention of photography in the 19th century things changed. It became possible to capture a scene as it was in reality in a few moments, with light rays hitting the photography plate directly. This could all be done without an interpretation of the painter. Notwithstanding this, the attraction to modify, improve or adapt real scenes in photomontage became famous nearly as soon as photography itself. As an example we show Oscar Rejlander's *Two Ways of Life* from 1857 in Figure 1.1 and realise that digital photo and video post-production seems to follow the natural evolution of imaging.

Today with digital post-processing we can do more than analogue photo artists did. While processing of analogue photos was mainly restricted to changing lighting effects in the dark room and cutting and pasting objects with real scissors, we now have nearly unlimited

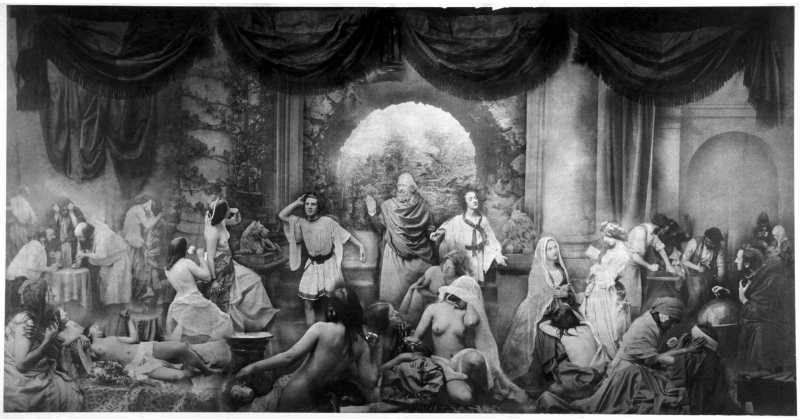


Figure 1.1: Oscar Rejlander's *Two Ways of Life* from 1857 is one of the first and most famous photomontages. It consists of about 32 negatives and was build within six weeks. [23]

possibilities to modify pictures. These include very subtle image changes such as removing red eyes or little spots in somebody's face, but also more drastic changes such as adapting a person's profile, correcting lens distortions or even computing new camera perspectives. Furthermore, the difference between photo and video cameras has shrunk and today both can already be combined into one device. The digitalisation of photography has opened up whole new fields, as for example light field photography, and has made it easier to take stereo images and videos. While in standard photography we create monocular images by capturing the light rays travelling through one lens, which is similar to seeing with just one eye, in stereo photography we capture two views form slightly different positions simultaneously. Afterwards, we can show an image to each eye giving the viewer the ability to see the scene in 3D. In light field photography we capture light rays that go through several lenses, instead of just one or two. This makes it possible to adjust the viewpoint and the focus after the fact or to retrieve depth information.

The fusion of photo and video cameras with large computational power happened in mobile phones within the last decade. Additional sensors, such as accelerometers, gyroscopes, GPS receivers and microphones provide us with information beyond the visual data that we can use to build photography and video applications. The usage of the accelerometer for video stabilization and photo deblurring [41] are two well known examples. Using GPS data to locate 3D models retrieved from images on a map [84] or using the sound of several video snippets for alignment in time [5] has become feasible.

The downside of cameras that are built into mobile phones is their size. With the aim to make mobile phones thinner, lighter, and smaller also the camera itself had to shrink. While the electrical parts of the camera could keep pace with the rest of the phone technology development, the optical parts of the cameras suffer from the miniaturization. It is best seen when mobile phone images already become noisy due to bad lighting conditions while standard cameras still are able to capture nice images. Further it is not possible to have an optical zoom with such small lenses and the depth of field is very large. Researchers and mobile phone manufacturers try to counteract these issues of small lenses with image editing algorithms. We added our piece [14] to this development, too.

Alongside with smartphones and app-stores, the connection to the internet became standard so that photo cameras and other applications now are constantly connected to the web. The opening of these new software marketplaces and the possibility to share images with friends instantly led to new business opportunities that helped to develop image editing software as well as sharing and storing solutions. Furthermore the availability of large image databases, with the internet being the biggest image database itself, brings along new possibilities. Nowadays we can use millions of pictures that were taken of tourist hot spots, for example, to obtain a 3D reconstruction of the venue [1]. We can also search the web to find the best fitting parts of images that were left with holes after cutting processes and use them to fill the gaps [29].

Not only point-and-shoot cameras and consumer photography

evolved quickly in the last decade. Stereo photography and stereo filming also has reached new heights through the upgrade of many movie theaters to the capability of displaying 3D movies. Furthermore the hardware technology has reached a level that made it possible to sell TVs and screens that can display 3D content as consumer electronics. These new advances in technology motivated researchers, including us [15], to develop specialized algorithms for stereo images and videos or multi-view content in general. Such algorithms include the production of stereo content by converting existing mono videos or photos into 3D content [25, 53, 91], optimizing the 3D perception of already captured content [49], the application of existing warping techniques to stereo content [71], but also the application of well known post-processing techniques of mono content to stereo content (such as copy and paste of image parts [59, 64]).

In this thesis we focus on computational tools for mono to stereo conversion and light field photography. Along these lines, our first research goal [15] deals with the conversion of mono into stereo images of scenes with strong geometric structure. Further we discuss light field photography as the topic of the second research objective [14], in which we focus on creating light fields with a mobile phone camera. To achieve this goal, we used techniques from video stabilization and image warping both of which shall be discussed in dedicated chapters, too.

1.1 Contributions

We contributed the following two tools to computational stereo and light field photography:

3D Conversion Using Vanishing Points and Image Warping

The interest of our research has focused on mono to stereo conversion in the field of stereo content. We have developed a user aided method [15] that can convert mono images into stereo images. The method focuses on images that mainly show man-made objects with a

strong and clear geometric structure. The user can draw some edges and faces of the depicted objects and indicate lines that have a common vanishing point. Further the user defines the desired disparity for some points. From this information our method computes two images with a new warping technique based on Caroll et al. [9], such that the two images are the left and right view of a stereo image pair and show the input image geometrically consistent in 3D.

Hand-Held 3D Light Field Photography and Applications

We have developed a mobile application [14] that exploits the computational power of modern smartphones and utilises the fusion of photo and video cameras. Instead of taking a single picture, the user takes a short video with a horizontal sweep with his hand-held device. We then convert this video, with a video stabilization method developed for this particular case, into a 3D light field. Thanks to the light field we can create a depth map and apply other image enhancement algorithms. For example the user can select the object she wants to have in focus with a finger tab on the touchscreen. Then the application computes an out-of-focus blur for the entire image. The amount of blur is adjustable by the user, too.

Video Linearization. Firstly, the application applies our video linearization, which aligns the frames of the user taken video horizontally and temporally in a way that the camera movement between two consecutive frames becomes constant. The result is a set of images of the scene that is taken from equidistant positions along a horizontal line and that we intuitively interpret as a 3D light field.

Depth Estimation. In a second step we create a depth map of the scene. In order to achieve this, we have combined and extended existing algorithms that compute the disparities for pixels form epipolar images in which each epipolar image consists of one line of each view of our 3D light field.

Large Aperture Simulation. The depth map and the 3D light field together make several applications possible. We can select two views of the light field as our left and right view and display the scene in 3D. We can also use the depth map to compute occlusions by inserting objects into the scene, or we can segment the image to create a selective grayscale image. However, the main application is the simulation of a larger aperture. As mentioned above mobile phones usually have very small camera optics leading to all-in-focus images. We have developed a method that uses the light field and a depth map to compute an artificial out-of-focus blur and can therefore narrow the depth of field. Furthermore, by refocussing using a very large synthetic aperture, we are able to remove very thin foreground objects from the image completely.

1.2 Thesis Organization

This thesis covers four chapters (Chapters 2 to 5) with background knowledge, followed by two chapters (Chapters 6 and 7) that cover our research results. The thesis is structured as follows:

Chapter 2 gives an overview of existing video stabilization methods including a short introduction into the structure-from-motion problem that, given a set of input images of a scene, computes 3D locations and orientations of cameras and a sparse feature point set that can be tracked over several of the input images.

Chapter 3 discusses the problem of converting mono images and videos into stereo content that can be perceived as "3D" by a viewer.

Chapter 4 introduces the technique of image warping in detail and includes some standard optimization methods for least squares problems. We further present many existing warping techniques that can solve a variety of different problems.

Chapter 5 explains the terms light field and epipolar images and discusses applications of light field photography.

Chapter 6 describes our own developed warping method that converts monocular images into geometrically consistent 3D stereo images.

Chapter 7 presents our video linearization algorithm, our method that we use to compute the disparities from EPIs, and how we simulate an arbitrary aperture size and other applications of light field photography.

Chapter 8 summarizes the results of our research and gives a brief outlook to possible future work.

Chapter 2

Video Stabilization

Jittery videos are a well known problem to any hobby filmer and their friends and relatives who have the pleasure to watch their movies. The problem tends to increase, when the cameras decrease. And they decreased dramatically in the last two decades. Professional movie makers use heavy hardware equipment as cameras on tracks or cranes to follow the action on a set. Another possibility is a Steadicam, which is a camera mount that can freely rotate around all 3 axis and has a counter weight to the camera. Neither are options for tourists or hobby filmer, who use small, less elaborate cameras or even mobile phones to take their shots. So, we have to find other solutions, which can be applied also after the fact and independent of the camera type which was used. The computer is always a good "other solution", especially nowadays, where nearly all videos are shot digitally. It seems to be a logical consequence, that video stabilization became an active area of research. Many different approaches were explored and new algorithms are still being developed. We discuss the most important and recent ones in this chapter.

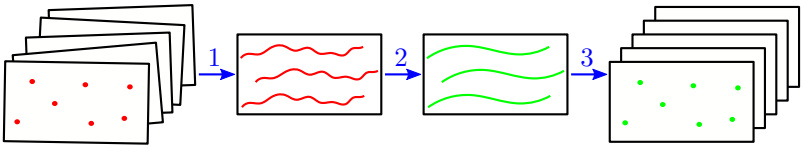


Figure 2.1: Overview of the three steps in a standard video stabilization pipeline. First, tracking features in a shaky video. Second, smoothing the jittery trajectories. Third, rendering new frames.

Before going into detail, we give an overview of the problem. First, we need to find a way to automatically detect unwanted camera motion. The human eye recognizes them suddenly and intuitively. But how can a computer do this? The most common strategy is the following. The algorithm first detects recognizable points, we call them *features*, and tracks them over several frames. This gives *feature trajectories*. There is no difficulty in finding out whether the trajectories are smooth or not. But not all trajectories have to become smooth. For example feature points tracked on a waving hand of a person bounce from left to right and back. Often the next step is to find out, which of the features belong to moving objects. These are then removed from the set of trajectories. Then we smooth these trajectories. The naive approach is to smooth them directly in 2D. Unfortunately, this does not lead to plausible results. One of the main reasons is, that 2D filtering does not respect the 3D structure of the depicted scene. Further, on each frame features appear and others disappear. As a consequence, on each frame some feature trajectories start, and some other may end. Further, some features disappear earlier than others leading to different lengths of the trajectories. Filtering these different trajectories separately yields different, inconsistent smooth trajectories. An example we show in Figure 2.2. This leads to the main question of the second step in the video stabilization process. How can we find smooth, but also geometrically consistent feature trajectories? The answer to this questions differs in every approach. We discuss one that reconstructs the 3D geometry of the feature points in Section 2.3.

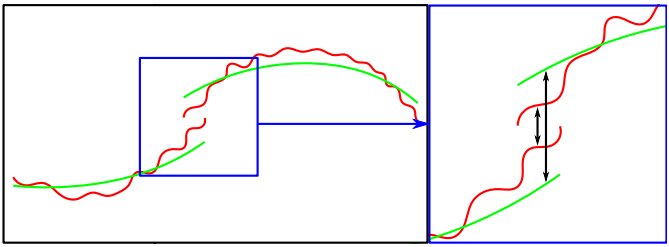


Figure 2.2: Two feature trajectories (red) are smoothed independently (green). In the close up (blue box) we see that this pulls apart the trajectories where they overlay.

For better understanding, we firstly review the 3D reconstruction of feature points in Section 2.2. A related, but simplified method bases on a subspace of the trajectory matrix. We discuss the theory in Section 2.5 and how it applies to video stabilization in Section 2.5. In Section 2.6 we summarize other methods.

After we found smooth feature trajectories, we proceed to the third step of the video stabilization pipeline. We need to render new frames which have the feature points located on the desired positions. Often, this is done with an image warp. Image warps are versatile and also used for other tasks. We spend the whole Chapter 4 on that topic. Therefore, this step of the stabilization process is not discussed in this chapter. To start from the beginning, we describe the detection and tracking of feature points.

2.1 Feature Points

To measure the movement of the camera, we need to be able to match objects in two consecutive frames. Afterwards, we can measure their movement by computing the difference in pixels. Robustly recognizing objects is a difficult task, since the object may look different in two images, because the camera angle or the lighting may be different or objects may become occluded. The workaround is to search for

distinctive points instead of whole objects. We call these points *feature points*.

Nowadays we know several methods to detect feature points. We discuss two, to give an idea of what feature points are and how they could be detected and tracked.

2.1.1 Corner Points

In his PhD thesis in 1980 [68] Moravec described a first and very simple feature detector. He states that feature points, or actually patches, which are relative easy to recognize, are non-uniform image areas. Therefore, he proposes to use pixels that are the center of an area that is most different to other close-by areas, as feature points. So, the similarity of the two pixels $\mathbf{x}_1 = (x_1, y_1)$ and $\mathbf{x}_2 = (x_2, y_2)$ is computed

$$S(\mathbf{x}_1, \mathbf{x}_2) = \sum_u \sum_v w(u,v)\big(I(\mathbf{x}_1 + (u,v)) - I(\mathbf{x}_2 + (u,v))\big)^2, \quad (2.1)$$

where u and v range from $-n$ to n for a patch of size $2n+1$, I denotes the image and w is a weighting function which can weight pixels closer to $\mathbf{x}_1$ respectively $\mathbf{x}_2$ higher than pixels further away.

Nearly ten years later, this approach was improved by Harris and Stephens, (1988, [27]), Tomasi and Kanade (1991, [87]), and Shi and Tomasi (1994, [81]). Although the derivations are slightly different, the conclusion is the same. To detect if a pixel is a corner pixel, often called a *Harris corner*, we now take into account the partial derivative instead of the pixel values directly. The partial derivative is approximated by a convolution of the image I with the row, respectively column vector $(-1, 0, 1)$. We denote the results of these convolutions I_x and I_y. So, the image gradient at the pixel $\mathbf{x} = (x, y)$ is $\nabla I(\mathbf{x}) = (I_x(\mathbf{x}), I_y(\mathbf{x}))^\mathsf{T}$. Next, we build on each pixel a symmetric 2×2 matrix by multiplying the gradient with its transposed, $\nabla\nabla^\mathsf{T} = \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$. We obtain the final *structure tensor* M by a

weighted sum of these matrices of the pixel and its neighbors,

$$M = \sum_u \sum_v w(u,v) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}, \tag{2.2}$$

where u and v again range from $-n$ to n for a patch of size $2n+1$ and I_x is the short form of $I_x(\mathbf{x}+(u,v))$. The same holds for I_y. To compute the weights, Harris and Stephens use the Gaussian function, $w(u,v) = e^{-(u^2+v^2)/2\sigma}$, [27].

Now we have a structure tensor $M(\mathbf{x})$ for each pixel but we are still left with the question, which pixels are corners. We denote by λ_1 and λ_2 the two eigenvalues of M. Harris and Stephens state that the two eigenvalues are proportional to the principal curvature and form a rotationally invariant description of M [27]. Our view is slightly different. The matrix M is the covariance matrix of the gradients of the pixels in the patch. This matrix is then used to do a principal component analysis. The result of the principal component analysis tells us in which direction we have the largest gradients, and the eigenvalues measure their importance. Both views on the problem boil down to an analysis of the eigenvalues of M. If these are small, we are on a texture-less patch. If only one eigenvalue is large, we have only gradients into one direction and the patch in question shows an edge. Finally, if we have two large eigenvalues, the patch shows two edges which are normal to each other and therefore build a corner. The decision whether a pixel is a corner or not can be made by simply testing the eigenvalues against a threshold. To reduce the computational cost, Harris and Stephens propose to not compute the eigenvalues themselves, but instead to compute a corner response number

$$R = \det(M) - \kappa \mathrm{tr}(M)^2, \tag{2.3}$$

where κ is a tunable sensitivity parameter. Note that $\det(M) = \lambda_1 \lambda_2$ and $\mathrm{tr}(M) = \lambda_1 + \lambda_2$. Then, R is positive in the corner regions, negative in the edge regions, and small in the flat regions [27]. So, it can be tested against a threshold, too.

2.1.2 Kanade–Lucas–Tomasi Feature Tracker

After detecting the features, we need to track them onto the next frames. The probably most popular algorithm that does this, is the Kanade–Lucas–Tomasi feature tracker, short KLT tracker. It bases on the work of Lucas and Kanade [63], and Tomasi and Kanade [87], but is best described by Shi and Tomasi [81]. Here we just introduce the idea of a KLT tracker and refer to the original publication for detailed explanations. Similar to the feature detection, we consider small windows around feature points in the image I and sequentially follow them onto the next image J. Neighboring pixels may move differently, hence Shi and Tomasi talk about an *affine motion field* [81], which can be described by a 2×2 deformation matrix D and a displacement vector $\mathbf{d}$. So, for each pixel in the window we have

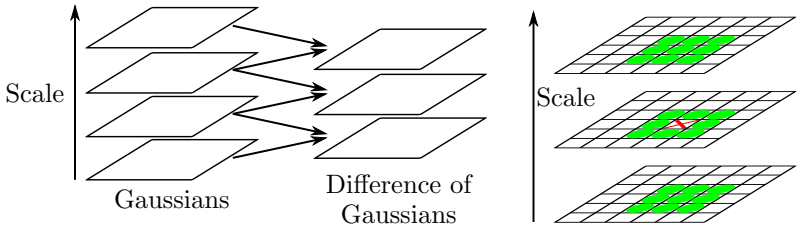
$$J(D\mathbf{x} + \mathbf{d}) = I(\mathbf{x}). \tag{2.4}$$

Note that D and $\mathbf{d}$ are the unknowns we search for. We write Equation (2.4) as the difference $J(D\mathbf{x} + \mathbf{d}) - I(\mathbf{x})$ and build a weighted sum of all differences over all pixels in the window. Then we search for D and $\mathbf{d}$ that minimize the sum. We skip the details of the further computation and point interested readers to Shi and Tomasi [81].

However, in many cases (small windows, small movement, etc.) it leads to better results when assuming D to be the identity matrix and searching just for d. Shi and Tomasi show that this leaves us with the equation system $M\mathbf{d} = \mathbf{e}$, where M is the structure tensor computed in the previous section and $\mathbf{e} = (I_x, I_y)^\mathsf{T} \sum_{\mathbf{u}} w(\mathbf{u})(I(\mathbf{x} + \mathbf{u}) - J(\mathbf{x} + \mathbf{u}))$ is the image gradient multiplied with the weighted sum of the differences of the two images inside the window.

2.1.3 SIFT

KLT feature tracking is fast and a good choice to track features on videos. Its downside is that it works only for relative small feature movements. Further, lighting condition should be the same, since we compare pixel values directly. This is not an issue in image sequences taken with a high frequency, such as movies, but it may become one in



(a) Differences of Gaussians of one Octave (b) Neighbors of a pixel

Figure 2.3: (a) shows the computation of the differences of Gaussians. The stack on the left contains the Gaussian smoothed images, the difference of two consecutive images gives the stack on the right. (b) shows the 26 neighbors (green) of one pixel (red) in the stack. [62]

other cases. In cases where the camera movement between two shots is large, KLT becomes slower when searching larger neighborhoods, looses its robustness or may even fail completely.

Nearly two decades after the Harris corners, Lowe introduced the scale-invariant feature transform SIFT [61, 62] to counteract these problems. He obtains scale-invariance by searching for feature points on different scaling levels of the image. Then he computes a rotation invariant feature descriptor and normalize it to reduce the influence of the lighting condition. These descriptors are finally used to search for matches between features on different images.

Feature Detection. Similar to others, SIFT feature tracking also heavily relies on gradients, however, not in its first step. To detect SIFT features, we take into account differences of Gaussians. The difference of two images that are filtered with different kernel size is large on edges and corners but small in uniform regions and therefore similar to a gradient image. The advantage over the standard gradient method is that filters take into account all directions and a larger neighborhood than gradients computed from the difference between neighboring pixels. This means, we smooth the image with Gaussian

functions with $\sigma_i = k_i\sqrt{2}$, where $k_i = 2^{\frac{i}{s}}$, $i = 0, ..., s$. As shown in Figure 2.3a, we compute the difference of Gaussians by taking the difference of the two images smoothed with neighboring scales i.e. σ_i and σ_{i+1}. This gives us s differences of Gaussians, which we call one *octave*. Next, we search for minima and maxima in the differences of Gaussians. First, we compare each pixel against its eight neighbors on the same scale. If it is a maximum or minimum, we also compare it against the nine neighbors on the scale below and above. Figure 2.3b shows the neighborhood of the red marked pixel with green circles.

After processing the whole octave, we can proceed to the next. To do so, we resample the image which is smoothed with $\sigma_s = 2\sqrt{2}$ by taking every second pixel in each row and column [62]. Then we use this image as the input and process the next octave in the same way.

Feature Descriptor. Once the SIFT points are found, we also create a descriptor. We use the Gaussian blurred image of the scale where we found the feature point. Lowe [62] suggests to use a patch of size 16×16, where we compute the gradient on each pixel by simply taking the difference in x and y direction to the next pixel. Further we compute the magnitude and direction of the gradient. We split the patch in 16 smaller patches of size 4×4. For each such patch, we build a histogram consisting of 8 bins, each representing a direction. These 16 histograms build a 128 entry long feature descriptor vector. To make the descriptor rotationally invariant, we first compute a main direction for each feature point, subject to its gradients, and compute all other directions relative to this main direction. To reduce the influence of the lighting, the feature descriptor is normalized.

Feature Tracking. The feature matching can be done through these descriptor vectors, assuming that we computed features and descriptors on two images. Then, we query the descriptors of the second image with each descriptor from the first image to find the closest one. The difference between two descriptors is defined as the Euclidean distance. Lowe proposes to use the best-bin-first algorithm, introduced by him and Beis [6], which is faster than the k-d-tree.

2.2 Structure from Motion

Video stabilization can be seen as moving the camera position and orientation for each frame slightly, such that the path described by the new camera locations becomes smooth. The same should be the case for the orientation of the camera. To be able to do this, we need to know the location and orientation of the camera, for each frame. The process, to extract this information, is what we call *structure from motion*. The basic idea is a very old one, and a process humans do automatically all day long: With our two eyes, which are located on different positions, we capture two slightly different images from our environment. These two images are never the same, which we can easily double check by stretching out one arm and holding up the thumb. Closing one eye gives us the view of the other eye. When we open the closed eye and close the open one at the same time, we get the impression of a "jumping thumb". The finger is on a different position relative to the background. Providing our brain with the views from both eyes at the same time, it is able to extract information about the distance to the objects. Replacing the words "eye" by "camera" and "brain" by "computer", we have a good idea of what we want to do in this section. The only difference is that humans have two eyes, which are capturing scenes at the same time. This is comparable to a stereo camera. The case we are interested in differs in that we have a sequence of images, which were taken from one and the same camera at different points in time. Therefore we also may have many more than two views from a scene. With this input, we compute the 3D location of a sparse set of features and the camera location and orientation.

Video stabilization is by far not the only application of 3D reconstruction of a scene. In the recent past, structure from motion became a basic tool for many computational photography and computer vision tasks. Understanding the process of capturing and depicting scene points with a camera helps to further understand other video stabilization algorithms, which we discuss later, and the conversion form single images and mono-videos into 3D content (Chapter 3).

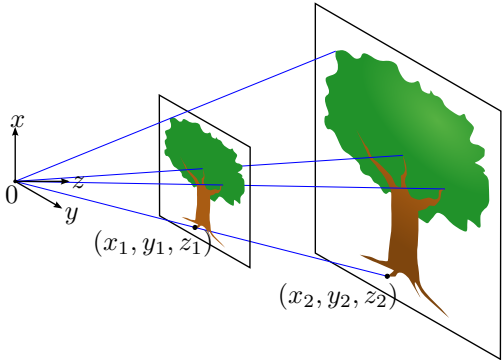


Figure 2.4: We project an image of a tree, with a projector located at the origin 0, onto two screens, one at distance z_1 and another at z_2. The image of the tree scales linear. See Figure 2.7a for image credits.

2.2.1 Projective Geometry

In this section we discuss the basics of the concept of projective geometry. It is highly linked to the camera model which we use and describe in Section 2.2.2. To give an intuitive understanding about projective geometry, we look at the actual inverse process of capturing pictures. We consider the projection of an image onto a screen. We assume the projector at the origin of the coordinate system of the 3D space such that it points along the z-axis. Then, each point of the image gets a third coordinate z, which measures the distance from the projector to the screen. See also Figure 2.4. Moving the screen closer or further away from the projector, let the image grow or decrease but does not distort it. We also could say it scales the image. Hence, it exists a factor k that scales the coordinates of an image point (x_1, y_1, z_1) on the first screen onto the second screen, $k(x_1, y_1, z_1) = (x_2, y_2, z_2)$. This behavior comes from the fact, that the projector sends one light ray for each pixel it wants to show. These rays travel straight until they hit the screen and therefore keep their relative distance to each other. In Figure 2.4 we show a few such light

rays as blue lines. The projective geometry describes the view of the projector, which knows the direction of the light rays but not how far they travel. Therefore, the projector does not distinguish between points $k(x, y, z)$, which differ only by factor k and all belong to the same ray. Likewise, this applies to projective geometry.

A line in a Euclidean 2D plane can be written as

$$ax + by + c = 0, \tag{2.5}$$

meaning that all points (x, y) satisfying the equation, form the line. This definition of the line is not unique. For any real $k \neq 0$, the vector (ka, kb, kc) defines the same line, since $kax + kby + kc = k(ax + by + c) \overset{(2.5)}{=} k0 = 0$. The two vectors (a, b, c) and $k(a, b, c)$, related by the scaling factor k, are considered to be equivalent. We denote this as $(a, b, c) \cong k(a, b, c)$. An equivalence class of vectors of such an equivalence relationship we call *homogeneous* vectors. The set of all these vectors in $\mathbb{R}^3 - (0, 0, 0)$ we call the *projective space* $\mathbb{P}^2$. The origin of $\mathbb{R}^3$ is excluded, since this vector corresponds to no line.

A 2D point (x, y) lies on line $\mathbf{l}$ if it satisfies the line Equation (2.5). With the column vector $\mathbf{l} = (a, b, c)^\mathsf{T}$ we can write the line equation as the scalar product $(x, y, 1)\mathbf{l}$. Similar as for the lines, we can also multiply the just created point vector $(x, y, 1)$ with any scalar non-zero factor and the Equation (2.5) still holds. It follows, that points in $\mathbb{R}^2$ can be represented as homogeneous vectors, similar to the lines. We therefore distinguish between the *homogeneous coordinates* (x, y, z) of a point and the *inhomogeneous* coordinates $(x/z, y/z)$.

Both, the lines and the points have 2 degrees of freedom in $\mathbb{R}^2$. It is obvious for points since we can choose the x and y coordinates independently. We learned that (a, b, c) and $k(a, b, c)$ describe the same line. Thus, we can choose $k = \frac{1}{b}$ and get $\frac{a}{b}x + y + \frac{c}{b} = 0$ as the line equation. Rearranging the terms yields $-\frac{a}{b}x + -\frac{c}{b} = y$, the standard line equation, which is usually denoted as $mx + b = y$. Note, the b is not the same number in the two equations. We see in the standard notation that we have two independent variables, the slope m and the point b where the line intersects the y axis. It follows that we also have 2 degrees of freedom.

We have seen that a point $\mathbf{x} = (x, y, 1)^\mathsf{T}$ lies on the line $\mathbf{l} = (a, b, c)^\mathsf{T}$ if and only if $\mathbf{l}^\mathsf{T}\mathbf{x} = \mathbf{x}^\mathsf{T}\mathbf{l} = 0$. Further, we compute the intersection of two lines $\mathbf{l}_1 = (a_1, b_1, c_1)^\mathsf{T}$ and $\mathbf{l_2} = (a_2, b_2, c_2)^\mathsf{T}$ by taking their cross product $\mathbf{x} = \mathbf{l}_1 \times \mathbf{l}_2$. Since the resulting vector of the cross product is perpendicular to both vectors which form the cross product, we have $\mathbf{l}_1^\mathsf{T}\mathbf{x} = 0$ and $\mathbf{l}_2^\mathsf{T}\mathbf{x} = 0$. It follows that $\mathbf{x}$ lies on both lines and therefore on the intersection point. With the same argument, we find the line $\mathbf{l}$ going through the two points $\mathbf{x}_1$ and $\mathbf{x}_2$ by taking their cross product. This example shows nicely how the role of points and lines may be interchanged in statements about the properties of points and lines. Indeed, there is a so called *duality principle*, which states:

Duality principle To any theorem of 2-dimensional projective geometry there corresponds a dual theorem, which may be derived by interchanging the roles of points and lines in the original theorem. [28]

All points with homogeneous coordinates $\mathbf{x} = (x, y, z)$ with $z \neq 0$ correspond to finite points in $\mathbb{R}^2$. Now, we extend this set by adding points with $z = 0$, the so called *ideal points* or *points at infinity*. This extended set of points in homogeneous coordinates spans the whole *projective space* $\mathbb{P}^2$. We can write the set of points at infinity as $(x, y, 0)^\mathsf{T}$, where a specific point is defined by the ratio $x : y$. They all lie on one line, denoted as $\mathbf{l}_\infty = (0, 0, 1)^\mathsf{T}$ and called the *line at infinity*. It is easy to verify that $(0, 0, 1)(x, y, 0)^\mathsf{T} = 0$ for all x and y values.

The intersection of the line $\mathbf{l} = (a, b, c)^\mathsf{T}$ with the line at infinity is $\mathbf{l} \times \mathbf{l}_\infty = (b, -a, 0)^\mathsf{T}$, which is an ideal point, too. Further, the parallel line $\mathbf{l}' = (a, b, c')^\mathsf{T}$ with the same coordinates a and b as $\mathbf{l}$, has the same intersection point $(b, -a, 0)^\mathsf{T}$ as $\mathbf{l}$. And indeed $\mathbf{l} \times \mathbf{l}' = (c' - c)(b, -a, 0)^\mathsf{T}$. Neglecting the factor $(c' - c)$, we obtain the same point at infinity as the intersection point of $\mathbf{l}$ and $\mathbf{l}'$. It is therefore consistent with the idea that parallel lines in Euclidean space intersect at infinity.

We call a linear mapping from a homogeneous vector to another one a *homography*. Synonyms are also *projectivity*, *projective transformation*, and *projective collineation*. We can write the homography H

as a non-singular 3×3 matrix. Then, the transformation becomes

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}, \tag{2.6}$$

or $\mathbf{u} = H\mathbf{x}$, respectively.

Obviously the homographies form a group: The inverse of a homography is a homography too, and also the composition of two homographies is one. Since we have a mapping of a projective space to a projective space, kH is equivalent to H for any scaling factor k. Assume it is not, then $\mathbf{x}' = H\mathbf{x}$ is different from $\mathbf{x}'' = (kH)\mathbf{x}$ for any $k \neq 0$. Thanks to the linearity of the homography we have $\mathbf{x}'' = (kH)\mathbf{x} = H(k\mathbf{x}) \cong H\mathbf{x} = \mathbf{x}'$, a contradiction to the assumption. It follows that the homographies form an equivalence class, too. Consequently, we call these matrices *homogeneous*. A homography has, as a consequence, only 8 degrees of freedom. We can always choose one entry of its matrix. We usually set $h_{33} = 1$. For any homography matrix H, we reach this by dividing all its entries by h_{33}.

The main property of the homography is that it maps lines onto lines. Assume the line $\mathbf{l}$ is defined as the line through the two points $\mathbf{x}_1$ and $\mathbf{x}_2$. Then, for $\forall \alpha \in \mathbb{R}$ we have $\mathbf{l}^\mathsf{T}\mathbf{x}_\alpha = 0$ with $\mathbf{x}_\alpha = \alpha\mathbf{x}_1 + (1-\alpha)\mathbf{x}_2$, since $\mathbf{x}_\alpha$ is a point on the line through $\mathbf{x}_1$ and $\mathbf{x}_2$. Using the linearity of H we write $H\mathbf{x}_\alpha = \alpha H\mathbf{x}_1 + (1-\alpha)H\mathbf{x}_2$, which is a line through $H\mathbf{x}_1$ and $H\mathbf{x}_2$. On the other hand we have $\mathbf{l}^\mathsf{T}\mathbf{x}_\alpha = \mathbf{l}^\mathsf{T}H^{-1}H\mathbf{x}_\alpha = 0$ and it follows that $\mathbf{l}^\mathsf{T}H^{-1}$ is the line we were looking for.

We started this section with a photo or video projector and stated that moving the screen closer or further away to the projector scales the image but keeps angles. We implied that the screen plane stays orthogonal to the projecting direction while moving it. If this assumption is breached, the projected image becomes distorted. We show this schematic in Figure 2.5. On the left, we see the projection of an image onto two screens. I is the image projected onto a screen orthogonal to the projection direction z. $\hat{I}$ is the new image projected onto a tilted screen and does not form a square. In the sketch left in Figure 2.5, we see the reason for this. The rays from the center of

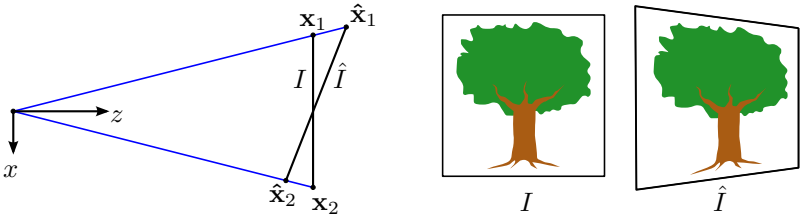


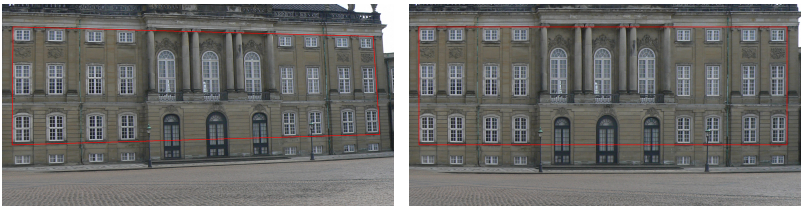
Figure 2.5: Here we show a sketch of a projection onto a plane that is orthogonal to the projecting direction and a plane which is tilted. I and $\hat{I}$ are the resulting images.

projection to the points $\hat{\mathbf{x}}_1$ and $\hat{\mathbf{x}}_2$ have different lengths. Therefore the two scaling factors k_1 and k_2, which scale $\mathbf{x}_1$ and $\mathbf{x}_2$ onto $\hat{\mathbf{x}}_1$ and $\hat{\mathbf{x}}_2$, differ, and the image increases on the left and decreases the right side. We can describe the relation between two images projected onto different planes by a homography.

An image of a wall, which we assume to be flat as plane, behaves the same as a projection onto screen. It acquires the same perspective distortions when the camera that takes the picture does not point perfectly orthogonal to the wall. We can correct these distortions through a homography and show a practical example, of how to remove the perspective distortion in Figure 2.6a through a homography. To do so, we map each pixel $\mathbf{x}$ from the distorted image with a homography H onto the non-distorted point $\mathbf{x}' = H\mathbf{x}$. We turn pixel coordinates (x, y) into a homogeneous vector $\mathbf{x} = (x, y, 1)$ by adding a 1 as the homogeneous coordinate. When we expand $\mathbf{x}' = H\mathbf{x}$ and denote it in pixel coordinates instead of homogeneous vectors, we get

$$x' = \frac{h_{11}x + h_{12}y + h_{13}}{h_{31}x + h_{32}y + h_{33}}, \qquad y' = \frac{h_{21}x + h_{22}y + h_{23}}{h_{31}x + h_{32}y + h_{33}}. \tag{2.7}$$

Here we divide by the third homogeneous coordinate, to obtain the representation $(x', y', 1)$ of $\mathbf{x}'$. All four values x, y, x', and y' are then local coordinates in the $\mathbb{R}^2$ coordinate system of the images, i.e. pixel coordinates. Rearranging the terms yields for one point



(a) Input image (b) After applying the homography

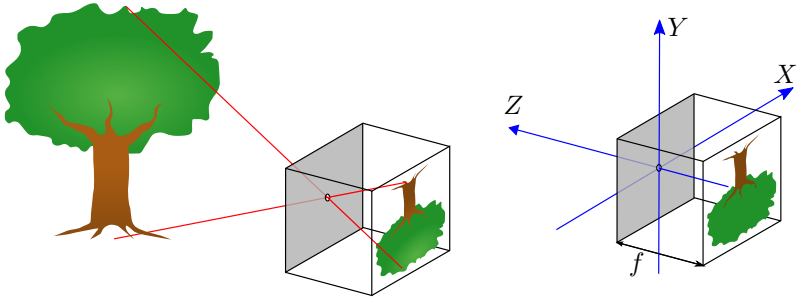
Figure 2.6: We determine the perspective distortion in the input image (a) by drawing lines that are parallel in reality. We use the four intersection points to compute a homography, which corrects the distortion and apply it to each pixel. (b) shows the result.

correspondence two linear equations,

$$x'(h_{31}x + h_{32}y + h_{33}) = h_{11}x + h_{12}y + h_{13} \tag{2.8}$$
$$y'(h_{31}x + h_{32}y + h_{33}) = h_{21}x + h_{22}y + h_{23}. \tag{2.9}$$

Since H has 8 degrees of freedom, we can set $h_{33} = 1$. With 4 known corresponding points, for example the corners of the red quadrilateral in Figure 2.6, we then build a system of 8 linear equations with 8 unknowns. Solving this, determines the matrix H of the homography we were looking for. The only constraint we have by choosing the four points is that they have to be in general position. This means, no more than two of them can lie on one line.

Through the application of homography we simulated a camera movement from one spot (Figure 2.6a), to the location from where the camera points exactly into the direction perpendicular to the depicted wall. This we did, without knowing about the camera, its location or pointing direction. The limitation of this application is, that it only works in the special case where a photograph depicts nothing but planes. In order to being able to handle more complex photos, we study the mathematical camera model in the next subsection.



(a) Pinhole camera (source: Wikipedia[1]) (b) Coordinate system

Figure 2.7: (a) shows a pinhole camera that captures a tree. (b) visualizes the 3D coordinate system with origin at the camera's pinhole.

2.2.2 Camera Model

Each camera captures a 3D scene and bans it onto a 2D image. Consequently, the camera model is a mapping function that maps 3D points onto a 2D plane. The purpose of this subsection it to describe how this mapping works. To begin with, we take a look at the simplest camera we can think of, the pinhole camera. It is basically an all black box, which has a small hole in the middle of one face. Through this hole, light comes into the box and projects the image onto the inner side of the opposite face (Figure 2.7a). All that needs to be done now, is to save the projected image. This is done by a photographic film in analog cameras and by a CCD sensor in nowadays digital cameras.

The mathematical description of a pinhole camera is easy to understand, bearing in mind the section about projective geometry. We choose the world coordinate system such that its origin is right at the hole of the pinhole camera. The x and the y-axis span the plane which contains the pinhole cameras surface. The z-axis is perpendicular to it so that it points outwards of the camera. See also Figure 2.7b. Each

[1]This image is taken from the Wikipedia article about the pinhole camera `https://en.wikipedia.org/wiki/Pinhole_camera`. All other figures showing a tree, are modifications of it, done by the author.

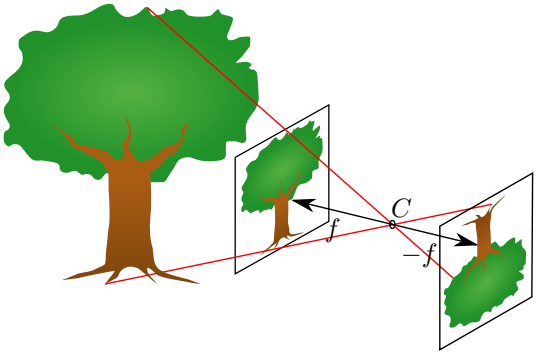


Figure 2.8: The real world tree (left) is projected onto two image planes with z-coordinate f and $-f$. C denotes the center of the projection, to photographers known as focal point.

point of the camera's field of view that is not occluded by another object has 3D coordinates (X, Y, Z). Note that Z is positive for each such point. For a better distinction with image coordinates, we denote 3D coordinates with capital letters. We now interpret the 3D coordinate as a homogeneous vector. This makes intuitively sense, since a homogeneous vector describes a line through the origin. In our case, we interpret this line as the light ray that travels from the seen 3D point (X, Y, Z) through the cameras pinhole onto the opposite side of the box. The point where the light ray hits the back side of the pinhole camera is the spot that the 3D point is projected onto. We assume the depth of the box to be f. In the 3D coordinate system this is the plane $(X, Y, -f)$. Then the projection point that we have just described has the 3D coordinates $-f(X/Z, Y/Z, 1)$. The minus sign indicates that the image of a pinhole camera is inverted. Further, f is the *focal length*. The coordinates in the 2D image coordinate system are $(-f\frac{X}{Z}, -f\frac{Y}{Z})$. We show a sketch in Figure 2.8.

The principle of the pinhole camera is key to understanding the projection of the 3D world onto a 2D plane and is used as the starting point to extend camera models to cameras with complex lens systems.

First we omit the minus sign by just pretending that the plane of projection is in front of the pinhole of the camera. Physically this is wrong, of course, but mathematically it is equivalent. Further, the formulation above assumes that the coordinate system is rectified and has the same scaling in the x and y-direction on the image plane. This means the pixels are squares. This assumption is usually true, with some practical exceptions. Considering this, we have $(x, y) = (\alpha_x \frac{X}{Z}, \alpha_y \frac{Y}{Z})$, with $\alpha_x = f m_x$ and $\alpha_y = f m_y$, where m_x and m_y are the size of a pixel in x and y-direction, respectively. To have only positive values in the image coordinate system we move the image plane by (p_x, p_y) pixels, where p_x and p_y are the half of the image width and height. If the CCD sensor is not placed carefully enough, its center is not incident with the z-axis of the real world 3D coordinate system. We can correct this by adjusting (p_x, p_y). We combine all these camera parameters in the 3×3 *camera calibration matrix*

$$K = \begin{bmatrix} \alpha_x & s & p_x \\ 0 & \alpha_y & p_y \\ 0 & 0 & 1 \end{bmatrix}. \tag{2.10}$$

All parameters which appear in the calibration matrix are called *internal* camera parameters. The *skew parameter* s can have non-zero values in rare cases of projective cameras. However, for our purposes we assume it to be zero. The parameter is just given for completeness. Usually, we assume to have square pixels and therefore f on the diagonal instead of α_x and α_y. To get the image coordinates we write the 3D location of a point as a column vector and multiply it with the calibration matrix, followed by the homogeneous division,

$$\begin{bmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} fX + p_x Z \\ fY + p_y Z \\ Z \end{bmatrix} \mapsto \begin{bmatrix} f\frac{X}{Z} + p_x \\ f\frac{Y}{Z} + p_y \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix}. \tag{2.11}$$

So far, we chose the origin of the 3D coordinate system such that it coincides with the camera's pinhole. We call such a coordinate system the *camera coordinate system*. Whenever we have the freedom to choose the coordinate system, we prefer this basic one. However, if

a so called *world coordinate system* is given, then our task is to map the 3D points from the world coordinate system into the camera coordinate system. This mapping consists of a rotation and a translation, given that both coordinate systems are Euclidean and have the same orientation. We usually assume to have this. Then, the three rotation angles and the three dimensional translation vector are the six *external* camera parameters. To add the rotation and the translation to the camera matrix, we write the rotation as a 3×3 rotation matrix R and the translation as a column vector t. This leads to

$$P = K\,[R|t] \tag{2.12}$$

where $[R|t]$ denotes the horizontal concatenation of R and t, and K is the camera calibration matrix form Equation (2.10). It follows, that P is a 3×4 matrix. To be able to apply P to a point in the world coordinate system, we have to add 1 as a fourth coordinate to it. It becomes a homogeneous vector, too. Hence, the camera matrix P is a projection form $\mathbb{P}^3$ onto $\mathbb{P}^2$. In the notation of Equation (2.12), the translation vector t is applied after the rotation. This has to be taken into account when we assemble the camera matrix. Further, we also need to take care if the rotation is given from the view of the camera or the world coordinate system.

2.2.3 Two-View Geometry

We saw in the previous subsection that we can lose one dimension by projecting a 3D point onto a 2D image. This also means, we lose information. More specifically, we lose the information of "how far away" a point is. Each 3D point interpreted as a homogeneous vector is actually a line in the 3D coordinate system, which goes through the origin. If now we want to invert this process, that is reconstruct the "lost" third coordinate of a point, we need additional information. This additional information is given through firstly, a second image of the same scene, taken from a different spot and secondly, the distance between the two camera locations and their focal length. With this information we can use *triangulation*. Triangulation is a technique,

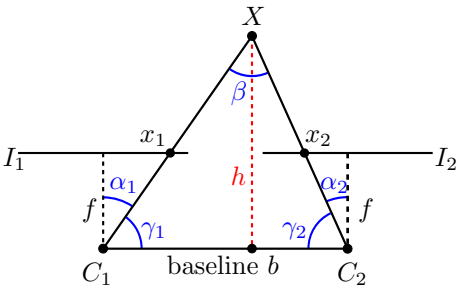


Figure 2.9: Two parallel cameras C_1 and C_2 with focal length f capture the point X. x_1 and x_2 are the locations of X on the image planes I_1 and I_2.

which was already developed over 2000 years ago and was used since the 16th century for land surveying and map-making until the rise of the GPS system in the 1980s.

To explain the triangulation technique we use the sketch in Figure 2.9. There we look from top onto two cameras, pointing in the same direction. The points C_1 and C_2 mark the center of projection for the two cameras. X is a world coordinate 3D point, depicted in the images I_1 and I_2 as x_1 and x_2. From the focal length f, which is actually the distance between the center and the projection plane, and the image coordinates x_1, x_2 of the depicted point X, we can compute the two angles $\alpha_1 = \arctan(x_1/f)$ and $\alpha_2 = \arctan(x_2/f)$. The inner angles of the shown triangle are then $\gamma_i = \frac{\pi}{2} - \alpha_i$, $i \in \{1, 2\}$ and $\beta = \pi - \gamma_1 - \gamma_2$, the angle opposite the baseline b. Then, we can compute the two unknown triangle edge lengths with the trigonometric formula $\frac{c_i}{\gamma_i} = \frac{b}{\beta}$, where c_i denotes the length of the edge $\overline{C_i X}$ and therefore the distance of X to camera i. The z-coordinate of X corresponds to the height h of the triangle, given that the world coordinate system coincides with one of the two camera coordinate systems. h can be computed using trigonometric formulas, too. We have $b = \frac{h}{\tan\gamma_1} + \frac{h}{\tan\gamma_2}$. Using the trigonometric identities

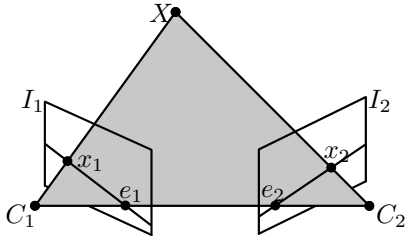
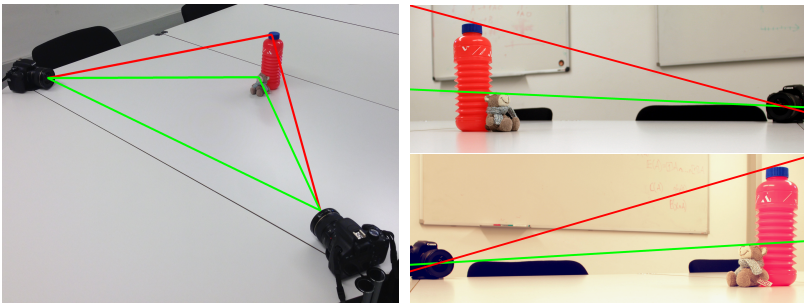


Figure 2.10: The two camera centers C_1 and C_2 and the point X span the epipolar plane. The intersection of the epipolar plane and the imaging plane I_1 yields the epipolar line through x_1 and e_1. e_1 denotes the epipole.

$\tan(\alpha) = \sin(\alpha)/\cos\alpha$ and $\sin(\alpha+\beta) = \sin(\alpha)\cos(\beta) + \cos(\alpha)\sin(\beta)$, we can rearrange the terms such that we get $h = b\frac{\sin(\gamma_1)\sin(\gamma_2)}{\sin(\gamma_1+\gamma_2)}$. It follows that we can compute the unknown edges and the triangle's height h. In other words we can compute all three coordinates of X in the world coordinate system.

In this example, we actually have a very special setting. The cameras point exactly in the same direction and their location differs only in the x-coordinate. This is a rare situation in practice. Still, the point X and the two camera centers C_1 and C_2 build a triangle. We exclude here degenerated cases where all three points lie on a line or two points coincide. This means the principle from above always works, it just may become more complicate to get the required numbers. Further, the two camera centers C_1 and C_2 and the point X span a plane in the 3D space, the *epipolar plane*. The intersection of this plane with the imaging plane is a line, which we call the *epipolar line*. We show a sketch in Figure 2.10. Note, each world point creates a different triangle and therefore a different epipolar plane, which leads to a different epipolar line. The intersection point of all epipolar lines in one image is called the *epipole*. It also is the intersection point of the camera baseline with the image plane, hence the projection of the camera center of the other camera. In Figure 2.11a we show an



(a) Epipolar plane (b) Epipolar lines

Figure 2.11: (a) shows two cameras depicting a bear and a bottle. The cameras and the bear's nose span a triangle and define an epipolar plane. (b) shows the two camera views and two epipolar lines.

example, in which two cameras point inwards such that they see each other. The green triangle with the camera centers and the bear's nose defines the epipolar plane. We show in Figure 2.11b the views of the two cameras from Figure 2.11a with the epipolar lines from the bear's nose (green) and the bottle's top (red).

Another description of the epipolar line is the following: The epipolar line of the real world point X in image I_1 is the projection of the light ray from X into camera C_2. Figure 2.10 shows that the epipolar line given by X in image I_1 is the projection of the triangle edge $\overline{XC_2}$. In the special setting, discussed at the beginning of this subsection and shown in Figure 2.9, all epipolar lines are horizontal. Consequently, their intersection point is at infinity. This makes sense because we find the intersection of camera baseline with the imaging planes, in Figure 2.9 drawn as I_1 and I_2, at infinity, too. This is because the imaging planes are parallel to the baseline.

2.2.4 Fundamental Matrix

Assume two cameras picture the same scene from different viewpoints, that is with different centers of projection. Just given the two images, we know nothing about the cameras and their positions. But there has to be something which connects the two images taken, right? At least, they show the same scene. The intuition is right! This "something" we can formulate mathematically as the *fundamental matrix* F. It is the unique 3×3 rank 2 homogeneous matrix which satisfies

$$\mathbf{x}_2^\mathsf{T} F \mathbf{x}_1 = 0 \tag{2.13}$$

for all corresponding point pairs $\mathbf{x}_1$, $\mathbf{x}_2$, denoted as homogeneous vectors [28].

To understand the fundamental matrix better, we assume for a moment that we know the two camera matrices P_1 and P_2, which project real world points onto the image planes $\mathbf{x}_1 = P_1\mathbf{X}$, $\mathbf{x}_2 = P_2\mathbf{X}$ respectively. $\mathbf{X}$ denotes the homogeneous vector (X, Y, Z, W) of a real world point. Remember, the camera matrix $P = K\,[R|t]$ is a 4×3 matrix and has therefore a null vector $\mathbf{C}$, $P\mathbf{C} = \mathbf{0}$. In conclusion, we only can determine its inverse up to this null vector. If we do this, we obtain for our camera matrix P_1

$$\mathbf{X}(\lambda) = P_1^+ \mathbf{x}_1 + \lambda \mathbf{C}_1 \tag{2.14}$$

where P_1^+ denotes the pseudo inverse of P_1, this is a 4×3 matrix with $P_1 P_1^+ = I$, λ is any real number and $\mathbf{C}_1$ denotes the null vector. Moreover, $\mathbf{C}_1$ is the location of the camera center in world coordinates. Since the camera calibration matrix K has full rank, the null vector of the matrix P_1 is also the null vector of $[R_1|\mathbf{t}_1]$. Therefore, $\mathbf{C}_1$ has to satisfy the equation $R_1(c_1, c_2, c_3)^\mathsf{T} = -c_4\mathbf{t}_1$, with $\mathbf{C}_1 = (c_1, c_2, c_3, c_4)^\mathsf{T}$. Assuming the homogeneous coordinate $c_4 = 1$, we have $(c_1, c_2, c_3)^\mathsf{T} = R_1^\mathsf{T}\mathbf{t}_1$, which is the location of the camera center.

Equation (2.14) shows, why we can not derive the depth of a point on an image from the image itself, even we have full knowledge about the camera. λ is unknown. Hence, we know two points on the ray, on which $\mathbf{X}$ has to lie: $P_1^+\mathbf{x}_1$ and the center $\mathbf{C}_1$ of the camera P_1.

The projection of these two points onto the second image defines the epipolar line $\mathbf{l}_2 = P_2\mathbf{C}_1 \times P_2P_1^+\mathbf{x}_1$, on which the projection of $\mathbf{X}$, $\mathbf{x}_2 = P_2\mathbf{X}$ has to lie. This is in formulas $\mathbf{x}_2^\mathsf{T}\mathbf{l}_2 = 0$. We learned already that the projection of the center of the first camera into the second image is the epipole, hence $\mathbf{e}_2 = P_2\mathbf{C}_1$. Further we can write the cross product as a matrix multiplication[2]. This allows us to transform the term $P_2\mathbf{C}_1 \times P_2P_1^+$ into a matrix

$$F = [\mathbf{e}_2]_\times P_2P_1^+, \tag{2.15}$$

which satisfies Equation (2.13) in the definition of the fundamental matrix. Further $[\mathbf{e}_2]_\times$ has rank 2, P_1 and P_2^+ have both rank 3. It follows that F has rank 2 and all requirements of a fundemantal matrix are shown. Another property of the fundamental matrix is that it has only 7 degrees of freedom. A 3×3 matrix has has 9 degrees of freedom by default. But F has rank 2, so we can only choose 8 entries arbitrarily. The ninth is then given by the fact, that the third line (or row) has to be linearly dependent on the first two. Additionally, F is homogeneous, which removes another degree of freedom.

To get more insight on the meaning of Equation (2.15), we consider the following example. We assume two cameras such that the world coordinate system coincides with the first camera's coordinate system,

$$P_1 = K_1[I|\mathbf{0}] \qquad\qquad P_2 = K_2[R|\mathbf{t}]. \tag{2.16}$$

Then we have

$$P_1^+ = \begin{bmatrix} K_1^{-1} \\ \mathbf{0}^\mathsf{T} \end{bmatrix} \qquad\qquad \mathbf{C}_1 = \begin{bmatrix} \mathbf{0} \\ 1 \end{bmatrix} \tag{2.17}$$

and inserting into Equation (2.15) yields

$$\begin{aligned} F &= [P_2\mathbf{C}_1]_\times P_2P_1^+ \\ &= [K_2\mathbf{t}]_\times K_2RK_1^{-1} = K_2^{-\mathsf{T}}RK_1^\mathsf{T}[K_1R^\mathsf{T}\mathbf{t}]_\times \end{aligned} \tag{2.18}$$

[2] $\mathbf{a} \times \mathbf{b} = [\mathbf{a}]_\times\mathbf{b} = \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$

where we skipped several steps in the last equation. For more details, we refer to Hartley and Zisserman [28]. We can identify the two epipoles

$$\mathbf{e}_1 = P_1 \begin{bmatrix} -R^\mathsf{T}\mathbf{t} \\ 1 \end{bmatrix} = -K_1 R^\mathsf{T}\mathbf{t} \quad \text{and} \quad \mathbf{e}_2 = P_2 \begin{bmatrix} \mathbf{0} \\ 1 \end{bmatrix} = K_2\mathbf{t} \tag{2.19}$$

in Equation (2.18) directly.

So far, $F_{12} = F$ is the projective mapping from a point in the projective space of the first camera onto the one of the second camera. We can compute similarly the fundamental matrix F_{21}, which projects into the other direction:

$$\begin{aligned} F_{21} \overset{(2.15)}{=} [P_1\mathbf{C}_2]_\times P_1 P_2^+ &= [\mathbf{e}_1]_\times K_1 R^\mathsf{T} K_2^{-1} \\ &= \left(K_2^{-T} R K_1^\mathsf{T} [-\mathbf{e}_1]_\times\right)^\mathsf{T} = F_{12}^\mathsf{T}, \end{aligned} \tag{2.20}$$

where we use $P_2^+ = \begin{bmatrix} R^\mathsf{T} K_2^{-1} \\ \mathbf{0}^\mathsf{T} \end{bmatrix}$ in the first step. Then we simply transpose the whole term and use that $[\mathbf{a}]_\times^\mathsf{T} = [-\mathbf{a}]_\times$. Finally we replace $\mathbf{e}_1$ by the result of Equation (2.19) and compare to the last term of Equation (2.18). What we derived in Equation (2.20) is not a speciality of this example. Moreover, we found another property of the fundamental matrix: If F is the projective transformation from I_1 onto I_2, then its transposed F^T is the projective transformation from I_2 onto I_1.

Before we close the subsection about the fundamental matrix and its properties, we want to highlight one additional and important remark. We assumed so far that the center of the cameras do *not* coincide. If the two centers have the same location, the translation vector $\mathbf{t}$ becomes zero, and $F = 0$ in Equation (2.18), too. The same is true for the epipoles, as we can see in Equation (2.19). Even if the two camera centers differ, there are many special cases, which we do not discuss here. We give only one more example. This is the probably simplest setting, where the two cameras point into the same direction and have only a translation in the x-direction. We used this example already for the description of the triangulation (Figure 2.9).

Then,

$$F = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix}.$$

2.2.5 Finding Camera Matrices

In the previous subsection, we computed the fundamental matrix from the two cameras. In this subsection, we show that the opposite is possible, too. For the moment we assume that a fundamental matrix F is given and the cameras P_1 and P_2 are the unknowns. The camera matrices are dependent on the world coordinate system while the fundamental matrix contains only information about the relative position between the cameras, but no information about the real world coordinate system. Assume we have two corresponding points $\mathbf{x}_1$, $\mathbf{x}_2$ and two cameras P_1 and P_2 which project the 3D world point $\mathbf{X}$ onto them. We now apply the projective transformation H, a 4×4 matrix, to the 3D world coordinates. Then $\mathbf{X}$ becomes $\tilde{\mathbf{X}} = H\mathbf{X}$ and the cameras become $\tilde{P}_1 = P_1 H^{-1}$ and $\tilde{P}_2 = P_2 H^{-1}$. The fundamental matrix remains unchanged

$$\begin{aligned} \mathbf{x}_2^\mathsf{T} F \mathbf{x}_1 &= (P_2\mathbf{X})^\mathsf{T} F (P_1\mathbf{X}) \\ &= (P_2 H^{-1} H\mathbf{X})^\mathsf{T} F (P_1 H^{-1} H\mathbf{X}) = (\tilde{P}_2\tilde{\mathbf{X}})^\mathsf{T} F (\tilde{P}_1\tilde{\mathbf{X}}). \end{aligned} \tag{2.21}$$

In conclusion, we obtain two camera pairs which both have the same fundamental matrix.

This ambiguity is not a downside since it gives us the freedom to choose the world coordinate system on our own. We usually choose the *canonical form* in order to set the first camera to the form $[I|\mathbf{0}]$. We are then left with the question what we can deduce for the second camera $P_2 = [A, \mathbf{a}]$.

Actually, we are looking just for A, since $\mathbf{a} = \mathbf{e}_2$ is the epipole. Remember, $\mathbf{e}_2 = P_2\mathbf{C}_1$, with $\mathbf{C}_1 = (0, 0, 0, 1)^\mathsf{T}$ the center of camera P_1 in homogeneous coordinates. The epipole $\mathbf{e}_2$ has to satisfy $\mathbf{e}_2^\mathsf{T} F \mathbf{x}_1$ for all possible $\mathbf{x}_1$ since it lies on all epipolar lines $\mathbf{l}_2 = F\mathbf{x}_1$. Therefore

the epipole $\mathbf{e}_2$ is the left null space of F, that is $\mathbf{e}_2^\mathsf{T} F = 0$ and can be computed directly from F.

If F is a fundamental matrix and P_1 and P_2 the corresponding two cameras, then the matrix $P_2^\mathsf{T} F P_1$ has to be skew-symmetric. This is because $0 = \mathbf{x}_2^\mathsf{T} F \mathbf{x}_1 = \mathbf{X}^\mathsf{T} P_2^\mathsf{T} F P_1 \mathbf{X}$ is true for any vector $\mathbf{X}$ if and only if $P_2^\mathsf{T} F P_1$ is skew-symmetric. We can satisfy this constraint by setting $A = SF$, with S a random skew-symmetrix matrix. This leads to a camera matrix $P_2 = [SF|\mathbf{e}_2]$ and we obtain

$$P_2^\mathsf{T} F P_1 = [SF|\mathbf{e}_2]^\mathsf{T} F [I|\mathbf{0}] = \begin{bmatrix} F^\mathsf{T} S^\mathsf{T} F & \mathbf{0} \\ \mathbf{e}_2^\mathsf{T} F & 0 \end{bmatrix}. \tag{2.22}$$

The right hand side is a skew-symmetric matrix, since $\mathbf{e}_2^\mathsf{T} F = 0$. Luong and Vieville [65] suggested using $S = [\mathbf{e}_2]_\times$. This leads to the two cameras

$$P_1 = [I|\mathbf{0}] \qquad \text{and} \qquad P_2 = [[\mathbf{e}_2]_\times F|\mathbf{e}_2] \tag{2.23}$$

which are computable directly from F.

In a next step we need to show that the two found matrices P_1 and P_2 are cameras indeed. While this is self-evident for P_1, we have to show that P_2 has rank 3. F has to have rank 2, otherwise it is not a fundamental matrix and $\mathbf{e}_2^\mathsf{T} F = 0$. It follows that two of the three column vectors of F are linearly independent and therefore span a plane. $\mathbf{e}_2$ is perpendicular to this plane, since it is perpendicular to all of the column vectors $\mathbf{f}_i$ of F, because $\mathbf{e}_2^\mathsf{T} \mathbf{f}_i = 0$ for $i \in \{1, 2, 3\}$. Hence, taking the cross product of $\mathbf{e}_2$ and $\mathbf{f}_i$ gives three vectors, all of which are perpendicular to $\mathbf{e}_2$ and span the same plane as $\mathbf{f}_1, \mathbf{f}_2, \mathbf{f}_3$. Therefore $[\mathbf{e}_2]_\times F$, is the same as taking the cross product of $\mathbf{e}_2$ and all the column vectors of F and has rank 2. Then, adding $\mathbf{e}_2$ as an additional column $[\mathbf{e}_2]_\times F$ leads to the desired rank 3 matrix.

2.2.6 Projective Reconstruction

The next and also last step in the reconstruction pipeline for two views, is to find a 3D point for each point pair $\mathbf{x}_1$, $\mathbf{x}_2$ with given cameras P_1 and P_2.

We have the two equations

$$\mathbf{x}_1 = P_1\mathbf{X} \qquad \text{and} \qquad \mathbf{x}_2 = P_2\mathbf{X}, \tag{2.24}$$

where the unknown 3D point is writen as the homogeneous 4-vector $\mathbf{X}$. The problem that arises at this stage is that the equation signs mean *in the same equivalence class*, since we can multiply the left hand side of the equations with any factor and they remain valid. To overcome this problem, we require the vectors of the left hand side and the right hand side to be parallel. Then, the length of the vectors, and therefore the scale factor, can be neglected. The cross product of two parallel vectors has to be zero, so the equations become $\mathbf{x}_1 \times P_1\mathbf{X} = 0$ and $\mathbf{x}_2 \times P_2\mathbf{X} = 0$. Writing this out yields

$$\begin{aligned} x_1(\mathbf{p}_1^{3\mathsf{T}}\mathbf{X}) - (\mathbf{p}_1^{1\mathsf{T}}\mathbf{X}) &= 0 & \qquad x_2(\mathbf{p}_2^{3\mathsf{T}}\mathbf{X}) - (\mathbf{p}_2^{1\mathsf{T}}\mathbf{X}) &= 0 \\ y_1(\mathbf{p}_1^{3\mathsf{T}}\mathbf{X}) - (\mathbf{p}_1^{2\mathsf{T}}\mathbf{X}) &= 0 & \qquad y_2(\mathbf{p}_2^{3\mathsf{T}}\mathbf{X}) - (\mathbf{p}_2^{2\mathsf{T}}\mathbf{X}) &= 0 \\ x_1(\mathbf{p}_1^{2\mathsf{T}}\mathbf{X}) - y_1(\mathbf{p}_1^{1\mathsf{T}}\mathbf{X}) &= 0 & \qquad x_2(\mathbf{p}_2^{2\mathsf{T}}\mathbf{X}) - y_2(\mathbf{p}_2^{1\mathsf{T}}\mathbf{X}) &= 0 \end{aligned}$$

where $\mathbf{p}_j^{i\mathsf{T}}$ denotes the ith row vector of the jth camera matrix. These equations are linear and we have a total of six for one point pair. By taking four of these equations, with two from each camera, we can build a linear equation system of the form $A\mathbf{X} = \mathbf{0}$ with

$$A = \begin{bmatrix} x_1\mathbf{p}_1^{3\mathsf{T}} - \mathbf{p}_1^{1\mathsf{T}} \\ y_1\mathbf{p}_1^{3\mathsf{T}} - \mathbf{p}_1^{2\mathsf{T}} \\ x_2\mathbf{p}_2^{3\mathsf{T}} - \mathbf{p}_2^{1\mathsf{T}} \\ y_2\mathbf{p}_2^{3\mathsf{T}} - \mathbf{p}_2^{2\mathsf{T}} \end{bmatrix}. \tag{2.25}$$

We now have four equations to solve to obtain four homogeneous coordinates. Hence, we cannot solve $\mathbf{X}$ directly, becaue it is defined only up to a scale, which also can be 0. There are two possibilities to solve this problem. One is to require the homogeneous coordinate of $\mathbf{X}$ to be 1 by setting $\mathbf{X}^{\mathsf{T}} = (X, Y, Z, 1)$. Another possibility is to require $\|\mathbf{X}\| = 1$. This is the equivalent problem to find the minimum of the ratio $\|A\mathbf{X}\| \,/\, \|\mathbf{X}\|$. The solution is the eigenvector of $A^{\mathsf{T}}A$ with the smallest eigenvalue [28]. We can find it through a singular value

decomposition $A = USV^\top$. Then, the column vector of V, which corresponds to the smallest singular value, represents this eigenvector.

Without a discussion in depth, we want to mention here, that the two possible workarounds behave differently whenever there is noise in the image points. It is only possible to set the homogeneous coordinate to 1 where we can be sure that we are not solving for a point at infinity. Remember, points on the infinite line have 0 as third coordinate. This marks a clear restriction when we are processing outdoor scenes, which include a visible horizon.

2.2.7 Finding the Fundamental Matrix

We have already discussed many properties of the fundamental matrix and the way we obtain two camera matrices from it. Indeed, the fundamental matrix opens up many secrets about the two cameras which depict a common scene. In this subsection we want to explore, how we can find this useful matrix, given point correspondences on two images. We basically have Equation (2.13) and the knowledge about its 7 degrees of freedom and the rank 2. First, we exploit Equation (2.13), which is

$$\begin{bmatrix} x_2 & y_2 & 1 \end{bmatrix} \begin{bmatrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ f_{31} & f_{32} & f_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} = 0 \tag{2.26}$$

when we write down all its elements. Just rearranging terms leads to

$$\begin{array}{rcrcrcl} f_{11}x_1x_2 & + & f_{12}y_1x_2 & + & f_{13}x_2 & + & \\ f_{21}x_1y_2 & + & f_{22}y_1y_2 & + & f_{23}y_2 & + & \\ f_{31}x_1 & + & f_{32}y_1 & + & f_{33} & = & 0, \end{array} \tag{2.27}$$

which is a linear equation for the 9 unknowns f_{ij}. Since F is homogeneous, we can reduce the unknowns by fixing one of them, for example $f_{33} = 1$. For the other unknowns, we can set up a system with 8 equations, given we have 8 pairs of corresponding points, and solve it. What sounds like a standard mathematical problem, does not work well in practice in this case. Feature points on an image

are never noise free and we usually obtain a matrix with full rank instead of rank 2. To correct this we use SVD in order to decompose the found $F = USV^\mathsf{T}$ into two orthogonal and one diagonal matrix S. The matrix S contains the singular values of F. We set the smallest of three singular values to zero and call the matrix S'. Then we use the manipulated diagonal matrix to compute a new $F' = US'V^\mathsf{T}$, which is the closest matrix under Frobenius norm to F that has rank 2.

The numerical stability of this method can be improved much by normalizing the image coordinates at the beginning. We apply a translation and scale all image points, such that their center is at the origin of the coordinate system, and the squared mean distance of the points to the origin is 2. These transformations can be done by applying 3×3 matrices T_1 and T_2 to the original image coordinates. To receive the fundamental matrix F for the original image coordinates, we have to multiply the found and corrected fundamental matrix F' with the inverse transformations: $F = T_2^{-1}F'T_1^{-1}$.

We can do even better when we combine the two steps of finding the fundamental matrix and computing a projective reconstruction. This hybrid algorithm is called the *Gold Standard* method [28]. It searches for the fundamental matrix $\hat{F}$ that minimizes

$$\sum_i d(\mathbf{x}_1^i, \hat{\mathbf{x}}_1^i)^2 + d(\mathbf{x}_2^i, \hat{\mathbf{x}}_2^i)^2 \tag{2.28}$$

subject to $\hat{\mathbf{x}}_1^i \hat{F} \hat{\mathbf{x}}_2^i = 0$, where $\hat{\mathbf{x}}_j^i$ denotes the pojection of $\mathbf{X}^i$ with P_j. Intuitively, it searches for the fundamental matrix, which is perfect for image locations $\hat{\mathbf{x}}_j^i$ that are as close as they can be to the given locations $\mathbf{x}_j^i$. Hence, the error computed in Equation (2.28) is minimized by the algorithm and measured directly in (squared) pixel units.

We start the procedure with an initial guess of the fundamental matrix F as described at the beginning of this section. Next, we obtain two camera matrices $P_1 = [I|\mathbf{0}]$ and $P_2 = [[\mathbf{e}_2]_\times F|\mathbf{e}_2]$, with $\mathbf{e}_2$ the left null vector of F. From these two camera matrices we can build the linear equation system given in Equation (2.25), to find a homogeneous representative of a 3D point $\mathbf{X}^i$ for each point correspondence $\mathbf{x}_1^i$, $\mathbf{x}_2^i$. The true point location on the two images we find by back projecting

the found 3D point. The so found locations $\hat{\mathbf{x}}_1^i = P_1\mathbf{X}^i$, $\hat{\mathbf{x}}_2^i = P_2\mathbf{X}^i$ we use to compute the error according to Equation (2.28). To minimize the error we build a non-linear equation system, in which we keep P_1 fixed and the $\mathbf{x}_j^i$s are known. P_2 and $\mathbf{X}^i$ are the unknowns. For n corresponding point pairs, this leads to a total of $12 + 3n$ variables. The non-linear system can be solved with one of the methods described in Section 4.2. Usually the Levenberg–Marquardt algorithm is the choice.

Once, the optimal solution, that is the one which minimizes (2.28) is found, we have actually a projective reconstruction. In the case we still need to know the best fitting fundamental matrix, we build it from the found camera $P_2 = [M|\mathbf{t}]$ as $\hat{F} = [\mathbf{t}]_\times M$.

2.2.8 n-View Geometry

The last step in every structure from motion pipeline is to refine the found cameras and 3D point locations. We call this important part the *bundle adjustement*. We project each found 3D point $\mathbf{X}^i$ with the found camera P_j and measure the error to the corresponding feature point $\mathbf{x}_j^i$ on the input image. Mathematically,

$$\min_{P_j, \mathbf{X}^i} \sum_{i,j} d(\mathbf{x}_j^i, P_j\mathbf{X}^i) \tag{2.29}$$

a non-linear minimization problem, which can be solved for example with the Levenberg–Marquardt algorithm (Section 4.2). The idea is actually the same as in the Gold Standard method for finding fundamental matrices. But it is now relaxed to several views respectively cameras.

In the simplest case of bundle adjustment, we search for camera matrices P_j and 3D point locations $\mathbf{X}^i$, like we did it in the previous section. If we keep the first camera fixed as $[I|\mathbf{0}]$, we have $12(m - 1) + 3n$ variables for m views and n different points. Further each point location in each view gives two equations. In case we have a decomposition of the cameras into internal parameters, rotation and translation, we can constrain the system to keep this. We can

refine the parameters directly. While this is straightforward for the entries of the calibration matrix K and the translation vector $\mathbf{t}$, we can denote the rotation with 3 Euler angles and use the standard conversion to build the rotation matrix R, each time we compute the projection error. If we know, that the internal parameters are the same for all views, for example in a video with fixed focus, we can use the same calibration parameters for all cameras to reduce the number of variables we search for.

Classical optimization algorithms get usually trapped in local minima. The best way to overcome this problem is to make educated guesses by finding values that are given to the algorithm and already come close to the solution. This is the main problem of bundle adjustment. We have already discussed how we find the cameras and a good estimate of the registered 3D points from two views. The naive approach to add the next view is to process the second and third view in the same way as the first two. Then we end up with two different reference coordinate systems, one which has the camera P_1 as its origin, one with P_2 in its origin. We can avoid this problem with the assumption that the point set, that we want to reconstruct, is affine. This means that the coordinate systems given by two different reference frames can be mapped to each other with an affine 4×4 matrix H, which has $(0, 0, 0, 1)$ as its last row. 12 unknown entries in H result. We can compute these from the two different representations of P_2, since $P_2 = \hat{P}_2 H$ gives 12 linear equations. Computing all initial guesses relative to the first camera is another option. However, it has the downside that camera P_1 needs to have overlap with all other cameras to be able to compute fundamental matrices between P_1 and P_j, $j > 1$. A third possibility is to use the found 3D locations to compute the next camera. Assume, we have already found j cameras and are now processing camera $j + 1$. Then we also have a 3D reconstruction for all tracked feature points in the j processed views. If some of them are also visible in the $j + 1$th view, we can use them to set up linear systems to compute camera P_{j+1}. Once camera P_{j+1} is found, we can use it to do the reconstruction of the points which are only visible in the cameras j and $j + 1$.

The conclusion of this paragraph is that there are many ways to obtain an initial guess for cameras and the reconstruction of 3D point locations. Until now there is no "gold standard". The decision which assumptions are most likely and which downsides can be neglected depends on the application.

Summary

To close this section, we give an overview of the steps, which need to be done in a structure from motion pipeline.

1. Extract and match feature points (Section 2.1).

2. Select two frames and estimate the fundamental matrix between them (Section 2.2.7).

3. Compute the two cameras (Section 2.2.5) and the 3D points from the fundamental matrix (Section 2.2.6).

4. Add successively frames, compute fundamental matrices and estimate the camera and 3D point locations (Section 2.2.8).

5. Refine the found cameras and 3D point locations with bundle adjustment (Section 2.2.8).

2.3 3D Video Stabilization

Now that we have learned how the reconstruction of 3D points, camera locations and orientations works, we describe how this applies to video stabilization. Buehler et al. [7] were the first to develop a video stabilization method, which uses structure from motion, in 2001. In their work Buehler et al. do a projective reconstruction and then upgrade it to a quasi-affine system. They restrict the application of their method to linear camera motions without rotations. The feature trajectories in the image space are then pushed to straight lines. New camera matrices, which project the found 3D points as

close as possible to these lines, are then searched. This finally leads to a relatively simple non-linear least squares problem.

In 2009, Smith et al. [82] used a camera array to capture videos. This makes the 3D reconstruction simpler. For each captured point in time, different camera poses are available and relative camera positions are known. This makes it possible to compute the 3D locations of the found and matched features separately for each time step. Then, similar to Buehler et al. [7], they compute new camera locations relative to the input camera, such that the features describe smooth trajectories. We come back to this method in Section 2.7.

In 2009, Liu et al. [55] followed the probably most intuitive way of video stabilization. They run the whole structure from motion pipeline to compute the 3D locations of a sparse feature set as well as the 3D camera location and orientation for each frame. For this step the Voodoo Tracker[3] is used. Once we know the camera path, we can either smooth it or replace it by another, desired one. While smoothing the actual path can be done by low-pass filtering the 3D coordinates of the camera poses directly, correcting the camera orientation is quite a bit harder. The trick Liu and colleagues use in their work was developed by Lee and Shin [50]. They propose to denote the camera orientations as quaternions. We can see this step as the transformation of the space of rotations into a linear vector space, where we can use standard filtering techniques. Afterwards, we transform the filtered quaternions back into rotation matrices. We can then use the rotation matrices directly to build the new camera matrix. Compared to this process, it sounds much simpler to directly replace the original camera path by a desired one. On the other side, the disadvantage is that the desired path may be too far away from the original one. This may become a problem, when it comes to the rendering step. Once the new camera locations and orientations are found, the 3D features can be projected. That gives us the pixel locations of these features on the frames taken with the stabilized camera. Liu et al. use the knowledge of the feature locations before and after the stabilization of the camera to guide their content-preserving warp. With this, each frame can

[3] http://www.viscoda.com

be newly rendered and appears as if it would have been taken by a camera moving along a smooth path. We discuss the warping in detail in Chapter 4.

One of the most recent works which uses structure from motion for video stabilization was presented at SIGGRAPH 2014 by Kopf et al. [46]. They target first-person videos. First person videos are movies that show the scene as it is seen through the eyes of the camera operator. Usually those kind of movies are much less smooth than those that were filmed with a hand-held camera. Additionally, they are often long and contain moments without a real plot, for example when the filmer has to wait on a traffic light while capturing his bike ride with a helmet camera. Therefore Kopf et al. were not satisfied with only stabilizing the videos, which is why they also speed up their movies by creating so called hyper-lapse videos. Similar to Liu et al. [55] this is done by reconstructing the 3D positions of features and cameras, including the camera orientation. Then each input frame is overlaid with a mesh grid and a depth value for each grid vertex is computed, based on the knowledge of the sparse feature set. In that way they construct a dense depth map, which they then use as a geometry proxy in the rendering step at the end. The new camera locations are computed through an energy minimization problem, which takes into account that the new camera path has to be smooth, fit a given length and avoids showing parts of the scene that where never captured on the input video.

2.4 Subspace Constraint

Doing a 3D reconstruction of a whole scene to stabilize a video or aligning an image sequence to a specific camera path may sound like cracking a nut with a sledgehammer. Indeed, there are methods to exploit other properties of feature points on images, instead of performing a full 3D reconstruction. One such method, which is closely related to structure from motion, is the subspace constraint firstly described by Michal Irani [37] in 2002. We first explain what subspace constraint means and afterwards show why this constraint

exists. In the following Section 2.5, we introduce a video stabilization algorithm that builds on the subspace constraint.

Provided that we already have tracked and matched feature points over a video or image sequence, we can fill their coordinates into a matrix

$$M = \begin{bmatrix} x_1^1 & x_2^1 & \dots & x_F^1 \\ y_1^1 & y_2^1 & \dots & y_F^1 \\ x_1^2 & x_2^2 & \dots & x_F^2 \\ \vdots & \vdots & & \vdots \\ y_1^N & y_2^N & \dots & y_F^N \end{bmatrix}, \tag{2.30}$$

such that the columns of the matrix contain all feature locations of one frame and the rows of the matrix contain either the x or the y-coordinates of a whole feature trajectory. For this section, we only consider features that may be tracked over the whole sequence. Intuitevely one thinks that such a matrix has some special properties, since it represents the projections of a 3D point set. Albeit this, the entries are not fully independent of each other and are restricted to a subspace of the matrix. This is where the term *subspace constraint* stems from. Already in 2002 Michal Irani [37] proved that such a subspace, and therefore also the matrix, has rank 9 at the most. We recapitulate this proof next.

What Irani actually showed was that the matrix of the point differences has rank 9. The differences to a reference frame are given by $u_j^i = x_j^i - x^i$ and $v_j^i = y_j^i - y^i$, where (x^i, y^i) is the location of the ith feature in the reference frame and (x_j^i, y_j^i) is the location of the same feature on the jth frame. Then, the matrix δM consists of all u_j^i and v_j^i instead of x_j^i and y_j^i.

We first consider one single point with 3D location X^i, Y^i, Z^i. For better readability we omit the superscript i. The projection of the point onto the reference frame is $x = f\frac{X}{Z}$, $y = f\frac{Y}{Z}$, where f denotes the focal length. Similarly, the projection onto the jth frame is $x_j = f_j\frac{X+\delta X}{Z+\delta Z}$, $y_j = f_j\frac{Y+\delta Y}{Z+\delta Z}$, where f_j denotes the focal length on the jth frame and δX, δY, δZ the movement of the 3D point relative to the camera, according to its rotation and translation. Then, we

obtain for the point differences

$$\begin{bmatrix} u_j \\ v_j \end{bmatrix} = \begin{bmatrix} x_j - x \\ y_j - y \end{bmatrix} = \begin{bmatrix} f_j \frac{X+\delta X}{Z+\delta Z} - f\frac{X}{Z} \\ f_j \frac{Y+\delta Y}{Z+\delta Z} - f\frac{Y}{Z} \end{bmatrix}. \tag{2.31}$$

We extend the fractions on the right hand side to the common denominator $(Z + \delta Z)Z$, which we write as Z^2 since $\delta Z \ll Z$. This leads to

$$\begin{bmatrix} u_j \\ v_j \end{bmatrix} = \frac{1}{Z^2} \begin{bmatrix} f_j(X + \delta X)Z - fX(Z + \delta Z) \\ f_j(Y + \delta Y)Z - fY(Z + \delta Z) \end{bmatrix} \tag{2.32}$$

$$= \frac{1}{Z^2} \begin{bmatrix} (f_j - f)XZ + f_j\delta XZ - fX\delta Z \\ (f_j - f)YZ + f_j\delta YZ - fY\delta Z \end{bmatrix}. \tag{2.33}$$

Next, we replace δX, δY, δZ by the derivatives $\dot{X}$, $\dot{Y}$, $\dot{Z}$. These derivatives are approximated by Longuet-Higgins and Prazdny [60] with the condition that $t_Z \ll Z$ and the approximations are

$$\begin{aligned} \dot{X} &= \Omega_Y Z - \Omega_Z Y + t_X, \\ \dot{Y} &= \Omega_Z X - \Omega_X Z + t_Y, \\ \dot{Z} &= \Omega_X Y - \Omega_Y X + t_Z. \end{aligned} \tag{2.34}$$

With this replacement we obtain for Equation (2.33)

$$\begin{aligned} \begin{bmatrix} u_j \\ v_j \end{bmatrix} = &\begin{bmatrix} (f_j - f)\frac{X}{Z} + \frac{f_j}{Z}(\Omega_Y Z - \Omega_Z Y + t_X) \\ (f_j - f)\frac{Y}{Z} + \frac{f_j}{Z}(\Omega_Z X - \Omega_X Z + t_Y) \end{bmatrix} \\ &+ \begin{bmatrix} -f\frac{X}{Z^2}(\Omega_X Y - \Omega_Y X + t_Z) \\ -f\frac{Y}{Z^2}(\Omega_X Y - \Omega_Y X + t_Z) \end{bmatrix}. \end{aligned} \tag{2.35}$$

Then we just rearrange the terms

$$\begin{aligned} \begin{bmatrix} u_j \\ v_j \end{bmatrix} = &\begin{bmatrix} -f\Omega_X \frac{XY}{Z^2} + f\Omega_Y \frac{X^2}{Z^2} + f_j\Omega_Y - f_j\Omega_Z \frac{Y}{Z} \\ -f\Omega_X \frac{Y^2}{Z^2} + f\Omega_Y \frac{XY}{Z^2} - f_j\Omega_X + f_j\Omega_Z \frac{X}{Z} \end{bmatrix} \\ &+ \begin{bmatrix} \frac{f_j t_X}{Z} - f t_Z \frac{X}{Z^2} + (f_j - f)\frac{X}{Z} \\ \frac{f_j t_Y}{Z} - f t_Z \frac{Y}{Z^2} + (f_j - f)\frac{Y}{Z} \end{bmatrix}. \end{aligned} \tag{2.36}$$

Finally, we can write the right hand side as two row vectors

$$U = \begin{bmatrix} -\frac{XY}{Z^2} & \frac{X^2}{Z^2} & 0 & 1 & -\frac{Y}{Z} & \frac{1}{Z} & 0 & -\frac{X}{Z^2} & \frac{X}{Z} \end{bmatrix} \tag{2.37}$$

and

$$V = \begin{bmatrix} -\frac{Y^2}{Z^2} & \frac{XY}{Z^2} & -1 & 0 & \frac{X}{Z} & 0 & \frac{1}{Z} & -\frac{Y}{Z^2} & \frac{Y}{Z} \end{bmatrix} \tag{2.38}$$

and a column vector

$$\begin{aligned} P_j = \Big[& f\Omega_X \quad f\Omega_Y \quad f_j\Omega_X \quad f_j\Omega_Y \quad f_j\Omega_Z \quad \dots \\ & f_j t_X \quad f_j t_Y \quad f t_Z \quad (f_j - f) \Big]^T . \end{aligned} \tag{2.39}$$

Now, the row vectors only contain point-dependent entries and the column vector contains only camera dependent entries. Finally, Equation (2.31) turns into

$$\begin{bmatrix} u_j \\ v_j \end{bmatrix} = \begin{bmatrix} U \\ V \end{bmatrix} P_j. \tag{2.40}$$

It is straightforward to extend this to many features and several frames. For each feature point we create U^i and V^i row vectors, and similarly, we build a corresponding camera vector P_j for each frame. To do so, we simply add a superscript i to all entries in U and V. In P_j we already distinguish between the focal lengths f and f_j. They remain the same. However, all ts and Ωs are frame-dependent and therefore we add a subscript j. Then, the matrix δM, which contains the differences to the reference frame of all features in all frames, can be written as

$$\delta M = \begin{bmatrix} u_1^1 & u_2^1 & \dots & u_F^1 \\ v_1^1 & v_2^1 & \dots & v_F^1 \\ u_1^2 & u_2^2 & \dots & u_F^2 \\ \vdots & \vdots & & \vdots \\ v_1^N & v_2^N & \dots & v_F^N \end{bmatrix} = \begin{bmatrix} U^1 \\ V^1 \\ U^2 \\ \vdots \\ V^N \end{bmatrix} \begin{bmatrix} P_1 & P_2 & \dots & P_F \end{bmatrix}. \tag{2.41}$$

The stack of U^i, V^i vectors always has width 9, independently of the number of features we track. This is similarly for the matrix formed

by the P_j column vectors. The height is always 9, but the width depends on the number of frames. Equation (2.41) is a decomposition of the matrix δM into two matrices, which have both rank 9 at the most. From the rule $\text{rank}(AB) \leq \min(\text{rank}(A), \text{rank}(B))$ it follows that δM has rank 9 at the most, too.

In the last step, we want to show that the same rank constraint applies to the matrix M that contains the actual feature locations (x_j^i, y_j^i). From the definition of u_j^i, v_j^i, we have

$$\begin{aligned} x_j^i &= u_j^i + x^i = U^i P_j + x^i \\ y_j^i &= v_j^i + y^i = V^i P_j + y^i, \end{aligned} \tag{2.42}$$

where $x^i = f\frac{X^i}{Z^i}$, $y = f\frac{Y^i}{Z^i}$. A closer look at U^i and V^i displays that the terms $\frac{X^i}{Z^i}$ or $\frac{Y^i}{Z^i}$ are already included as the last element of the vectors. So, we can increase the last element of P_j by f to get

$$\begin{aligned} \hat{P}_j = \Big[f\Omega_X \quad f\Omega_Y \quad f_j\Omega_X \quad f_j\Omega_Y \quad f_j\Omega_Z \quad \ldots \\ f_j t_X \quad f_j t_Y \quad f t_Z \quad f_j \Big]^T. \end{aligned} \tag{2.43}$$

Then, we can write $x_j^i = U^i \hat{P}_j$ and $y_j^i = V^i \hat{P}_j$, and the same argumentation for δM becomes valid for the matrix M.

2.5 Subspace Video Stabilization

We saw in Section 2.2, where we discussed the structure from motion pipeline briefly, that it is quite expensive to compute the correct 3D location for all feature points and the camera itself. Further, in some situations structure from motion fails. This may occur, when the camera movement is mainly rotational, or the scene is totally flat. In such a flat scene, as for example a painting or portrait, that is filmed with a hend-held camera, even small camera jitters are highly recognizable, but structure from motion algorithms in general fail.

On the other hand, filtering the trajectories of the features directly in 2D is not possible as we have discussed in the beginning of this

chapter. Even though, it is worth to store all feature locations in a matrix M, such that each row contains either the x or the y-coordinates of one feature over all frames as in Equation (2.30). Then the columns contain all feature point locations of one frame. We saw in the previous section that this matrix M has rank 9 at the most. Liu et al. [56] exploit this fact to stabilize videos. One way to understand the idea is that we look for a basis of this 9 dimensional subspace. Then, we filter the basis and therefore all feature trajectories in one go and consistantly with respect to their 3D location. In the remainder of this section we summarize how Liu and colleagues do this.

2.5.1 Filtering with Subspace Constraint

To begin with, let us assume all features can be tracked over all frames. In practice this is not the case, but a good starting point for our explanation. We store the feature locations in a trajectory matrix M as in Equation (2.30). From the previous section we know that we can approximate this matrix M, with a matrix which has rank 9. Therefore we can factorize this approximation matrix into two matrices, such that we have

$$M \approx CE \tag{2.44}$$

where C has the same height as M but width 9 and E has height 9 but the same width as M. Remember, the height of M is two times the number of feature trajectories, the width is given by the number of frames. Liu et al. [56] call the row vectors of E *eigen-trajectories*. They can be seen as the basis of the 9 dimensional subspace. On the other hand, C, the coefficient matrix, describes the 2D feature location as a linear combination of the eigen-trajectories.

Once we are at the point where we factorized M successfully, we can smooth the eigen-trajectories directly. This can be done with a simple 1D Guassian or even a box filter applied to E row wise. In addition, more complex filtering operations are possible according to Liu et al. [56]. The filtered eigen-trajectories then build a new matrix $\hat{E}$. To get the smoothed 2D trajectories, we multiply the smooth eigen-trajectories with the coefficient matrix C.

In practice our assumption that all features can be tracked over all frames is wishful thinking. Luckily, there is a simple solution for this problem. When filling the feature locations into M, we leave the entries in M empty wherever a feature trajectory ends or has not yet started. The empty places are filled with zeros. To make sure that we do not add features through the filter step, we compute a binary matrix W, consisting of ones, where M has a non-zero entry and zeros elsewhere. We then multiply the product of the coefficient matrix C and the smoothed eigen-trajectories $\hat{E}$ element-wise with the mask W to get the smoothed feature trajectory matrix,

$$\hat{M} \approx W \odot C\hat{E}. \tag{2.45}$$

The symbol $\odot$ represents the element-wise multiplication.

Given the smoothed output feature locations in matrix $\hat{M}$ and its input locations, the stabilized frames can be rendered with the content-preserving warp by Liu et al. [55]. We postpone the discussion of the warp to Section 4.3.1 and address the question of how we can find the factorization.

2.5.2 Moving Factorization

It is unlikely that we have enough trajectories which can be tracked over the whole video to do the stabilization. This leads to empty entries in M that we filled with zeros. But to be able to factorize M, we need to have a full matrix that has no empty or zero entries. The workaround is to avoid doing the factorization for all frames at once. Liu et al. [56] divide the matrix M in overlapping sub-matrices, such that each of them consists of only known non-zero entries and can therefore be factorized. The overlap of each sub-matrix with the previous one guarantees that the factorization is consistent to the one which is already computed.

To initialize the process we reorder the trajectories in M such that all trajectories, which start on the first frame and can be tracked for at least k frames, are at the top. k is the window size, which has to be at least 9, but should be as large as possible. Liu et al. [56] suggest

to set $k = 50$. Assume we have m trajectories which can be tracked over the first k frames or longer. We define M^0 as the sub-matrix containing the first $2m$ rows and the first k columns of M. So, M^0 stores the feature locations of the m trajectories, which can be tracked over the first k frames. M^0 is a full matrix and can be factorized as

$$M^0_{2m\times k} = C_{2m\times 9} E_{9\times k}. \tag{2.46}$$

The subscript indicates the size of the matrices. To yield this initial factorization, we compute the singular value decomposition (SVD) of M^0. In the obtained diagonal matrix we set all eigenvalues except the 9 largest to zero and multiply the square root of the modified diagonal matrix with the other two matrices obtained by the SVD. Note, to be able to keep the 9 largest eigenvalues we need to have $2m > 9$. If this is not the case, we can choose a smaller k.

After we have found the factorization for M^0 we move the factorization window forward by δ frames. Similar to k, δ can be arbitrarily chosen and is suggested to be $\delta = 5$ [56]. The next sub-matrix, M^1, is built in the same fashion as M^0 and consists of all the feature trajectory locations on the frames $\delta+1, ..., \delta+k$, which can be tracked over the whole range. M^0 and M^1 overlap where they contain the locations of feature trajectories that could be tracked from frame 1 until frame $\delta+k$ and are therefore in M^0 and M^1. After ordering the trajectories accordingly, we can divide $M^0 = \left[\begin{smallmatrix} A^{00} & A^{01} \\ A^{10} & A^{11} \end{smallmatrix}\right]$ and $M^1 = \left[\begin{smallmatrix} A^{11} & A^{12} \\ A^{21} & A^{22} \end{smallmatrix}\right]$ into sub-matrices, such that A^{11} contains the overlapping entries. We visualize this matrix factorization in Figure 2.12. The factorization for $A^{11} = C^1 E^1$ is already computed at this stage and is kept fixed. C^1 and E^1 are the corresponding sub-matrices from C and E. We can now use C^1 and E^1 to solve for C^2 and E^2 in the quadratic equation

$$\begin{bmatrix} A^{11} & A^{12} \\ A^{21} & A^{22} \end{bmatrix} = \begin{bmatrix} C^1 \\ C^2 \end{bmatrix} \begin{bmatrix} E^1 & E^2 \end{bmatrix} \tag{2.47}$$

in a least-squares manner. This can be done with standard methods like Gauss–Newton algorithm or the Levenberg–Marquardt method, but it is time consuming. A much faster method is to approximate

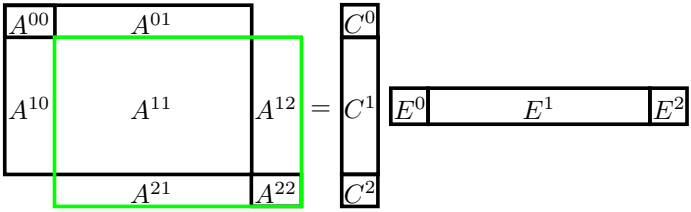


Figure 2.12: The factorization of the matrix M^1 (green): C^0, C^1, E^0 and E^1 are kept fix. C^1 and E^1 are used to compute C^2 and E^2.

the solution through two linear equations. Instead of solving for C^2 and E^2 at the same time, we solve one after each other. First we compute

$$C^2 = A^{21} {E^1}^T \left(E^1 {E^1}^T\right)^{-1} \tag{2.48}$$

and afterwards

$$E^2 = \left(\begin{bmatrix} C^1 \\ C^2 \end{bmatrix}^T \begin{bmatrix} C^1 \\ C^2 \end{bmatrix} \right)^{-1} \begin{bmatrix} C^1 \\ C^2 \end{bmatrix}^T \begin{bmatrix} A^{12} \\ A^{22} \end{bmatrix}. \tag{2.49}$$

These linear approximations are nearly as accurate as solving the quadratic equation. For our purposes, it suits the need.

Once M^1 is factorized, meaning C^2 and E^2 are computed, we move the factorization window forward by again δ frames. We build a new sub-matrix M^2 and repeat the steps described in the previous paragraph with M^1 in place of M^0. This is done until the end of M is reached.

As a last step, we check if we have found coefficients for all trajectories. The algorithm does not process a trajectory if it is shorter than k frames, or starts on a frame between $n\delta$ and $(n+1)\delta$ but ends before frame $(n+1)\delta + k$. For such a trajectory, the coefficients can be computed easily by solving a linear system, once the whole E is known. Then, smooth output feature trajectories can be computed with the filtering step as discussed in the previous Subsection 2.5.1.

2.6 Other Methods

So far we have focused our discussion on video stabilization methods that use a 3D reconstruction of feature points, camera location and orientation directly [7, 82, 55, 46]. In addition, we have discussed a method that skips the explicit reconstruction step, but still decomposes the feature trajectories such that the point locations and the camera locations can be described separately [56]. In this section we discuss two recent methods which do not use any reconstruction of the scene or camera at all. However, at least in the first work that we discuss, some of the basics about the relationship between camera positions (also compare Section 2.2) are still being used.

2.6.1 Video Stabilization using Epipolar Geometry

The title of this section is also the title of the work presented by Goldstein and Fattal at SIGGRAPH Asia 2012 [21]. In their actual stabilization process, no knowledge about the scene structure or the camera itself is being used. Not even information about the epipolar geometry, as the title claims. Instead, they filter the trajectories of feature points directly! The epipolar geometry is only used before and afterwards.

To be able to filter the trajectories in the image space, Goldstein and Fattal need to ensure that they have very long trajectories, which are also well distributed over the whole frame. This is reached by extending the existing trajectories in a technique which we call *epipolar point transfer* [16, 28]. Epipolar point transfer works as follows: We have three different views from a scene and we have found and matched a set of feature points successfully. Then, we can compute the fundamental matrices between all three views, F_{12}, F_{23} and F_{13}. In a next step we choose a feature point p^i, which was detected and matched on the first two views, but not yet on the third one. Thanks to the fundamental matrices, we know where to search for $\mathbf{p}_3^i$, the feature point location in the third image. It has to lie on the epipolar

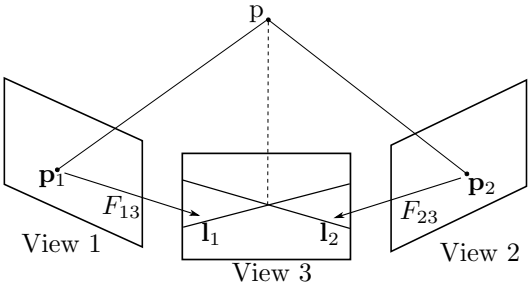


Figure 2.13: Point p could be tracked on views 1 and 2 as $\mathbf{p}_1$ and $\mathbf{p}_2$. With the fundamental matrices F_{13} and F_{23}, which relate the first two views with the third, we can compute the epipolar lines $\mathbf{l}_1$ and $\mathbf{l}_2$ on view 3. The intersection point marks the location of $\mathbf{p}_3$.

line $\mathbf{l}_1 = F_{13}\mathbf{p}_1^i$. Remember, $\mathbf{l}_1$ is the projection of the ray going through the 3D point location of p^i and the center of the first camera. At the same time, $\mathbf{p}_3^i$ also has to be on the epipolar line $\mathbf{l}_2 = F_{23}\mathbf{p}_2^i$, the projection of the ray through the 3D point and the center of the second camera. It follows, that $\mathbf{p}_3^i$ is the intersection point of the two epipolar lines $\mathbf{l}_1$ and $\mathbf{l}_2$ and we successfully found the projection of p^i onto the third view.

Goldstein and Fattal use this method to extend the found feature trajectories. For a greater robustness, they use five views or fundamental matrices and epipolar lines in order to compute the point on the sixth image instead of only two. The idea remains the same. For a good distribution of the trajectories over the whole frame, they partition the frame regularly with a rectangular grid. Then they ensure that at least one trajectory goes through each grid cell.

Since the trajectories are filtered in the 2D image space directly and independently of each other, it may happen that the filtered trajectories do not represent a valid geometric structure anymore. Therefore, Goldstein and Fattal do not use the filtered trajectories directly to render the stabilized frames. Instead, they fit new fundamental matrices $\tilde{F}_{st}$, which relate the feature points on the input

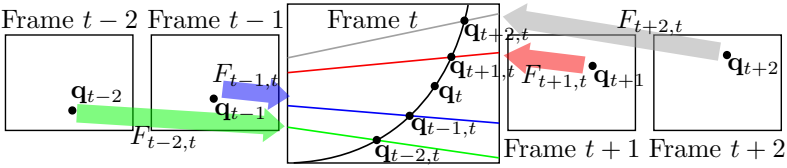


Figure 2.14: Here we show 5 consecutive frames. We map the moving point q with the fundamental matrices F_{st} onto frame t, which gives the epipolar lines, drawn in color. Together with the assumption that q moves smoothly, we can reconstruct its trajectory in frame t.

frame s to the stabilized frame t. Afterwards, they apply the epipolar point transfer with $\tilde{F}_{st}$ for several s again in order to find geometrically correct point locations on the stabilized frames. As a last step, the frames are rendered with the content-preserving warp by Liu et al. [55].

The most remarkable point in this work is probably the fact that it also addresses moving objects. The described stabilization process relies on features of static objects only. Despite, in the rendering step Goldstein and Fattal consider features on moving objects too. This makes sense, since moving objects, for example people, may be much closer to the camera than static background. Therefore, the displacement to the input image may be different to the background in such an area. The problem is that we cannot use the point transfer method for static points directly, even though we know the fundamental matrices F_{st}. The epipolar line $\mathbf{l}_s = F_{st}\mathbf{q}_s$ represents the ray of point q into camera s at time s in the view of camera t. Since q is a moving object, $\mathbf{q}_t$ does not lie on $\mathbf{l}_s$ and it is not possible to transfer a point using two or more epipolar lines to another view. Nevertheless, we can exploit this information about the point q on time s in view t and, together with the assumption that features on real world objects move smoothly, we can build an equation system. On the one side $\mathbf{q}_{s,t}$, the feature point location at time s on frame t, has to lie on $\mathbf{l}_s$. On the other side, $\mathbf{q}_{s,t} - \mathbf{q}_{s-1,t}$ should be constant for all s. After

solving this system, we get the feature location of a moving point q on time t in different views s. These $q_{t,s}$ and the fundamental matrices $\tilde{F}_{st}$ we can now use to transfer the point q_t from the input frame onto the stabilized one.

2.6.2 Bundled Camera Paths for Video Stabilization

The second work we discuss in this section is entitled "bundled camera paths for video stabilization" and was presented at SIGGRAPH 2013 by Shuaicheng Liu and colleagues [58]. Even with having the word "camera" in its title, the described method is far from computing cameras, as employed in 3D stabilization methods, or decomposing trajectories into camera and point information, as done in the subspace stabilization method (Section 2.5). The described method does not rely on any scene geometry or actual camera information at all. It is solely based on content-preserving warps [55]. People not familiar with warping methods may come back to this section after reading Chapter 4.

All previously discussed methods firstly compute goal feature locations, which are consistent with the 3D structure of the scene and then use these to guide the warp in the last step of their pipeline. Liu et al. [58] do it the other way round. The warp is used at the beginning to compute a kind of camera paths. First, they overlay a frame with a rectangular grid. Then, they warp this frame onto the next frame with the content preserving warp [55]. The warping is guided through feature points, which were found on the current and the following frame. The transformation that each grid cell undergoes can be described by a homography, which is defined by the four grid cell vertices. These homographies can be interpreted as camera movements. To do so, we treat each video frame as an image, composed of as many images as we have grid cells, each taken with an independent camera. We keep on warping the next frame to the one after the next and computing the homographies for each grid cell. This way we compute a path, consisting of the homographies from one frame

to the next, for each of our cameras.

Once the paths for all cameras are found, Liu et al. [58] set up an optimization system to smooth the camera paths. The system balances the smoothness of the camera paths and tries to keep them close to the input paths. A further requirement is that the path of each camera stays close to the one of its eight neighboring cameras. The resulting energy minimization problem is quadratic and therefore solvable as a large sparse linear system.

After finding the optimal camera paths, we can compute the output positions for each grid cell directly. Then, it is straightforward to again compute homographies, from the input frames to the output frames, and do a forward warp. If small wholes appear between neighboring cameras, these are filled with bilinear interpolation.

The way of describing the camera paths is the main point of this work and is most likely going to be used in other papers for different applications. Further this method avoids the step of finding long feature trajectories, which often can be difficult. And last but not least, this method does not need any knowledge about the used camera or the scene structure.

2.7 Stereo Video Stabilization

Camera sensors and lenses become smaller and cheaper to produce with time. Nowadays, building even two cameras into mobile phones has become economically affordable. This opens up a whole field of new possibilities. Mostly, the additional camera in mobile phones is used to create depth maps for the images. Even though this feature is not yet common for mobile phones and consumer cameras, a few of them are available on the marked[4] already. A popular example of a stereo consumer camera is the Fujifilm FinePix REAL 3D, which we used for a few experiments, too.

Wherever there are consumer video cameras, shaky videos are close. This is valid for stereo videos too and the reason for us to

[4]Wikipedia provides a list of 3D-enabled mobile phones:
`https://en.wikipedia.org/wiki/List_of_3D-enabled_mobile_phones`

discuss it in more detail. By having two views for each time step, we also gain a lot of additional information, hence a lot more possibilities. However, we also need to stabilize two camera views simultaneously.

2.7.1 3D Stereo Video Stabilization

Daniel Frey, one of our students, investigated this topic during his Bachelor thesis [17]. He implemented the 3D video stabilization algorithm by Liu et al. [55]. We had previously already thought about possible solutions to keep the camera baseline fixed while stabilizing the paths. Smoothing only one path and computing the locations for the second camera out of it? This is possible, since the two cameras of a stereo camera have a fixed distance. Or shall we add an additional constraint to the filtering step? To start with, we tested the simplest of all methods. We filtered the 3D path of the left and right cameras independently. The results were surprisingly good and we did not find visible artifacts or disparity distortions. The major problem that we faced was the structure from motion part (moving content, "flat" scene, little camera motion) similar to the case of mono-videos. Most of these problems can be solved when considering the video streams from both cameras in the reconstruction process. Smith et al. [82] had exploited this fact already.

To the best of our knowledge, the first published work which targets the stereo video stabilization is an extension of Smith et al. [82], which was presented as a demo at ICCV 2011. Since the original paper is about light field video stabilization, where the light field is captured with a 5×5 camera array, the stereo camera is just a special case of it. To briefly review, Smith et al. [82] use edge points instead of feature locations. They then match the found edge points across the other views of the same time. The camera is calibrated and therefore the baseline and focal length is known, and the depth of the matched feature point can be computed through simple triangulation. Of importance, these 3D locations are relative to the corresponding camera locations, and independent of the previous or next one. Nevertheless it is now possible to find new camera orientations and locations rel-

ative to the original one, such that the projections of the 3D points describe a smooth path. Unfortunately, in the ICCV demo, Smith et al. do not reveal how they compute the camera location for the second view, but two possible solutions are conceivable. One is to compute only one stabilized camera and use the knowledge about the fixed baseline to compute the location for the second camera. The other possibility is to compute the two output cameras simultaneously and to add an additional energy term to the system, which either controls the camera baseline or keeps the feature disparity similar to the input. Both constraints are easy to implement and do not make the energy minimization more complex, since both constraints lead to linear energy terms.

As we see, this method exploits the advantages of stereo videos directly. The method uses the possiblity to compute the depth of feature points on each frame pair, independently of whether they are on moving or static objects. Further, no long feature trajectories are needed. It is sufficient, if we can track a feature on the previous and the next frame and additionally on the second view of the current frame. The downside of this method is that it cannot stabilize low frequency shake well.

2.7.2 Subspace Stabilization for Stereo Videos

The subspace video stabilization approach that we have discussed in Section 2.5, was extended to stereo videos by its first author [57]. The main contribution of this work is the proof that feature trajectories of both views, the one from the left and the one from the right eye, lie in the same subspace. This makes it possible to find, match and track features on both views of the stereo video independently. Afterwards we can build a matrix of all feature locations in the same fashion as we did in Equation (2.30) for a mono video. Then we can use the same steps as as for mono videos. These steps are applying the moving factorization, smoothing eigen-trajectories and then multiplying the smooth eigen-trajectories with the coefficient matrix resulting in output feature locations. In the end, the smoothed trajectory matrix

is used to guide the content preserving warps, which are applied to both views on each frame separately. We have already discussed all steps in detail in Section 2.5.

In this paragraph, we assume that the trajectory matrix is already factorized. Then, the disparity of a feature trajectory can be computed as

$$\begin{aligned}\{d(t)\} &= \{(x_R(t) - x_L(t)),\, (y_R(t) - y_L(t))\} \\ &= \big((C_R^x - C_L^x)E,\, (C_R^y - C_L^y)E\big),\end{aligned} \tag{2.50}$$

where C_R^x denotes the coefficients, which describe the x-coordinate of the feature location for the right view. Similar C_L^x, C_R^y and C_L^y denote the coefficients for the y-coordinates and the left view respectively. We directly see in Equation (2.50) the impact of the filtering step on the disparity, since this means replacing E by the filtered $\hat{E}$. In a video from a horizontally aligned stereo camera have noise free features zero vertical disparity. From Equation (2.50) follows that $C_R^y = C_L^y$. Therefore, a perfect vertical disparity cannot be distorted by smoothing E. Moreover, a smooth $\hat{E}$ leads to smooth horizontal disparities and therefore to smooth changes in the perceived depth and a more pleasant 3D view. Nevertheless, a downside of this method could be that the stabilization algorithm skews the disparities over time. Liu et al. [57] report that they did not experience this in their experiments. They assume the reason is that disparities do not change significantly over time and therefore the smoothing does not have a great impact.

We conclude this section with the proof of the existence of the common subspace of the feature trajectories of two views. Liu et al. [57] show for two parallel cameras with fixed focal length that the two subspaces are the same. We do this for the more general case in which the focal length is varying. In the beginning, we stack all found feature trajectories of the left camera in a matrix M, as in Equation (2.30). We extend M by adding the trajectories from the right camera row-wise in the same manner. Similar to the proof for mono videos in Section 2.4, we consider the differences of the feature locations to the reference frames. To keep the notation simple, we

skip the trajectory number i and the frame number j. Then, u_L, v_L denote the displacement of one feature in a frame of the left view. We choose the global coordinate system such that it coincides with the coordinate system of the left reference camera. This leads to the projection $x_L = f\frac{X}{Z}$, $y_L = f\frac{Y}{Z}$ on the reference frame and we can directly re-use the proof for mono videos by simply adding the subscript L to the variables in Section 2.4.

The right camera is translated by a fixed vector $-T$ relative to the left camera. More specifically, the 3D point locations are translated by $T = (T_X, T_Y, T_Z)^\mathsf{T}$ relative to the right camera with $T_Y = T_Z = 0$, since the camera is only translated in x-direction. This leads to the projection $x_R = f\frac{X+T_X}{Z}$, $y_R = f\frac{Y}{Z}$ for a feature visible in the right eye view. Next, we compute u_R and v_R according to Equation (2.31).

$$\begin{bmatrix} u_R \\ v_R \end{bmatrix} = \begin{bmatrix} f_j \frac{(X+T_X)+\delta X}{Z+\delta Z} - f\frac{X+T_X}{Z} \\ f_j \frac{Y+\delta Y}{Z+\delta Z} - f\frac{Y}{Z} \end{bmatrix}. \tag{2.51}$$

We follow the path of the proof in Section 2.4 and replace δX, δY, δZ with the derivatives of X, Y, Z from Equation (2.34). The derivatives remain the same, since T is a constant and disappears by deriving it. After this replacement and rearranging terms, we obtain

$$\begin{aligned} \begin{bmatrix} u_R \\ v_R \end{bmatrix} = &\begin{bmatrix} -f\Omega_X \frac{(X+T_X)Y}{Z^2} + f\Omega_Y \frac{(X+T_X)X}{Z^2} + f_j\Omega_Y - f_j\Omega_Z\frac{Y}{Z} \\ -f\Omega_X \frac{Y^2}{Z^2} + f\Omega_Y \frac{XY}{Z^2} - f_j\Omega_X + f_j\Omega_Z\frac{X}{Z} \end{bmatrix} \\ &+ \begin{bmatrix} \frac{f_j t_X}{Z} - ft_Z\frac{X+T_X}{Z^2} + (f_j - f)\frac{X+T_X}{Z} \\ \frac{f_j t_Y}{Z} - ft_Z\frac{Y}{Z^2} + (f_j - f)\frac{Y}{Z} \end{bmatrix}. \end{aligned} \tag{2.52}$$

Again, we can write the right hand side as two row vectors

$$U_R = \begin{bmatrix} -\frac{(X+T_X)Y}{Z^2} & \frac{(X+T_X)X}{Z^2} & 0 & 1 & -\frac{Y}{Z} & \frac{1}{Z} & 0 & -\frac{X}{Z^2} & \frac{X+T_X}{Z} \end{bmatrix}$$

and

$$V_R = \begin{bmatrix} -\frac{Y^2}{Z^2} & \frac{XY}{Z^2} & -1 & 0 & \frac{X}{Z} & 0 & \frac{1}{Z} & -\frac{Y}{Z^2} & \frac{Y}{Z} \end{bmatrix} \tag{2.53}$$

and a column vector

$$P_j = \Big[f\Omega_X \quad f\Omega_Y \quad f_j\Omega_X \quad f_j\Omega_Y \quad f_j\Omega_Z \quad \dots \\ f_j t_X \quad f_j t_Y \quad f t_Z \quad (f_j - f) \Big]^T. \tag{2.54}$$

The row vectors only contain point-dependent entries and the column vector only camera-dependent entries. If we now compare the P_j vector to the one from the mono-video case in Equation (2.39), which is at the same time the vector for the left view in the stereo case, we find them to be identical. Hence, we found a 9 dimensional vector P_j, which can describe the camera difference from the reference camera for the left and right view at the same time. This concludes our proof.

Chapter 3

2D to 3D Conversion

Nowadays arrive many block buster movies as "3D movie" in theaters. What cinemas sell as 3D movies is usually what we call *stereo movies*. Stereo in this case means that the theater shows two movies on the screen, one movie for each eye. We discuss in the first section of this chapter why this gives the viewer the impression of depth in the scene. Two ways of producing such 3D movies can be applied. One is to film each scene with two cameras. However, this method has complicating drawbacks. First, it doubles the amount of equipment used on the set and consequently editing the movie afterwards needs additional effort. Second, finding the right distance between two cameras is a difficult task especially if the distance has to be adjusted for each scene. In the end the two cameras have a certain size and therefore a minimal distance to one another, which can not be reduced and thus making it impossible to reach the desired disparities by filming with two cameras.

The second way of producing a 3D movie is to take the shot with one single camera. This reduces the complexity of filming and allows using all methodology that has been developed over the last decades without restriction. The 3D conversion is then done after editing the movie with standard tools. Adding the 3D effect on the computer

Figure 3.1: Colors from objects far away tend to fade. The trees in the foreground are pictured in green, while the color of the ground and the sky become very similar near the horizon.

artificially also gives us the possibility to produce geometrically incorrect results. We can emphasize depth cues or reduce them to keep the scene in a comfortable viewing zone. As we will further see in this chapter, creating a second view retrospectively is very challenging and we therefore discuss problems which may occur with this method in Section 3.3. Possible solutions for the two main problems, which are finding out the depth of the scene and rendering new views, follow in Sections 3.4 and 3.5. But first of all, we want to understand what makes us perceive (Section 3.1) and display (Section 3.1) images in 3D.

3.1 3D Perception

There are a number of effects that give a viewer a cue about how far away an object had been at the time it was captured. We experience this everyday, by looking at photographs. Even if the picture is only

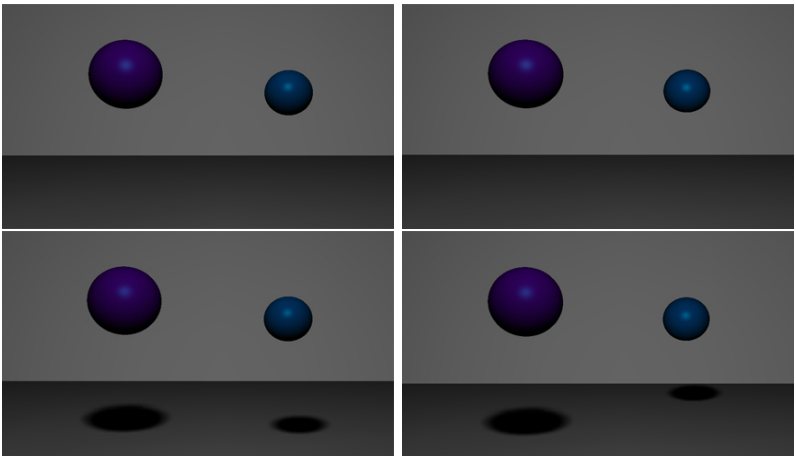


Figure 3.2: Without the shadows (top row) it is impossible to detect the distance of the viewer to the spheres. Adding the shadows (bottom row) makes clear that in the right image the right sphere is further away than the other. [67]

2D, we usually know exactly what is closer and what is further away. The maybe most obvious depth cue is the relative size of the objects. We all know that a tree is higher than a standing person. So if we see an image that shows a person being taller than the tree next to the person, we intuitively know that the tree is actually much further away from the viewer than the pictured person. We can experience a similar behavior with parallel lines, like railway tracks or roads, which in 2D come closer together the further away they go.

Whenever there are several objects on an image we can also identify the depth and order of objects by occlusion. If one object is occluded by another one, we intuitively know one is in front of the other.

Two other depth cues are color saturation and shadows. Colors tend to fade and lose intensity for objects that are far away, as can be seen in the example on Figure 3.1. This effect however clearly depends on the air pollution and the current weather but usually the effect

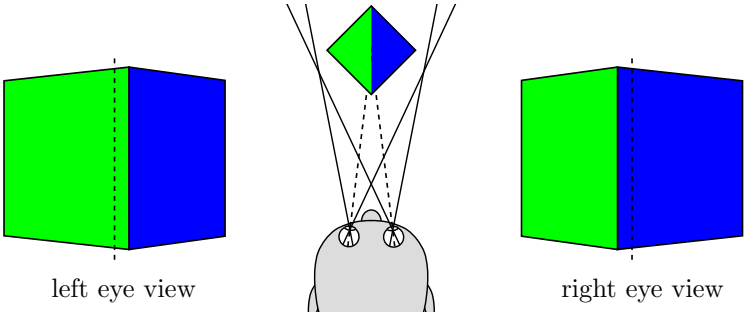


Figure 3.3: A person which looks at a green and blue colored cube in the middle of this figure. To its left and right we show the cube as it is seen from the according eye.

is noticeable. Shadows are especially important for flying objects, or objects with no obvious connection to the ground. In Figure 3.2 we show an artificial example of the importance of shadows. The two images in the top row show two spheres each. The viewer cannot determine, where the two spheres float in space. After adding shadows on the ground (bottom row), it becomes clear that in the right image, the blue sphere is further away than the purple one.

Another and even more obvious depth cue is blur. Only objects that are captured at a specific distance, in the so called *depth of field*, are captured sharp. The size of the depth of field depends on the aperture size. Parts or whole objects, which are closer or further away, and are outside of the depth of field, become blurry. The blur amount increases when objects move away from the optimal camera distance meaning they come closer or move further away from the camera.

The discussed depth cues are all *monocular depth cues*, since they give a viewer a hint of the distance from camera to object or about the order of the pictured objects, and they all work on single 2D images. However, none of them let us see the pictured scene in 3D. To create a 3D impression as perceived from real 3D scenes, we have to show the viewer two different images, one for each eye. From two

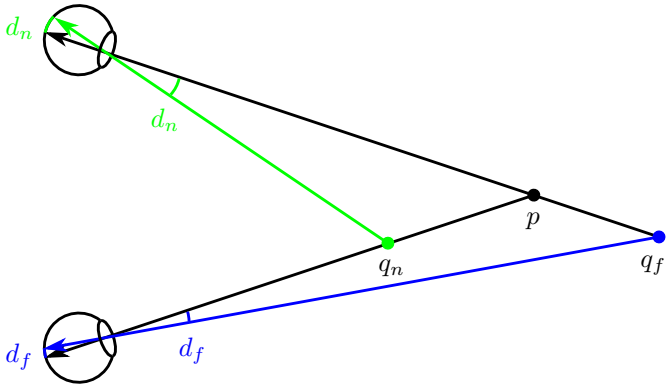


Figure 3.4: The eyes focus on point p. The left eye then sees the point q_n from a slightly different angle than p. This difference d_n, here measured in degree, we call the disparity. The same is valid for the point q_f and the right eye.

slightly different views of each eye, our brain is able to do a kind of triangulation, similar to what we discussed in Section 2.2.3 in the two-view case in the structure from motion part. Then we talk about *binocular depth cues*.

Let us investigate this in more depth. In the middle of Figure 3.3 we see a pair of eyes that look onto an edge of the cube directly from the front. One half of the cube is colored green, the other one blue. In this case being right in front of the cube means the nose of the viewer is right in front of the cube edge. The two eyes are slightly to the left and right of the frontal position. Therefore, the left and right eye see the cube from a slightly different angle and get two different images of the same cube. How those images may look is shown on the sides of Figure 3.3. We call this effect *parallax*. The extend to which the two images differ depends for one thing on the distance between the two eyes. This distance is called *baseline* and is in the range of 50-75mm for the eyes of a human. Furthermore, the distance between

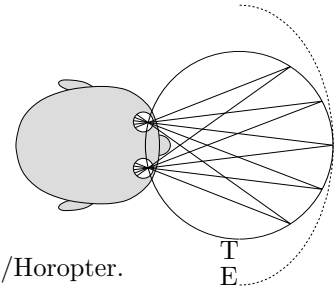


Figure 3.5: Schematic representation of the theoretical (T, solid) and the empirical (E, dotted) horopter. Image by Rainer Zenz, https://en.wikipedia.org/wiki/Horopter.

the object and the viewer has a large influence, too. We explore this by considering those points that the two eyes see under the same angle. In Figure 3.4 this is point p. The eyes are rotated slightly towards each other, such that the point p is exactly in the center of the retina in both eyes. This simultaneous movement of both eyes in opposite direction we call *vergence* in general and if they rotate towards each other it is called *convergence*. We also can say that the point p is exactly in the viewing direction of both eyes. For the right eye and the point q_n this still holds true. Assuming the eyes do not move, the point q_n is then seen from a different angle for the left eye. This difference d_n, here measured as angle, we call the *disparity*. We have the same effect for points further away than p. To keep Figure 3.4 simple, we select the point q_f such that it lies on the viewing direction of the left eye. Now q_f appears in the right eye at an angle that differs by d_f from the viewing direction.

In theory, all points that are seen under the same angle on both eyes lie on a circle (see Figure 3.5). This circle is called the *horopter*. Geometrically the horopter is the circle from Thale's theorem, generalized to non-right-angled triangles. Empirical measures show a slightly more oval curve, as shown in Figure 3.5 [97]. In either case, the horopter increases when the focused point gets farther away and the eyes *diverge*. This means, the curvature of the horopter decreases and the approximation by a plane becomes closer to reality for a short piece of it. In our case, we deal with movies and images that are displayed on a relatively small screen or quite far away in theaters. For

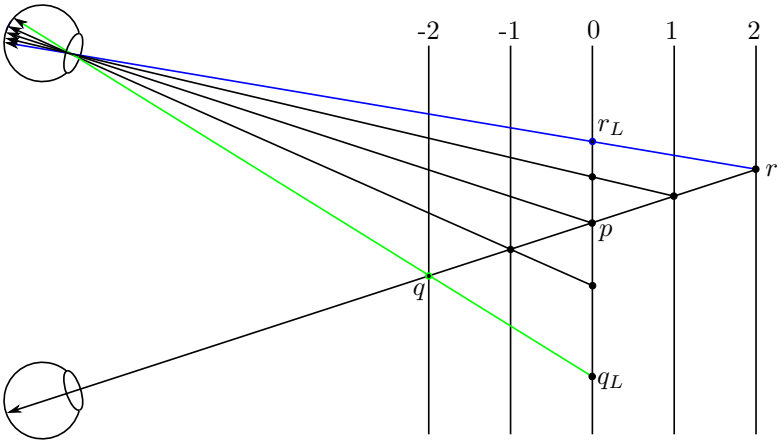


Figure 3.6: A pair of eyes focusing on point p. This defines the zero disparity plane, denoted by 0. q and r appear in front, respectively behind the zero disparity plane to the viewer. The effect is created by showing q at position q_L, r at r_L to the left eye, while both points are shown at p to the right eye.

those cases this approximation works. We call this the zero disparity plane, since objects which appear on that plane have no disparity. In Figure 3.6, the middle one of the five vertical lines is the zero disparity and denoted by 0. We set this as our screen. This makes sense because a viewer usually focusses on the display while watching a movie or looking at pictures. So if we show an image at that distance it still appears as a normal 2D image. Let us assume we can display a separate image for both eyes. In Section 3.2 we mention technologies to do that. We then can trick the viewer's eye by displaying a point q for the left eye at the position q_L and for the right eye at $q_R = p$. The viewer's brain then triangulates these two points and concludes that point q is in front of the screen and the illusion of a 3D image is created successfully. With the same trick we can also let points appear behind the screen. We show an example in Figure 3.6, with a

point r, which is displayed on the blue line for the left eye and on p for the right eye. By having a closer look at Figure 3.6, we see, that the disparities do not change linearly with the perceived depth. The disparity to create q that is the distance between p and q_L is obviously larger than the distance between r_L and p, although the distances from r or q to the zero-disparity plane are the same.

3.2 3D Displays

In the previous section, we have learned that we need to show a viewer two images to create the 3D illusion. In order to do so we need specialized hardware, such as a screen that creates or reflects polarized light. In addition, each viewer needs a pair of glasses that filters out "wrongly" polarized light for each eye. Another technology that allows reception of a 3D illusion are shutter glasses, which alternately let the light go through for one eye. On the screen we show images for the left and right eye alternating, too. The problem that arises is that screen and glasses need to be synchronised and work at the same shutter speed. The probably oldest and simplest method are anaglyph glasses. They work similar to the polarized glass with the difference that anaglyph glasses filter some color ranges, instead of polarized light. This has the obvious downside, that some colors can be seen only with one eye. Therefore gray scale images often work best for this method. Despite the fact that anaglyph is not the best method to show stereo images, we use it in this thesis too, since it is the only method which currently works on paper. However, there are also methods that can avoid glasses. We discuss this topic briefly in Chapter 5 about light fields.

3.3 2D to 3D Conversion Pipeline

In this section we explain the standard procedure to convert 2D images or videos into stereoscopic ones and discuss the problems that have to be solved.

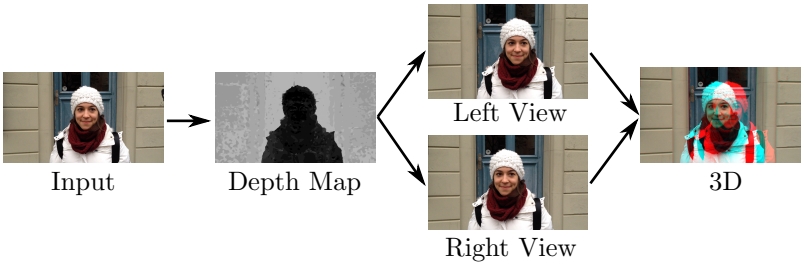


Figure 3.7: Overview of the 2D to 3D conversion pipeline. Firstly we compute a depth map, which can be converted directly into a disparity map that is used in the second step to compute two new views. The last step is then to display the two new images to each eye separately

We discussed and showed in Figure 3.6, that the disparity defines whether an object appears nearer or farther away from a viewer. Vice versa, when we create stereo images, we have to create the disparity accordingly. To do so we need to know, whether an object is in the front or the back of an image. Extracting this information is the first step of the conversion pipeline shown in Figure 3.7 and is not a trivial problem, assuming it should be solved by a computer. Methods, which try to find out about the depth of objects in images or videos are discussed in the next section in more depth. Additionally to the methods discussed in Section 3.4, we have developed a user-guided method that is presented in Chapter 6.

The second step in Figure 3.7 is rendering the images. Assume we have an image taken with a digital photo camera and information about the depth for each pixel of the image we see is given too. So, we actually have all information we need. We decide to keep the given image as the one which we show to the right eye. We then need to create a new image for the left eye. Assume the point q has a depth that is very close to the viewer, as in Figure 3.4. Originally, and therefore also on the right eye view, it is displayed on point p. But for the left eye, we need to display the point at q_L. This means, when

we compose the left eye image, we have to cut out p and paste it on location q_L. As a result, we are left with a whole at position p. This then is a classical "whole filling problem". Ideas, how to fill these wholes or even avoiding them, are discussed in Section 3.5.

The last step of the pipeline is then showing the two images to the two eyes separately. Methods how this can be done we have already mentioned in the previous Section 3.2.

3.4 Depth and Disparity Maps

To be able to create 3D images, we first need to know which objects are closer to the viewer and which are further away. When discussing methods to find out about image depth we distinguish two cases. The first one is a situation, where we have several images as input, or frames in the case of a movie, taken from slightly different locations. In this case, we can use structure from motion techniques (Section 2.2). In the other case we only have one single image and it is not possible to extract depth information with today's technology. In this case we require user input. One exception, which does not ask for user input but has other limitations, we discuss at the end of this section.

The result of all methods we discuss in this section is a depth map. A depth map is an array that comprises as many entries as there are pixels of an input image. Each entry corresponds to one pixel of the image and gives information about the depth of the object pictured at that particular pixel. From this depth map, we can then create a disparity map. We already saw in Figures 3.4 and 3.6 that the depth and the disparity are closely connected. The mathematical relation between disparity d and depth Z is $d \sim \frac{1}{Z}$. This can be illustrated as follows. We assume a setting as shown in Figure 3.8: Both cameras are pointing to the z direction, i.e. are parallel. The distance between the two cameras, the baseline b, and the first (top) camera is located at the origin of the coordinate system. Then the projection of the 3D point $\mathbf{X} = (X, Y, Z)$ onto the image of the first camera is $\mathbf{x}_1 = (X/Z, Y/Z)$. We assume a focal length $f = 1$ for simplicity. The same point is also pictured by the second camera.

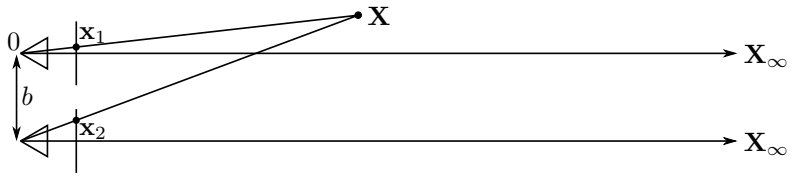


Figure 3.8: Two parallel cameras picture the 3D points $\mathbf{X}$ and $\mathbf{X}_\infty$. The first one is seen on the two images with a disparity of b/Z, while the latter is infinite far away (e.g. on the horizon) and does not create a disparity on the two images.

In its local coordinate system the 3D point $\mathbf{X}$ has the coordinates $(X - b, Y, Z)$, since the camera center is moved by b with respect to the world coordinate system. It follows, that the projection of $\mathbf{X}$ onto the second image is $\mathbf{x}_2 = ((X - b)/Z, Y/Z)$. We can then compute the disparity by taking the difference between the two point locations. In y direction this is zero. In x direction we have $d = \frac{X-b}{Z} - \frac{X}{Z} = \frac{-b}{Z}$. Note that b is a constant. Hence, we have the proposed relation $d \sim \frac{1}{Z}$.

In the case of two parallel cameras, as illustrated in Figure 3.8, points at infinity have a disparity of 0. Also see X_∞. This is still consistent with $d \sim \frac{1}{Z}$, since any finite number divided by infinity equals zero by definition. And this also seems a realistic scenario, because when scenes are captured outside, at least the pixels showing the horizon are at infinite depth. To avoid such entries in depth maps, we usually use disparity maps directly.

3.4.1 Multi-View Stereo

We begin the discussion of methods to generate disparity maps with the situation where we have several input images from different locations. As we have learned, we can then apply a structure from motion pipeline (Section 2.2). This gives us the location and viewing direction for each camera. Intrinsic parameters such as focal length or

camera center relative to the image coordinate system can be found, too. The same is true for the 3D point locations for a set of features. This feature set can be used as a starting point for a dense depth map. However, it can never be used directly because it is simply too sparse. The process of a dense reconstruction is called *multi-view stereo*, or sometimes dense multi-view stereo, to emphasize the density of the depth reconstruction.

In the classical structure from motion process we assume to have point correspondences. We then look for camera parameters and the 3D location of the given points. In the multi-view stereo problem we assume to know the camera parameters and use this knowledge to find the depth for each pixel in one image and its correspondence in the other views at the same time. This gives us a one dimensional search space for each pixel. For simplicity, we assume the coordinate system is chosen such that the camera $P_1 = [I|\mathbf{0}]$, this means its center is at the origin and it points in the z-direction, the focal length is 1. A homogeneous 3D point $\mathbf{X} = (X, Y, Z, 1)^\mathsf{T}$ becomes the homogeneous 2D point $(X, Y, Z)^\mathsf{T} = P_1\mathbf{X}$ through multiplication with the camera matrix and projects onto $(x, y) = (X/Z, Y/Z)$ on the image plane. Conversely, the pixel (x, y) shows the 3D point $(xZ, yZ, Z, 1)^\mathsf{T}$, written in homogeneous coordinates and with unknown Z. Thanks to the homogeneous notation, $(xZ, yZ, Z, 1)^\mathsf{T}$ is equivalent to $(x, y, 1, d)^\mathsf{T}$, with $d = 1/Z$, the disparity. d is always finite and we can further stick with the image coordinates. Additionally, it isolates the one unknown parameter, which we are looking for. We try to find this parameter by projecting $(x, y, 1, d)^\mathsf{T}$ for all d with P_2 onto the second image and looking for the best matching pixel. Note that the projection of $(x, y, 1, d)^\mathsf{T}$ for all d draws the epipolar line on the second view. We have now discussed the situation for two views. The extension to three ore more views is straightforward.

Most multi-view stereo algorithms use the color or brightness similarity of the pixels as a matching measure. Instead of single pixels are usually small patches considered and cross correlations computed [47, 45]. Auclair et al. [2] suggests to use SIFT descriptors because these do not assume that the objects are locally planar. Ad-

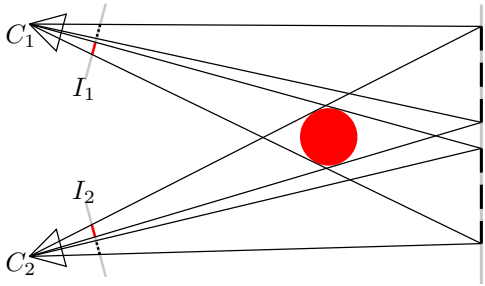


Figure 3.9: Tow cameras C_1, C_2, capture a red sphere in front of a grey wall. In image I_1 of camera C_1 the part of the wall with black dashes can not be matched to image I_2. This part in image I_2 is hidden by the sphere. Vice versa, the same applies for the dashed area in I_2.

ditional constraints such as smoothness constraints may be used. Wei et al. [96] build directly on that constraint and start with a sparse reconstruction from feature points which is then propagated through filtering. Another option is the ordering constraint, which states that the order of points along the epipolar lines is preserved [75]. We also have to take into account that for some pixels of the first view there is no matching pixel in the other views. Due to parallax effects, some parts which are visible in the first view may be hidden in the second view. In Figure 3.9 we show a sketch to this phenomenon.

Taking all constraints into account and solving the problem for all pixels at the same time leads to a large optimization problem. This can be solved as a cost or energy minimization problem [45, 47]. Others, for example Roy and Cox [75] or Hernández et al. [33], formulate the optimization problem as a graph cut problem. A list and classification of the most important technologies together with a public data set[1] to benchmark them is provided by Seitz et al. [80].

[1] `http://vision.middlebury.edu/mview/`

3.4.2 User Input

Mutli-view stereo applied to an ordinary video cannot retrieve depth information for all pixels in each shot. Whenever we have only one image or non-static content in a video, we always only have one view per time instance and the structure from motion ideas can not be applied.

Even though other depth cues exist, most algorithms that do not rely on mutli-view stereo only ask for user input. This is often combined with other techniques, as we see in a moment. First, we discuss a work that relies on user input only in order to understand the most commonly used technique. The article is called StereoBrush [91] and appeared in 2011. As already indicated in the title, the user has some kind of brush to draw sparse strokes onto an image. The brightness of the stroke then indicates depth. Brighter strokes mark objects closer to the camera. Since the brush strokes only cover a few pixel of the image, a technique is needed to propagate the information that was provided by the user. Wang et al. [91] use an idea which goes back to 2004. Levin et al. [51] developed the technique to colorize gray scale images with just a few brush strokes. The assumption is simple. If neighboring pixels have a similar intensity, they also have a similar color. Or for the StereoBrush: similar colors lead to similar disparity. Mathematically, we write the following energy term which should be minimized:

$$\sum_{p \notin M} \left\| D(p) - \sum_{q \in N(p)} w_{pq} D(q) \right\| + \sum_{p \in M} \|D(p) - M(p)\| . \tag{3.1}$$

$D(p)$ denotes the disparity at pixel p, M the set of strokes and $M(p)$ the user given disparity at p. The first term then says that the disparity of pixel p should be the same as the weighted sum of all its neighbours. The second term deals with the pixels that are marked by the user and constrains them to the user given value. The weighting function is chosen such that it sums to one, and the weights are large when $I(p)$ is similar to $I(q)$, where I denotes the input image. More details about the weighting function are not revealed by Wang et

al. [91]. But in their colorization paper Levin et al. [51] give two possible formulas and additionally a method to write the problem in terms of linear equations.

Once the concept of the scribbles is understood, it can be used in combination with many other techniques. As an example we shortly review Guttmann et al. [24]. To convert short film sequences, the user is asked to provide depth information on the first and last frame of the sequence with scribbles similar as above. With this information, a vector support machine is trained and then used to predict the depth of each remaining pixel on both the two marked frames and all frames in between. For each pixel where the reliability of the prediction is high enough, the information is included in the equation system in form of $c_4 \|D(x,y,t) - T(x,y,t)\| = 0$. $T(x,y,t)$ denotes the predicted depth of pixel (x,y) in frame t. More equations are gained similar as before. $c_3 \|D(x,y,t) - V(x,y,t)\| = 0$ for the pixels marked by a scribble, where V denotes the value defined by the user. Spatial neighbors lead to equations of the form $c_1 W_E \|D(x,y,t) - D(x-1,y,t)\| = 0$, similar in the y-direction, where W_E weights the color similarity of the two pixels. It is computed as the square root of two minus the Laplacian. To make a connection between frames, optical flow is used. Time-wise corresponding pixels lead to similar equations as spatial neighbors. The W_E is replaced by a W_M, which takes into account that the depth of an object may change over time. Guttmann et al. [24] found that, "lateral motion seldom changes the depth of the objects, while objects moving in the vertical direction may change depth. Moreover, a zoom-in or a zoom-out motion results in motion in both directions, and is likely to result in a depth change". Therefore, W_M is set according to the vertical motion of the pixel. We finally end up with a system of linear equations, in which we can give weight to the equations by choosing different c_i.

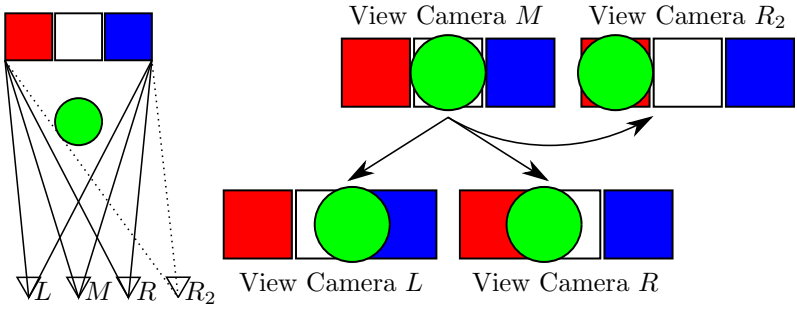
As an example of the case where the user input supports other techniques, we describe the work of Liao et al. [53]. Liao et al. work with video snippets too, and begin their pipeline with standard structure from motion. This already gives a depth value for many pixels. In the next steps further techniques like optical flow or moving objects

segmentation are used to determine relative depth. This then leads to equations of the form $D(p) = D(q)$ and inequalities $D(p) > D(q)$. Pixels, to which no depth can be assigned, are marked and shown to the user. The user has two kinds of brushes. With the first the user can mark unknown depth as being the same as other areas. With the second brush the unknown area can be marked as before or behind known areas. The depth propagation over the frames is done with the assumption that pixels with the same color have the same depth. In the end a large but sparse quadratic optimization problem results.

3.4.3 Machine Learning

After the discussion of standard methods and recent work, we also mention an article which is quite unique in its kind. We believe that in the future we will see more work following this path. The reason is simple: this method actually does what people do. It tries to understand the image by using the knowledge it has gained. Or in other words, *machine learning methods* are used in Hoeim et al. [34] to label each pixel as "ground", "vertical" or "sky". Distinguishing only these three labels appears very simple. Even though, the results by Hoiem et al. from ten years ago are impressive and already give a good information about depth, it is tempting to imagine using those information to create disparity maps. For the machine learning part colors, texture, location and geometry are taken into account. Geometry in this case means just long straight lines. Furthermore the image is split into superpixels, which are clusters of pixels with a similar color.

At the same time that Hoeim et al. published their work [34], Saxena et al. [76] also approached the extraction of 3D information from still single images with machine learning methods. Instead of superpixels they used image patches on different scale levels to exploit the monocluar depth cues. Then, their algorithm estimates a depth map directly. Later on, Saxena et al. [77] combined their method with stereo depth cues, which led to a method which outperformed existing depth-form-stereo-images algorithms. A combination of the



(a) Top View (b) Four Camera Views

Figure 3.10: (a) The setting from top: the middle or mono camera M and cameras L, R and R_2, that we want to simulate. (b) The four views of the cameras. The white box marks the holes, which we need to fill in the rendered images.

knowledge gained through the patches with an estimation of surface normals of superpixels further improved the results and gave Saxena et al. [78] the possibility to convert 2D images of quite complex scenes into photorealistic 3D models[2].

3.5 Creating Output Images

The last step in the mono to stereo conversion pipeline is the rendering of stereo views. We have already begun to discuss this problem in Section 3.3. Figure 3.10a illustrates the problem that we need to solve. The view from the mono camera M is given and we want to create two output views L and R, one for the left and one for the right eye. One solution is to use the given input view as one of the outputs which simplifies the problem because we only have to render one new image or video, instead of two. The two cameras M and R_2 in Figure 3.10a correspond to this kind of solution. However, in practice it turns out

[2] `http://make3d.cs.cornell.edu/`

that rendering two new outputs is the better approach. The holes that appear in new views are smaller and distributed to both views. This is shown in Figure 3.10b. The green sphere that appears in the middle of the input image (view camera M), moves slightly to the left and right in the case of two new output views. We show these camera views L and R in the bottom row of Figure 3.10b. The unknown region that is drawn in white appears in both views L, R and is much smaller in each view than in the view of camera R_2, which is the view we create for the case that we keep the input view M as our left eye view. We can reduce the hole creation with this simple trick, but it remains the main problem.

The most straightforward way to produce stereo output is used by Liao et al. [53]. To produce two new views they read out the disparity values from the previously created disparity map and move each pixel by half of the disparity to the left or right. Where two pixels are moved to the same output location, the decision is made from the disparity map, which is also a depth map at the same time. The empty pixels are filled with the color from the neighboring pixels that have the largest depth. Liao et al. use this simple method, because the rendering of their output is not the focus of their work and, in addition they state that this method gives sufficient results for all small holes that arise. Indeed, the results[3] look surprisingly good.

A quite similar, but more sophisticated method is image warping. The idea behind it is to stretch pixels in order to cover holes that appear, instead of copying their color. Wang et al. [91] developed a warping method which targets the mono-to-stereo conversion. Another warping method, which deals with existing stereo content on which the disparity is adjusted, has been described by Lang et al. [49]. In the latter methods the authors state that it can also be used to convert mono into stereo content. We discuss these methods in Section 4.3.2 in the chapter about image warps.

While warping techniques do not use any depth information, *image-based rendering* does. It is often used in approaches where the disparity

[3] `http://vis.uky.edu/~gravity/Research/stereolization/stereolization.htm`

map is generated through dense multi-view stereo techniques. Since this gives us the camera parameters and a 3D location for each pixel on the input frames, we can project all these points onto views of cameras that are moved slightly to the left or right. The generated point clouds can also be projected onto new views of any other frame, which is why holes that appear in new views can be filled with information from other input frames. As an example for an algorithm that uses this technique for the mono-to-stereo conversion of movies, we cite Kunter et al. [48].

Image-based rendering is not limited to stereo content production and also explored in other contexts. A recent work on image-based rendering was presented by Chaurasis et al. [11] at SIGGRAPH 2013. Chaurasis and colleagues compute the depth of super pixels instead of single pixels. These super pixels are then projected onto the view of the new camera location. All super pixels are warped individually during the projection. The warp balances the optimal projection and the preservation of the shape of each super pixel. We do this procedure for the four views which are closest to the new camera. This leads to four images, all showing the synthetically created view. The final output is then created by blending these four views. If holes remain in the output image, they are filled through solving the Poisson equations [73] with zero gradient values. This creates a blurry area which is less noticeable than a hole would be.

A method that lies somewhere between warping and classical image-based rendering is the video mesh [12]. In this method, feature points are firstly tracked over the whole video and through structure from motion techniques augmented with depth information. In a second step the feature points become vertices of the Delaunay triangulation. In order to being able to represent depth discontinuities, the user has to provide the algorithm with occlusion boundaries. Triangles that contain such depth discontinuity are cloned. Each of the two overlaying triangles holds either foreground or background pixels. To render two new camera views for each frame, we project the 3D location of the triangle vertices onto the new views. Then, the triangles can be rendered with standard methods. The authors

show successfully converted images and videos in the supplemental material[4].

We close this chapter with one more remark. Rendering stereo videos from mono videos is actually the same task, as rendering the frames for a stabilized video. In both cases, we have to synthesize new frames that are from slightly different view points than those that we got as the input. Which method solves the problem best highly depends on how much the new camera position differs from its original location. In cases in which the camera only moves a little, simpler methods such as warps do a good job. However, the more the artificial camera deviates from the original one, the more we are in need of sophisticated methods. This is nicely illustrated when we compare the video stabilization from Liu et al. [55] with the one from Kopf et al. [46]. Both video stabilization methods build on the same idea of first applying structure from motion and then camera path smoothing. In the work of Kopf et al. [46] in which totally new camera paths may be found, Liu's warping technique is not good enough anymore. Nevertheless, the image warping developed by Liu et al. does an adequate job by producing views for the just slightly filtered camera paths.

[4] `http://groups.csail.mit.edu/graphics/videomesh/`

Chapter 4

Image Warps

After having referred to image warps for computing views for new camera locations at several occasions throughout this thesis, we finally come to a detailed description of warping and dedicate an entire chapter to it. However, image warps are a much more versatile tool as they can also be used to straighten lines in wide-angle photographs [10] and panoramas [30], or to let people appear slimmer or bolder in photos [100] and videos [39]. Generally, image warps can be utilized for problems, where we need to re-arrange an image such that the ordering of all objects in the image stays the same, but the relative distance and their size may change. The probably most accessible way of illustrating image warps is the following. Usually we see an image as a rigid piece of cardboard. In contrast, by applying an image warp, we interpret the image as something like a piece of rubber or a dough, which we can stretch or squeeze as we like. This way we can create many effects without tearing a hole in the image itself.

These characteristics eventually explain why warping does work well for computing views for different camera locations. If the camera only moves a little, the order of the depicted objects does not change, but spacings between different objects and their occlusion may change due to parallax effects.

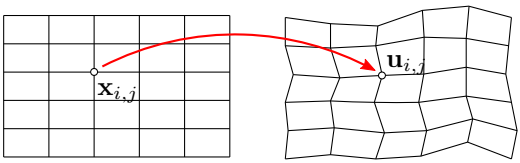


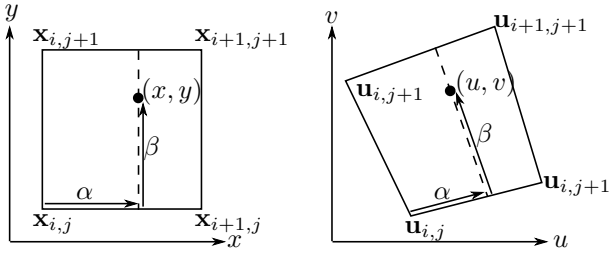
Figure 4.1: A regular grid is warped into one which satisfies given constraints.

4.1 Basics of Image Warping

In this section, we develop the mathematical model of the image warping technique and explain how we can find warps that behave as intended through the definition of so called *energy terms*. Moreover, we describe some general energy terms that are widely used in different warping applications.

Computer simulations of the deformation of physical objects, analogous to the rubber band, are often done by the finite element method. Since an image consists of many finite elements, the pixels, the finite element method seems to be a nearby way to compute warps. Indeed, there are approaches, such as the famous seam carving algorithm [3], which work on the level of single pixels. Seam carving removes or inserts pixels, instead of stretching or squeezing them, and is therefore not a warping method of the kind which we describe here.

An image warp is an $\mathbb{R}^2 \to \mathbb{R}^2$ mapping $M : (x, y) \to (u, v)$ that maps each point that may also be a sub-pixel, from the input image onto a new location in an output image. A warping algorithm searches for the mapping that maps the image to a new one which satisfies the given objectives best. In order to keep the cost for searching the best mapping low, we search for a piecewise linear mapping. Computing a piecewise linear warp means we overlay the input image with a regular grid and compute a new location for each vertex. The size of the grid and its cells depends on the applications. Some applications assume to have a grid with square cells, while others request just rectangular cells. In any case, each grid vertex has a location in pixel coordinates,



(a) Rectangular grid cell (b) General grid cell

Figure 4.2: Illustration of the bilinear interpolation coefficients α, β.

which we denote as $\mathbf{x}_{i,j} = (x, y)^{\mathsf{T}}_{i,j}$, where (i, j) denotes the vertex in the ith column and jth row. The warp is then defined through the mappings $M_{i,j} : (x, y)_{i,j} \to (u, v)_{i,j}$ of all grid vertices. Once the grid mesh is defined, we perform operations on it, until it comes to the final rendering step.

The objective of an image warp can be versatile. As an example, we can ask non-straight lines to become straight through the warp; or we can mark an object and define a specific size or position. It is unlikely that the lines or objects we want to modify are incident with the grid vertices. We consider the following basic warping problem. We want to warp an image such that the feature point (x, y) on the input image is mapped onto the coordinates (u, v) on the output image. Hence, we compute the bilinear interpolation coefficients α and β of the feature point (x, y) with respect to the four enclosing grid vertices. Then we can write the feature location as

$$a\mathbf{x}_{i,j} + b\mathbf{x}_{i+1,j} + c\mathbf{x}_{i,j+1} + d\mathbf{x}_{i+1,j+1}, \tag{4.1}$$

where the $\mathbf{x}_{i,j}$ denote the four neighboring grid vertices and

$$a = (1-\alpha)(1-\beta),\ b = \alpha(1-\beta),\ c = (1-\alpha)\beta \text{ and } d = \alpha\beta. \tag{4.2}$$

See also Figure 4.2a. This allows us to express any given feature point with grid vertices.

Most constraints we apply to an image warp come down to the example above, as we see in Section 4.3 about specific warps. We now explain how one can formulate such a constraint in a computer-readable manner by sticking with a basic example. We formulate each constraint as an equation that should be satisfied after the warp. In order to map the feature point (x, y) onto (u, v), we formulate the equation

$$a\mathbf{u}_{i,j} + b\mathbf{u}_{i+1,j} + c\mathbf{u}_{i,j+1} + d\mathbf{u}_{i+1,j+1} = (u, v)^\mathsf{T}, \tag{4.3}$$

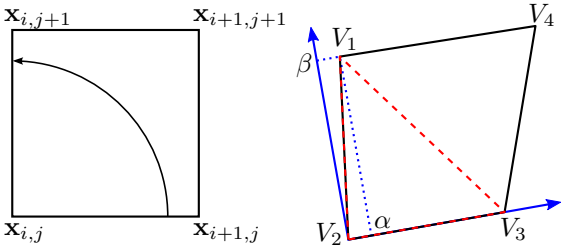
where a, b, c, d are the coefficients from Equation (4.2) and $\mathbf{u}_{i,j}$ denote the vertices of the enclosing grid after warping. Usually it is not possible to find a warp that satisfies all constraints. Therefore, we do not formulate the constraints as equations, but as an energy term. The goal is then to find the warp that minimizes the energy of the whole system. The equation above can be turned into such an energy term:

$$E = \left\| a\mathbf{u}_{i,j} + b\mathbf{u}_{i+1,j} + c\mathbf{u}_{i,j+1} + d\mathbf{u}_{i+1,j+1} - (u, v)^\mathsf{T} \right\|. \tag{4.4}$$

The energy in this case is actually the error between the feature location after the warp and the target location. These *energy minimization* problems are actual *error minimization* problems and we discuss methods to minimize them in Section 4.2.

We call the energy terms that describe the main goal of the warp *data term*. Having only data terms is not sufficient for two reasons. Firstly, we are not guaranteed to have a data term in each grid cell. This may happen in the case of a feature point-based data term, like in the example above, due to featureless regions, which appear in most images or video frames. To be able to define the warp in featureless regions too, we are in need of energies that act independently of the image content. The second reason to add other energy terms is to prevent the warp from distorting the image too much. Therefore, we often add one or both of the following two constraints to the system.

The *conformality constraint* penalizes the distortion of a single grid cell. First we explain the case of a grid consisting of squared cells. In this case, two neighboring edges of a grid cell become coincident



(a) Square grid cell (b) General grid cell

Figure 4.3: (a) Illustration of the conformality constraint for squared grid cells. In (b) we split a general quadrilateral into triangles and use a local coordinate system to express V_1 with V_2, V_3, α and β.

through a rotation of 90 degrees around the common vertex. As shown in Figure 4.3a for the edges incident on the lower left vertex, we have $\mathbf{x}_{i,j+1} = R_{90}\mathbf{x}_{i+1,j}$, where $R_{90} = \left[\begin{smallmatrix} 0 & -1 \\ 1 & 0 \end{smallmatrix}\right]$ is the matrix describing a 90 degree rotation in 2D. We can convert this constraint into an energy by replacing $\mathbf{x}$ by $\mathbf{u}$ and writing it component-wise,

$$E_c = \left\| \begin{matrix} (u_{i,j+1} - u_{i,j}) + (v_{i+1,j} - v_{i,j}) \\ (v_{i,j+1} - v_{i,j}) - (u_{i+1,j} - u_{i,j}) \end{matrix} \right\|. \tag{4.5}$$

The same idea is used to solve the problem for non-square, even non-rectangular grid cells. The method is described in Igarashi et al. [36] and was firstly used for warps by Liu et al. [55]. In a first step, we split the quadrilateral into triangles. Then we build a local coordinate system for each triangle. To do so, one triangle vertex defines the coordinate system's origin, the triangle side to its right the α-axis. Next we rotate the α-axis about 90 degrees around the origin to get the β-axis. In Figure 4.3b we illustrate this with vertex V_2 as the origin of the coordinate system. In the local coordinate system of Figure 4.3b, we can describe the location of the vertex V_1 that does not lie on the α-axis, with the two vertices V_2, V_3 and α, β-coordinates. We get

$$V_1 = V_2 + \alpha(V_3 - V_2) + \beta R_{90}(V_3 - V_2), \tag{4.6}$$

where R_{90} is again the rotation matrix. The transformation of Equation (4.6) into an energy term works the same way as in the constraints discussed previously. This conformality constraint allows the grid cell to be moved, rotated or scaled but nevertheless, it recognizes similartity deviations even after such transformations.

The conformality constraint prevents the system from too strong deformation, but it does this for each grid cell separately. To encourage the warp to behave similarly on neighboring cells, we add a *smoothness constraint*. One smoothness constraint that we introduce here was developed by Carroll et al. [10, 9]. We have described above that the warp is a mapping function $M : (x, y) \rightarrow (u, v)$. Hence, we can write the coordinates u and v as two functions $u(x, y)$ and $v(x, y)$ of the variables x, y. Then the derivative of u and v with respect to x and y describes the distortion of the grid and the second derivative describes the change of the distortion. In conclusion, if we want to have a slow and smooth change of the distortion, the second order derivative of $u(x, y)$ and $v(x, y)$ should be close to zero. The second order partial derivatives define the Hessian matrix, which are the following two 2×2 matrices in our case:

$$H(u) = \begin{bmatrix} \frac{\partial^2 u}{\partial^2 x} & \frac{\partial^2 u}{\partial y \partial x} \\ \frac{\partial^2 u}{\partial x \partial y} & \frac{\partial^2 u}{\partial^2 y} \end{bmatrix} \qquad H(u) = \begin{bmatrix} \frac{\partial^2 v}{\partial^2 x} & \frac{\partial^2 v}{\partial y \partial x} \\ \frac{\partial^2 v}{\partial x \partial y} & \frac{\partial^2 v}{\partial^2 y} \end{bmatrix}. \tag{4.7}$$

Setting the matrices to be zero and taking their symmetry into account results in three constraints from each of the two Hessian matrices. We compute them only for $H(u)$ since $H(v)$ works the same way. We compute the first order partial derivatives with the forward differences $\frac{\partial u_{i,j}}{\partial x} = u_{i+1,j} - u_{i,j}$ and $\frac{\partial u_{i,j}}{\partial y} = u_{i,j+1} - u_{i,j}$. For the second order derivative we use the backward differences,

$$\begin{aligned} \frac{\partial^2 u}{\partial^2 x} &= (u_{i+1,j} - u_{i,j}) - (u_{i,j} - u_{i-1,j}) \\ \frac{\partial^2 u}{\partial x \partial y} &= (u_{i+1,j} - u_{i,j}) - (u_{i+1,j-1} - u_{i,j-1}) \\ \frac{\partial^2 u}{\partial^2 y} &= (u_{i,j+1} - u_{i,j}) - (u_{i,j} - u_{i,j-1}). \end{aligned} \tag{4.8}$$

Similar to the conformality constraint for squared grid cells (Equation (4.5)), we combine the right hand sides of the Equations (4.8), together with those from the Hessian matrix of v, into a vector. The norm of this vector is our smoothness energy term E_s.

We now have discussed two basic energies and gave an idea how application specific data terms can look like. To balance the different energy terms, we assign weights to them before assembling the final energy minimization problem. The reason to do this is because the amount of energies for one objective may be different than the one for another objective. For example, on the one side we have one conformality energy term per grid cell. On the other side, we may have several feature points that we want to move to a new location in one grid cell. Consequently, the algorithm tends to favor the feature point constraints. Vice versa, in an application where only very few data energy terms are given, the resulting warp could be the mapping onto itself, since this has zero conformality and smoothness energy, and may minimize the overall energy best. In essence we weight the different constraints by multiplying them with a factor w. Usually, those weights are found experimentally and given by the authors of the papers which include warping techniques. Sometimes weights can also be user adjustable variables. In rare cases weights are computed by the system, based on some heuristic. We want to mention one of them here, the so called *salience* weight [55]. The idea in salience is to tell the warp in which grid cells distortions are more noticeable. To do so Liu et al. compute the L_2 norm of the color variance inside a grid cell and use this as the weight for the conformality energy. The intuition is that grid cells with a higher color variance display more textured regions and therefore distortions become more recognizable in such cells than in uniform areas. Consequently, the conformality energy needs to be weighted higher in such a cell.

Once the warp is computed, we know the output location for each vertex of the grid that we have overlaid over the input image. As the last step we render the full image as follows. For each pixel in the output image we search for its enclosing grid cell, or more specifically, its four corresponding vertices. Then we do an inverse

bilinear interpolation, meaning that we go from the right to the left in Figure 4.2. Finding the α and β values is not trivial but fortunately described well in Heckbert's Master thesis [32]. Knowing α and β we can find the source location of the new pixel on the input image. All we need to do is to look up the color in the original image, and color the new pixel accordingly. This color look-up may need another bilinear interpolation because the computed source location may lie between pixels.

4.2 Optimization Techniques

In the previous section we have learnt how we can describe a warp and formulate its objective with formulas. In this section, we discuss methods to find the warp that fits the objective best.

Since the warp is defined through its grid vertices, we are actually searching for new locations of grid vertices. Therefore, grid vertex locations become the unknowns in our optimization system. We further formulate constraints that express the objective of the warp as energy terms, for which examples are given in Equations (4.4) and (4.5). Energy terms can thus be written as

$$E_i = \|f_i(\mathbf{x})\|, \tag{4.9}$$

where the subscript i indicates that it is one out of many energies, f_i the function which describes the objective and $\mathbf{x}$ the set of variables we have. $\mathbf{x}$ may be a bit misleading at first glance, since $\mathbf{x}$ is the vector consisting of the new grid vertex locations, which we denoted as $\mathbf{u}_{i,j}$ in the previous and also next section. We use $\mathbf{x}$ in this section, to be compatible with the standard math notations, especially the one from Madsen et al. [66], where $\mathbf{x}$ denotes the unknowns. Here $\mathbf{x} = (u_{1,1}, v_{1,1}, u_{1,2}, \ldots, v_{n,m})$ is a vector consisting of the coordinates of the grid vertices.

In order to find the best warp we minimize an energy that consists of several energy terms. By neglecting the weights w_i, we have

$$E = \sum_i E_i = \sum_i \|f_i(\mathbf{x})\| = \sum_i \sqrt{f_i(\mathbf{x})^2} \tag{4.10}$$

for the total energy. Because of the monotonicity of the root function $\sqrt{f(x)^2}$ and $f(x)^2$ have the same minimum. Moreover, the minimum stays the same when we multiply a function with a constant value and therefore searching for the $\mathbf{x}$ that minimizes E is the same as finding the minimum of

$$F(\mathbf{x}) = \frac{1}{2}\sum_{i=1}^{m}(f_i(\mathbf{x}))^2. \tag{4.11}$$

4.2.1 The Least Squares Problem

If we have one smoothness or conformality constraint per vertex and additionally energy terms which describe the objective, we are guaranteed to have more functions f_i than variables. Thus, our energy minimization problem satisfies the conditions of the following definition:

Least Squares Problem Finding $\mathbf{x}^*$, the minimizer for a function $F(\mathbf{x})$ as described in Equation (4.11), with $f_i : \mathbb{R}^n \to \mathbb{R}$ and $m \geq n$, n the number of variables and m the number of functions f_i, is called a *least squares problem.*

For the further discussion of the problem we follow Madsen et al. [66] and use the Taylor expansion to write

$$F(\mathbf{x}+\mathbf{h}) = F(\mathbf{x}) + \mathbf{h}^\mathsf{T}\mathbf{g} + \frac{1}{2}\mathbf{h}^\mathsf{T}H\mathbf{h} + O(\|\mathbf{h}\|^3), \tag{4.12}$$

where $\mathbf{g}$ denotes the gradient,

$$\mathbf{g} = F'(\mathbf{x}) = \begin{bmatrix} \frac{\partial F(\mathbf{x})}{\partial x_1} \\ \vdots \\ \frac{\partial F(\mathbf{x})}{\partial x_n} \end{bmatrix}, \tag{4.13}$$

and H is the Hessian matrix,

$$H = F''(\mathbf{x}) = \left[\frac{\partial^2 F(\mathbf{x})}{\partial x_i \partial x_j}\right]. \tag{4.14}$$

If $\mathbf{x}^*$ is the minimum of $F(\mathbf{x})$, we have $F(\mathbf{x}^*) < F(\mathbf{x})$ for all $\mathbf{x} \neq \mathbf{x}^*$ and hence $F(\mathbf{x}^*) < F(\mathbf{x}^* + \mathbf{h})$ for all $\mathbf{h}$. With regards to Equation (4.12), this means $\mathbf{h}^\mathsf{T}\mathbf{g} \geq 0$ for all $\mathbf{h}$. Consequently, $\mathbf{g}(\mathbf{x}^*) = 0$[1] is a necessary condition. Furthermore $\mathbf{h}^\mathsf{T}H\mathbf{h} \geq 0$ has to be satisfied for all $\mathbf{h}$. Matrices, which fulfill this condition are called positive definite. According to Madsen et al. [66] the last term of Equation (4.12) is dominated by $\mathbf{h}^\mathsf{T}H\mathbf{h}$ and is also positive if H is positive definite.

4.2.2 Linear Least Squares Problems

A least squares problem is called a *linear least squares problem*, if all functions $f_i(\mathbf{x})$ in Equation (4.11) have the from

$$f_i(\mathbf{x}) = a_{i,1}x_1 + a_{i,2}x_2 \cdots + a_{i,n}x_n - b_i. \tag{4.15}$$

Here we can stack all $f_i(\mathbf{x})$ into a column vector $\mathbf{f}(\mathbf{x})$ of functions and write each function as the multiplication of the row vector $\mathbf{a}_i^\mathsf{T}$ and $\mathbf{x}$ minus the scalar b_i. Combining all the vectors $\mathbf{a}_i^\mathsf{T}$ into a matrix and the b_i into a vector, turns Equation (4.11) into

$$F(\mathbf{x}) = \frac{1}{2}\mathbf{f}^\mathsf{T}\mathbf{f} = \frac{1}{2}(A\mathbf{x} - \mathbf{b})^\mathsf{T}(A\mathbf{x} - \mathbf{b}). \tag{4.16}$$

We can directly compute the gradient and the Hessian:

$$\mathbf{g} = F'(\mathbf{x}) = A^\mathsf{T}A\mathbf{x} - A^\mathsf{T}\mathbf{b} \quad \text{and} \quad H = F''(\mathbf{x}) = A^\mathsf{T}A. \tag{4.17}$$

The Hessian is clearly symmetric and positive definite and consequently $\mathbf{g}(\mathbf{x}) = 0$ is a sufficient condition for $\mathbf{x}$ to be a minimum. Therefore, we have to solve

$$A^\mathsf{T}A\mathbf{x} = A^\mathsf{T}\mathbf{b}, \tag{4.18}$$

which is a linear equation system that has as many unknowns as equations. This can be solved through a Cholesky decomposition of the matrix $A^\mathsf{T}A$.

[1] Assume we have two non-zero vectors $\mathbf{h}$ and $\mathbf{g}$ such that $\mathbf{h}^\mathsf{T}\mathbf{g} > 0$. Then we have for the vector $-\mathbf{h}$, $(-\mathbf{h}^\mathsf{T})\mathbf{g} = (-1)(\mathbf{h}^\mathsf{T}\mathbf{g}) < 0$.

Another possibility to solve for $\mathbf{x}$ in Equation (4.18) is the method of the orthogonal decomposition. This method is somewhat slower than the Cholesky decomposition but numerically more stable, since it avoids forming the product $A^\mathsf{T}A$. Instead, A is decomposed with a singular value decomposition or a QR decomposition. In a QR decomposition Q is an orthogonal matrix and R is an upper triangular matrix with positive elements in its diagonal. Thus we get $A = QR$ with $Q^\mathsf{T}Q = I$ and $r_{i,j} = 0$ for $i > j$ and $r_{i,i} > 0$. It follows that $R = Q^\mathsf{T}QR = Q^\mathsf{T}A$ and we can multiply the equation system $A\mathbf{x} = \mathbf{b}$ with Q^T, which leaves us with the system $Rx = Q^\mathsf{T}b$. This can be solved by a backward substitution. According to Matlab's documentation pages, the QR decomposition is what Matlab's backslash operator uses. And this backslash operator is what we used in practice in our experiments.

4.2.3 Non-Linear Least Squares Problems

Unfortunately, not all energy terms, besides the ones discussed in Section 4.1, can be formulated as linear functions. Some examples of non-linear constraints are given in the next section, where we discuss more known applications of warps. In other words, we also have to consider the more general case in which the $f_i(\mathbf{x})$ in Equation (4.11) are non-linear and we deal with a *non-linear least squares problem*.

The standard approach to solve non-linear least squares problems is to start from any point $\mathbf{x}$ and search for a direction $\mathbf{h}$ in which the function value decreases, such that $F(\mathbf{x}) > F(\mathbf{x} + \mathbf{h})$. By making a step in that direction and using this location as our new $\mathbf{x}$ location we start over again, meaning that we search iteratively for the $\mathbf{x}^*$ that minimizes $F(\mathbf{x})$. We reach this goal when there is no more $\mathbf{h}$ that lets F decrease. In order to illustrate the procedure, we show a pseudo code in Algorithm 4.1. The iteration counter k is added to make sure that the algorithm terminates, even though it is not able to find a solution by itself.

Open problems which remain are how we can find the best possible search direction $\mathbf{h}$ and choose the step length α. While the above algo-

Algorithm 4.1 Descent method

```
k ← 0
x ← x_0
found ← false
while not found and k < k_max do
    h ←SEARCH_DIRECTION(x)
    if no such h exists then
        found ← true
    else
        α ←STEP_LENGTH(x,h)
        x ← x + αh
        k ← k + 1
    end if
end while
```

rithm solves any minimization problem, we next discuss two methods, which exploit the specific form of non-linear least squares problems, to find $\mathbf{h}$ and α.

Gauss–Newton Algorithm

Like in linear least squares problems, we denote the vector $\mathbf{f}$ that we produce by stacking up all functions $f_i(\mathbf{x})$. Again, we have $F(\mathbf{x}) = \frac{1}{2}\mathbf{f}^\mathsf{T}\mathbf{f}$. We derive the Tailor expansion of the vector $\mathbf{f}$,

$$\mathbf{f}(\mathbf{x}+\mathbf{h}) = \mathbf{f}(\mathbf{x}) + J(\mathbf{x})\mathbf{h} + O(\left\|\mathbf{h}^2\right\|), \tag{4.19}$$

where $J(\mathbf{x})$ is the Jacobian matrix of $\mathbf{f}$, consisting of the elements $\frac{\partial f_i}{\partial x_j}$, where i denotes the row and j the column. We define the first two terms of the Taylor approximation as $\mathbf{l}(\mathbf{h})$, such that $\mathbf{l}(\mathbf{h})$ approximates $\mathbf{f}(\mathbf{x}+\mathbf{h})$ for small $\mathbf{h}$,

$$\mathbf{f}(\mathbf{x}+\mathbf{h}) \approx \mathbf{l}(\mathbf{h}) = \mathbf{f}(\mathbf{x}) + J(\mathbf{x})\mathbf{h}. \tag{4.20}$$

We use this to approximate $F(\mathbf{x}+\mathbf{h})$ and define $L(\mathbf{h})$,

$$\begin{aligned} F(\mathbf{x}+\mathbf{h}) \approx L(\mathbf{h}) &= \frac{1}{2}\mathbf{l}(\mathbf{h})^\mathsf{T}\mathbf{l}(\mathbf{h}) \\ &= \frac{1}{2}\mathbf{f}^\mathsf{T}\mathbf{f} + \mathbf{h}^\mathsf{T}J^\mathsf{T}\mathbf{f} + \frac{1}{2}\mathbf{h}^\mathsf{T}J^\mathsf{T}J\mathbf{h}. \end{aligned} \tag{4.21}$$

Note that $\frac{1}{2}\mathbf{f}^\mathsf{T}\mathbf{f} = F(\mathbf{x})$. The idea of the Gauss–Newton algorithm is to use this $\mathbf{h}$ as the search direction $\mathbf{h}_{gn}$ to minimize $L(\mathbf{h})$. This search direction minimizes $F(\mathbf{x}+\mathbf{h})$ too, since $L(\mathbf{h})$ is an approximation of it. We have learnt earlier that one necessary condition for $\mathbf{h}^*$ to be a minimizer of L is that the gradient of L,

$$L'(\mathbf{h}) = J^\mathsf{T}\mathbf{f} + J^\mathsf{T}J\mathbf{h}, \tag{4.22}$$

is zero at $\mathbf{h}^*$. Actually, this condition is sufficient, since the Hessian matrix $L''(\mathbf{h}) = J^\mathsf{T}J$ is positive definite, independently of $\mathbf{h}$. Therefore, we find the search direction $\mathbf{h}_{gn}$ of the Gauss–Newton algorithm by solving

$$J^\mathsf{T}J\mathbf{h}_{gn} = -J^\mathsf{T}\mathbf{f}, \tag{4.23}$$

and use it as the $\mathbf{h}$ in Algorithm 4.1. In the classical Gauss–Newton algorithm $\alpha = 1$ in all steps. More sophisticated choices can be made with a line search, where α is computed such that it minimizes $F(\mathbf{x}+\alpha\mathbf{h})$. For more details we refer to Madsen et al. [66].

Levenberg–Marquardt Algorithm

The Gauss–Newton algorithm does not always converge and it can be costly to compute the step length α if other values than 1 are being used. The Levenberg–Marquardt algorithm computes α implicitly and is more robust than the Gauss–Newton algorithm, even though it is based on it. One step $\mathbf{h}_{lm}$ of the Levenberg–Marquardt algorithm is computed by solving

$$(J^\mathsf{T}J + \mu I)\mathbf{h}_{lm} = -J^\mathsf{T}\mathbf{f} \quad \text{with} \quad \mu \geq 0, \tag{4.24}$$

where $J = J(\mathbf{x})$ is again the Jacobian of $\mathbf{f}$, I is the identity matrix and μ the *damping parameter*. The latter is the main difference between

the two algorithms and causes the Levenberg–Marquardt algorithm also to be called the damped Gauss–Newton Algorithm. There are various reasons to add this damping parameter.

1. $\mu > 0$ guarantees that the coefficient matrix on the left hand side of Equation (4.24) is positive definite, even if J does not have a full rank. This ensures that $\mathbf{h}_{lm}$ is a descent direction [66].

2. μ actually balances two algorithms. If μ is large, we have $\mathbf{h}_{lm} \approx -\frac{1}{\mu} J^\mathsf{T}\mathbf{f}$. Since $J^\mathsf{T}\mathbf{f}$ is the gradient of F,

$$F'(\mathbf{x}) = \left[\frac{\partial F(\mathbf{x})}{\partial x_i}\right] = \left[\sum_j f_j(\mathbf{x}) \frac{\partial f_j(\mathbf{x})}{\partial x_i}\right] = J^\mathsf{T}\mathbf{f}, \qquad (4.25)$$

   and the Levenberg–Marquardt step for a large μ is a short step in the direction of the steepest descent. For a very small μ, we have $\mathbf{h}_{lm} \approx \mathbf{h}_{gn}$ and do a Gauss–Newton step. This is good towards the end of the iteration, when the algorithm is close to the minimum.

3. By a closer look at 2, we see that μ influences not only the direction, but also the step length. This makes a line search to find the best step length dispensable.

The next question is how to choose μ. If $F(\mathbf{x}) > F(\mathbf{x} + \mathbf{h}_{lm})$, the step is too big and we have to increase μ. Often, this simple indicator is used. By a closer look, this is also the case in Algorithm 4.2, where we test $\varrho > 0$. ϱ denotes the *gain ratio*, which is the ratio between the actual and predicted decrease of the function value,

$$\varrho = \frac{F(\mathbf{x}) - F(\mathbf{x} + \mathbf{h}_{lm})}{L(\mathbf{0}) - L(\mathbf{h}_{lm})}, \qquad (4.26)$$

where $L(\mathbf{h})$ is defined as in Equation (4.21). Thus,

$$\begin{aligned} L(\mathbf{0}) - L(\mathbf{h}_{lm}) &= -\mathbf{h}_{lm}^{\mathsf{T}} J^{\mathsf{T}}\mathbf{f} - \frac{1}{2}\mathbf{h}_{lm}^{\mathsf{T}} J^{\mathsf{T}} J \mathbf{h}_{lm} \\ &= -\frac{1}{2}\mathbf{h}_{lm}^{\mathsf{T}} \left(2J^{\mathsf{T}}\mathbf{f} + (J^{\mathsf{T}}J + \mu I - \mu I)\mathbf{h}_{lm}\right) \\ &= \frac{1}{2}\mathbf{h}_{lm}^{\mathsf{T}}(\mu \mathbf{h}_{lm} - J^{\mathsf{T}}\mathbf{f}). \end{aligned} \tag{4.27}$$

Furthermore, $\mathbf{h}_{lm}^{\mathsf{T}}\mathbf{h}_{lm}$ and $-\mathbf{h}_{lm}^{\mathsf{T}} J^{\mathsf{T}}\mathbf{f}$ are both positive[2]. Therefore the denominator is positive and $\varrho > 0$ equivalent to $F(\mathbf{x}) - F(\mathbf{x}+\mathbf{h}_{lm}) > 0$.

The closer ϱ comes to one, the better the approximation $L(\mathbf{h}_{lm})$ of $F(\mathbf{x} + \mathbf{h}_{lm})$ becomes. We can decrease μ such that the next step is closer to the Gauss–Newton step. Otherwise, we increase μ, which leads to a smaller step going more towards the steepest descent method.

A good initial value μ_0 should be related to the size of the elements of $J^{\mathsf{T}}(\mathbf{x}_0)J(\mathbf{x}_0)$. $\mathbf{x}_0$ denotes the location where the algorithm starts its search. It is a good suggestion to use the largest element of the diagonal of $J^{\mathsf{T}}(\mathbf{x}_0)J(\mathbf{x}_0)$ multiplied with a user given number τ as μ_0,

$$\mu_0 = \tau \cdot \max\{\mathrm{diag}(J^{\mathsf{T}}(\mathbf{x}_0)J(\mathbf{x}_0))\}. \tag{4.28}$$

Last but not least we need to discuss the stopping criteria. Obviously, we confirm that the gradient of F is zero or close enough to zero,

$$\left\| J^{\mathsf{T}}(\mathbf{x})\mathbf{f} \right\|_{\infty} < \epsilon_1. \tag{4.29}$$

Further, we check if the algorithm proceeds or stalls and stop if the steps become too small,

$$\|\mathbf{h}_{lm}\| = \|\mathbf{x}_{new} - \mathbf{x}\| < \epsilon_2(\|\mathbf{x}\| + \epsilon_2). \tag{4.30}$$

Both, ϵ_1 and ϵ_2 are user chosen parameters and only make sense, if they are greater than zero. Finally, we have our life insurance $k < k_{max}$.

[2]From Equation (4.24) and we get $-\mathbf{h}_{lm}^{\mathsf{T}} J^{\mathsf{T}}\mathbf{f} = \mathbf{h}_{lm}^{\mathsf{T}}(J^{\mathsf{T}}J + \mu I)\mathbf{h}_{lm}$.

Algorithm 4.2 Levenberg–Marquardt Algorithm [66]

$k \leftarrow 0$
$\mathbf{x} \leftarrow \mathbf{x}_0$
$\nu = 2$
$\mu = \tau \cdot \max\{\text{diag}(J^\mathsf{T}(\mathbf{x})J(\mathbf{x}))\}$
$found \leftarrow (\left\|J^\mathsf{T}(\mathbf{x})\mathbf{f}(\mathbf{x})\right\|_\infty < \epsilon_1)$
while not $found$ and $k < k_{max}$ **do**
  $k \leftarrow k + 1$
  $\mathbf{h}_{lm} \leftarrow$SOLVE($(J^\mathsf{T}J + \mu I)\mathbf{h}_{lm} = -J^\mathsf{T}\mathbf{f}$)
  if $\|\mathbf{h}_{lm}\| < \epsilon_2(\|\mathbf{x}\| + \epsilon_2)$ **then**
    $found \leftarrow true$
  else
    $\mathbf{x}_{new} \leftarrow \mathbf{x} + \alpha\mathbf{h}_{lm}$
    $\varrho \leftarrow (F(\mathbf{x}) - F(\mathbf{x} + \mathbf{h}_{lm})) \,/\, (L(\mathbf{0}) - L(\mathbf{h}_{lm}))$
    if $\varrho > 0$ **then**
      $\mathbf{x} \leftarrow \mathbf{x}_{new}$
      $found \leftarrow (\left\|J^\mathsf{T}\mathbf{f}\right\|_\infty < \epsilon_1)$
      $\mu \leftarrow \mu \cdot \max\{\frac{1}{3}, 1 - (2\varrho - 1)^3\}$
      $\nu \leftarrow 2$
    else
      $\mu \leftarrow \mu \cdot \nu$
      $\nu \leftarrow 2 \cdot \nu$
    end if
  end if
end while

4.3 Image Warping Applications

In this section we introduce some of the recent and most popular applications of the warping technique. We start by introducing the warps that build the basis for our own work, which we present in Chapters 6 and 7. After that we present warps with different objectives to illustrate the versatility of the technique and finally, we close the circle by explaining an approach similar to the finite elements method.

4.3.1 New Viewpoint for Given Images

From the both previous chapters two core questions have emerged: first, where can we find a better camera location in vicinity of the original one, and second, how do we render images from that new viewpoint. See also Figures 2.1 and 3.7. While we have discussed the first question in the previous chapters already, we now focus on the second question in this section.

Content-Preserving Warps for 3D Video Stabilization

In Section 2.3 we have discussed that Liu et al. [55] use structure from motion to find the camera location and orientation on each frame. Structure from motion additionally gives a sparse 3D point cloud consisting of the reconstructed feature points that we have tracked during the video. In order to obtain locations and orientations of a smoothly moving camera, Liu et al. firstly filter the input camera locations and orientations. In a next step, the 3D points are projected onto the new camera and finally we obtain for each frame a sparse set of features for which we know the 2D input location and the desired output coordinates.

The question that arises is how the image depicted by the frame can be modified such that features become the desired coordinates. Liu et al. answer this in a two step warping process. The first step is to compute for each frame the homography which transforms the input feature location as close as possible to the desired coordinates. The resulting homography is then applied to each frame as a pre-warp.

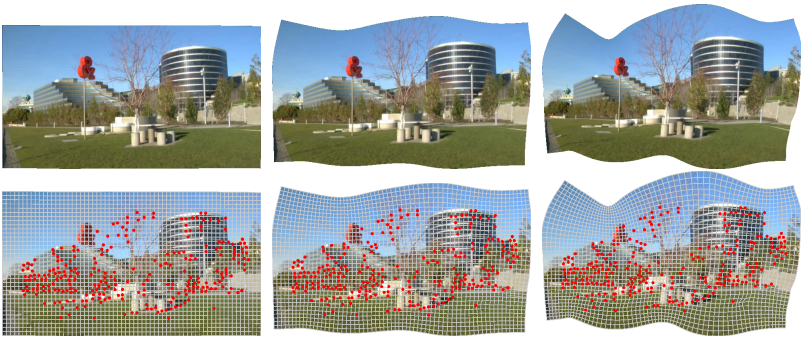


Figure 4.4: From left to right Liu et al. [55] move the camera further away from its original location. The top row shows the entire warp result before cropping; the bottom shows the corresponding grids and the points that guide the warp.

This step is important for feature less regions and applies the new camera orientation, but not its location. This can already be a good stabilization, depending on how shaky the video originally was. To apply the new camera locations we warp the image as described in Section 4.1. The data term is defined as described in Equation (4.4), where the bilinear interpolation coefficients a, b, c, and d are computed with the input feature coordinates and where (u, v) denotes the output feature coordinates. To prevent the algorithm from distorting the frames too much, the conformality constraint described in Equation (4.6) is used. Further this constraint is weighted with the salience weight that we describe at the end of Section 4.1. Also the data terms are weighted, because features appear and disappear on every frame. If a feature is in a region where it is one out of only a few, or even the only one, it has a strong influence on the warp and its appearance or disappearance may change the warp drastically. In order to get a temporally consistent warp, all features are faded in and faded out. This is done by increasing and decreasing the features' weight linearly from zero to the maximum over 50 frames after they

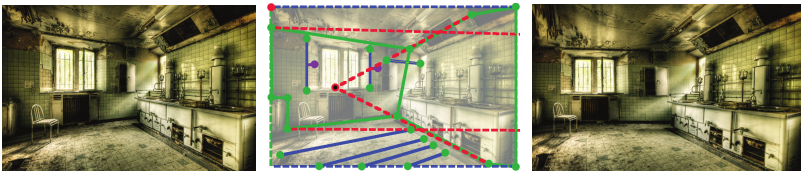


Figure 4.5: On the left we show the input image, in the middle the user given constraints and on the right the result of Carroll et al. [9].

appear and before they disappear again.

Both energies, data and conformality, are formulated as linear equations. Thus, we have to solve a linear least squares problem in the end. This makes the warping fast, which matters in video stabilization where we have to render many, sometimes several hundred frames.

Image Warps for Artistic Perspective Manipulation

The work of Carroll et al. [9] also manipulates the perspective the viewer has on a scene, but in this case we do not have a geometrically correct 3D structure that we want to preserve. Moreover, the user has to supply the geometry of the depicted objects as input (Figure 4.5, middle). The advantage, and in this case also the objective, is that the user can influence the warp directly and create an image which looks geometrically plausible to a person. The user can indicate planar regions, line segments and mark lines that have a common vanishing point. Furthermore, the user can mark fix points that are not allowed to move during the warp and can constrain line segments to be vertical or horizontal. To determine the new perspective the user moves the vanishing points in a last step. The warp is then computed such that all given geometric constraints remain valid and the vanishing lines intersect the new vanishing points. Since this work inspired us to the publication described in Chapter 6, we have a closer look at the energies, which are built from the user given constraints.

We begin with the planar regions. To make sure that the user given region behaves like a planar region during the warp, we ask the

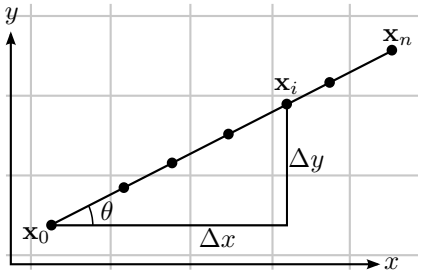


Figure 4.6: We show a line segment starting at $\mathbf{x}_0$ and its control points $\mathbf{x}_i$. The coordinate differences of $\mathbf{x}_0$ and $\mathbf{x}_i$ are given by $\Delta x = \|\mathbf{x}_i - \mathbf{x}_0\| \cos(\theta)$ and $\Delta y = \|\mathbf{x}_i - \mathbf{x}_0\| \cos(\theta)$. The ratio $\frac{\Delta x}{\Delta y} = \frac{\cos(\theta)}{\sin(\theta)}$ is the same for all $\mathbf{x}_i$. This leads to the line constraint $\Delta x \sin(\theta) = \Delta y \cos(\theta)$ formulated in Equation (4.33).

warp to be a homography on that region. We do this by fitting the homography and searching for the new grid vertex locations at the same time. This leads to the energy term

$$E_h = \sum_{i,j} \left\| \begin{bmatrix} u_{i,j} \\ v_{i,j} \end{bmatrix} - \begin{bmatrix} \frac{h_1 x_{i,j} + h_2 y_{i,j} + h_3}{h_7 x_{i,j} + h_8 y_{i,j} + 1} \\ \frac{h_4 x_{i,j} + h_5 y_{i,j} + h_6}{h_7 x_{i,j} + h_8 y_{i,j} + 1} \end{bmatrix} \right\|, \text{ where } H = \begin{bmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & 1 \end{bmatrix} \tag{4.31}$$

is the homography. The fraction in the energy term, which comes from the homogeneous division, makes this constraint non-linear.

If two planar regions share an edge, the homographies that act on the two regions have to be consistent on that edge. This means, each point on the edge, $\alpha\mathbf{x}_1 + (1-\alpha)\mathbf{x}_2$, where $\mathbf{x}_1, \mathbf{x}_2$ are the endpoints of the edge, has to be mapped onto the same point by both homographies H_a and H_b. This leads to the following homography compatibility energy,

$$E_{hc} = \|H_a(\alpha\mathbf{x}_1 + (1-\alpha)\mathbf{x}_2) = H_b(\alpha\mathbf{x}_1 + (1-\alpha)\mathbf{x}_2)\|. \tag{4.32}$$

The second type of user given constraint are straight line segments, which should be preserved during warping. Any line in 2D can be

defined by one point of the line and an angle. We use this to build a line constraint as follows. We randomly pick one of the two end points of the line and define it as $\mathbf{x}_0$. Further we select in each grid cell that the line segment crosses one line point $\mathbf{x}_i$ as a control point. We add the other end point of the line segment to the set of control points. Figure 4.6 shows an illustration of a line segment with control points in each grid cell. We request these points to be on the same line after the warp, meaning that all the lines defined by $\{\mathbf{x}_0, \mathbf{x}_i\}$ have the same angle θ. This leads to the following line energy,

$$E_l = \sum_i \left\| \cos(\theta)(v(\mathbf{x}_i) - v(\mathbf{x}_0)) - \sin(\theta)(u(\mathbf{x}_i) - u(\mathbf{x}_0)) \right\|, \quad (4.33)$$

where $v(\mathbf{x}_i)$ and $u(\mathbf{x}_i)$ denote the u and v coordinate of the point $\mathbf{x}_i$ after the warp. Depending on whether the user denotes the line as horizontal or vertical, or lets the line orientation be free, the angle θ in Equation (4.33) is fixed or a variable. In the latter case, this again is a non-linear constraint.

The vanishing line energy E_v is actually the same as E_l. The only difference is that the vanishing point $\mathbf{x}_{van}$ takes the role of $\mathbf{x}_0$, while $\mathbf{x}_0$ itself is treated like any other control point of the line. Depending on the user input, the new location $\mathbf{u}(\mathbf{x}_{van})$ of the vanishing point may be a fixed point or may be chosen by the algorithm.

The fixed point energy E_p is defined the same way as the data term in the previously discussed algorithm, the energy is defined according to Equation (4.4).

Again, to counteract excessive distortions, the conformality and smoothness energies are added to the system as we have described in Section 4.1, Equations (4.5) and (4.8). Here, the user can request straight image borders which constrains all u or v coordinates of the vertices on that border to be the same.

The total energy of the warp is the weighted sum of all energy terms and the minimization problem is non-linear, as stated before. Carroll et al. [9] use the Gauss–Newton method to find the minimal energy and therefore the warped grid mesh.

4.3.2 Stereo Image Warps

Both warping methods that we describe above can be used to create stereo content from mono content. While it seems obvious for the first method, we refer for the second to our work described in Chapter 6. Methods to create stereo content specific warps have been developed in recent years and we discuss three of the most important ones in the following.

Nonlinear Disparity Mapping for Stereoscopic 3D

The first paper that we present deals with the adjustment of the disparity of given stereo videos. The focus of the work of Lang et al. [49] lies on the function ϕ, which computes the optimal maximal disparity on each frame pair and the adjustment of all other disparities according to it. Nevertheless, for the rendering step of the new frames, Lang et al. have developed their own energy terms to warp images.

Clearly, we want the tracked features to have the desired disparity after warping the two images. Thus, the first energy term is simply

$$\|u_l(\mathbf{x}_l) - u_r(\mathbf{x}_r) - \phi(d(\mathbf{x}_l))\|\,, \tag{4.34}$$

where u_l and u_r denote the u-coordinate of the left and right warp respectively, ϕ maps the input disparity d to the desired output disparity. This first energy term gives only relative output locations. In order to also get absolute goal coordinates, we add the following constraint to the 20% temporally most stable features. Here we show only the energy term for the left image,

$$\mathbf{u}_l(\mathbf{x}_l) = \frac{\mathbf{x}_l + \mathbf{x}_r}{2} + \frac{\phi(d(\mathbf{x}_l))}{2}. \tag{4.35}$$

Similar to video stabilization [55], we need an energy term that ensures temporal coherence. Therefore, Lang et al. enforce the partial derivatives of the warp to be the same on two consecutive frames at the corresponding pixel $\mathbf{x}$,

$$\frac{\partial \mathbf{u}^t(\mathbf{x}_t)}{\partial x} = \frac{\partial \mathbf{u}^{t-1}(\mathbf{x}_{t-1})}{\partial x}, \qquad \frac{\partial \mathbf{u}^t(\mathbf{x}_t)}{\partial x} = \frac{\partial \mathbf{u}^{t-1}(\mathbf{x}_{t-1})}{\partial y}, \tag{4.36}$$

where t and $t-1$ denote the frames. This constraint is applied to the left and right frame independently, therefore we omitted the subscripts l and r.

Further we add conformality and smoothness constraints similar to the previously discussed warps. Interestingly, they are weighted according to a saliency map. This saliency map is computed not only from image-based cues but it also takes the depth of pixels in a grid cell into account. The authors noticed that people tend to focus on closer objects. Therefore the conformality and smoothness energies are weighted higher on grid cells that appear closer to the viewer.

StereoBrush: Interactive 2D to 3D Conversion Using Discontinuous Warps

We met the StereoBrush technique [91] already in the discussion of mono to stereo conversion through user input in Section 3.4.2. We have discussed how the user indicates the depth in some areas through a sparse set of scribbles and show an example in Figure 4.7. This depth is propagated over the whole image with an energy minimization. For convenience of the reader we show the energy term of Equation (3.1) again,

$$\sum_{p \notin M} \left\| D(p) - \sum_{q \in N(p)} w_{pq} D(q) \right\| + \sum_{p \in M} \left\| D(p) - \Delta(p) \right\|. \qquad (4.37)$$

Here we use p and q instead of $\mathbf{x}$ for the input coordinates, to emphasize that this warp works on pixels directly. M denotes the set of the pixels the user has marked, Δ is the assigned disparity. Like in the paper we just discussed, the data term of the warp has to ensure that each pixel gets the desired disparity. We can combine Equation (4.34) directly with the disparity propagation from Equation (4.37). This

Figure 4.7: Input image with scribbles (left), continuous disparity map (middle) and discontinuous disparity (right). Wang et al. [91].

leads to the following data term

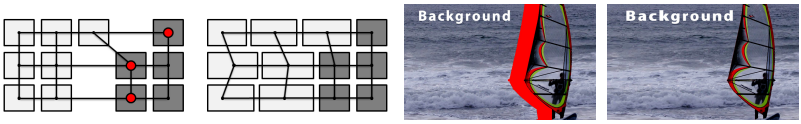
$$E_d = \sum_{p \notin M} C(p) \left\| (u_l(p) - u_r(p)) - \sum_{q \in N(p)} w_{pq} (u_l(q) - u_r(q)) \right\| + \sum_{p \in M} \left\| (u_l(p) - u_r(p)) - \Delta(p) \right\| . \quad (4.38)$$

$C(p)$ is a conformality weight, which is higher for pixels closer to user input and lower for those that are further away.

The second energy term is the smoothness energy, which is different from the one we discussed earlier. It requests pixels that probably belong to the same object to have the same relative distance to each other before and after the warp. In other words, these pixels all have the same disparity. On the other side, pixels may require a different disparity if they do not belong to the same object. To determine which pixels belong to the same object, an edge detector is applied to the image. From this the weighting function δ_{pq} is built. δ_{pq} is set to be very small, if p or q are on an edge, or an edge is between them. Otherwise, $\delta_{pq} = 1$. This is used to turn the smoothness energy term on or off.

$$E_s = \sum_p S(p) \left(\sum_{q \in N(p)} \delta_{pq} \left\| \big(u(p) - u(q)\big) - \big(x(p) - x(q)\big) \right\| \right),$$

where $S(p)$ is a saliency weight according to Goferman et al. [19].



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

Figure 4.8: (a) and (c) show the result of the first warp u, which is supposed to create holes in the image (in (c) marked red). In the second warp $\hat{u}$ are foreground pixels (in (a) marked red) fixed and the smoothness term is turned on. As a result, the background is stretched and holes filled as in (b) and (d). Wang et al. [91].

Because Wang et al. turn off the smoothness term on edges, they obtain a discontinuous warp with sharp edges in the disparity map (Figure 4.7). While these discontinuities are wanted in the depth map, they lead to holes in the output image. An example is given in Figure4.8c where the hole is marked with red. Wang et al. fix this issue with a second warp $\hat{u}$ as follows. All previously marked edges are searched for discontinuities. If a pixel p is in front of pixel q ($D(p) < D(q)$), we fix the location of p such that $\hat{u}(p) = u(p)$. Further we set $\hat{\delta}_{pq} = 1$ if $u(p)$ and $u(q)$ do not overlap. Afterwards the final warps $\hat{u}_l$ and $\hat{u}_r$ are computed with the same energy terms as before, but this time with $\hat{\delta}_{pq}$ instead of δ_{pq}. The previously fixed pixels, in Figure 4.8a marked with red dots, ensure that the foreground keeps its original shape and the disparity from the first warping step, while the smoothness term now acts everywhere except on overlaps and therefore "pulls over" the background on discontinuities and fills the holes (Figure 4.8b). Moreover, the data term ensures that the desired disparities in the stretched background are kept, too.

Enabling Warping on Stereoscopic Images

So far we have introduced warps that can convert mono images and videos into stereoscopic ones or at least adjust the perceived depth in stereo content. The work of Niu et al. [71], which we discuss next,

Figure 4.9: Applying existing warps directly onto stereo images may lead to vertical disparities as shown in the middle in an example from Niu et al. [71]. The right image shows, that their method overcomes this problem.

enables any warping technique on stereo content. The input of this method is a stereo image and a 2D image warp $\mathbf{u}_l(\mathbf{x}_l)$ which is applied to the left view. The method then finds the corresponding warp $\mathbf{u}_r(\mathbf{x}_r)$ for the right view. Applying the given warp of the left view to the right view is the first step, but not the solution. An example which shows why we still need to adjust the warp for the right view is a rotation of the image. Due to disparity, the same object in an image has a different distance to the rotation center in the left view than it has in the right view. If the images for both views are rotated the same way this leads to vertical disparities. We show an example in Figure 4.9. Another reason to treat the left and right view differently is that the horizontal disparity has to be adjusted by the warp, too. Figure 4.10 shows the example of a sphere (left) which is scaled through the warp without adjusting the disparity (middle) and with adjusting the disparity (right). In order to avoid depth distortions Niu et al. compute a disparity map for the warped stereoscopic image, such that it represents the local scaling from the 2D image warp. The scaling factor $s(p)$ is computed by fitting a similarity transformation of the coordinates of the four diagonal neighbors of pixel p before and after

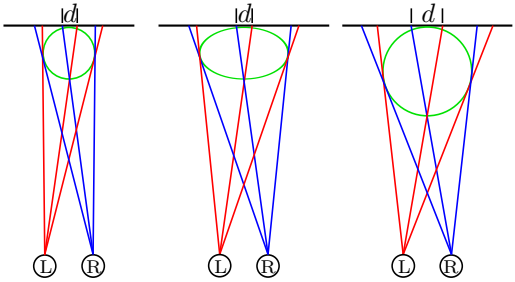


Figure 4.10: This is Figure 3 in Niu et al. [71] and shows that the disparity has to be adjusted by the warp, too. Otherwise, may objects, here the green sphere, that originally had a depth (left) appear flat (middle). In the figure right the disparity is scaled properly.

the warp $\mathbf{u}_l$. Then, we compute the new disparity map by minimizing the following energy function:

$$\sum_{p} \sum_{q \in N(p)} \left\| (\hat{D}(p) - \hat{D}(q)) - s(p)(D(p) - D(q)) \right\|, \tag{4.39}$$

where $D(p)$ and $\hat{D}(p)$ denote the disparity of pixel p before and after the warp.

The warp $\mathbf{u}_r(\mathbf{x}_r)$ for the right image is computed similarly to the content-preserving warp by Liu et al. [55]. It also consists of two steps. In a first step we apply the warp $\mathbf{u}_l$ for the left image, which is user given, to the right image, too. Afterwards, we compute the final warp with an energy minimization. The data term pushes the location of previously matched SIFT features towards the goal location, which is computed from the location in the left view plus the desired disparity $\hat{D}$,

$$E_d(\mathbf{x}_l, \mathbf{x}_r) = \left\| \mathbf{u}_r(\mathbf{x}_r) - \left(\mathbf{u}_l(\mathbf{x}_l) + \hat{D}(\mathbf{x}_l) \right) \right\|. \tag{4.40}$$

The total energy consists of the sum over the data energies for all feature pairs $(\mathbf{x}_l, \mathbf{x}_r)$ and a conformality energy as defined in Equation (4.5). Note that all energy terms are linear.

4.3.3 Wide-Angle Images and Panoramas

In all warps that we have discussed so far, our main goal was to keep the image geometrically correct or at least geometrically plausible. The next two papers we briefly summarize here actually do the opposite. They take a geometrically correct projection of a wide angle image and try to avoid unwanted artifacts. The first paper [10] straightens lines that appear as curves through the geometrically correct projection. In the second, panorama images consisting of several photographs are warped in a way that they become rectangles [30].

Optimizing Content-Preserving Projections for Wide-Angle Images

Carroll et al. [10] consider the problem that straight lines may appear as curves on images taken with a wide-angle or a fish-eye lens. The problem stems from the fact that an image is actually the projection of a sphere onto a 2D plane. It is like mapping the earth, which is a sphere too, onto a 2D map. Different types of mappings are available: perspective, mercator or stereographic mapping are the most common ones, but each of them has its downside. Either the shape is locally preserved, but area sizes are not, or it is the other way around. Carroll et al. [10] approach the problem with a warp instead of projection and receive visually pleasing images as the comparison in Figure 4.11 shows.

The user feeds an image into the algorithm and specifies its type (wide-angle, fish-eye, panorama, etc.) and the field of view in degree. Further, the user marks those curves in the image that should become straight lines in the final output. In a first step, the algorithm back-projects the image, including the lines, onto a sphere. Afterwards the warping step, which is actually the same as in many previously discussed works, follows. The only difference is that the input grid mesh now lies on a sphere and has λ (longitude) and ϕ (latitude) as coordinates, instead of x and y. So, we denote the warp of a point as $\mathbf{u}(\lambda, \phi)$.

Three energies are defined in the usual manner. These are the

Figure 4.11: The top row shows the geometrically correct mercator, perspective and stereographic projections, the bottom row shows input and result of the warping method by Carroll et al. [10].

conformality energy, the smoothness energy and one energy term for the user marked lines which should become straight. While the ideas of the constraints are the same as in other papers, the mathematical formulation changes slightly because the mapping goes from a sphere onto a plane. The conformality energy is designed following the assumption that the angle between two grid edges incident on a vertex is 90 degree. This means we can rotate one edge by $R_{90} = \left[\begin{smallmatrix} 0 & -1 \\ 1 & 0 \end{smallmatrix}\right]$ to become incident with the other. The conformality energy in Equation (4.5) bases on the above and the additional assumption that the input grid mesh consists of square cells. But we have no square cells, since the input grid lies on a sphere. The edge of a grid cell on the sphere consists of two great arc segments and two arc segments. All arc segments have the same length in degree, but not the same length in Euclidean metrics. The length of the circles representing the longitudes depend on their latitude, as we show in Figure 4.12. Thus, we have to take this into account and Equation (4.5) for the conformality

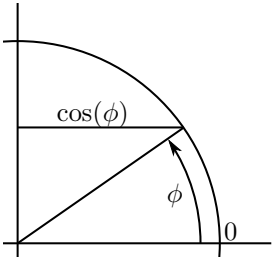


Figure 4.12: We show a quarter of a vertical cut through the center of a sphere. The bottom horizontal line is the radius of the sphere and the great circle measuring the longitude at latitude 0 (equator). For simplicity we let the radius be 1. At latitude ϕ, the radius of the longitude circle is $\cos(\phi)$.

energy turns into

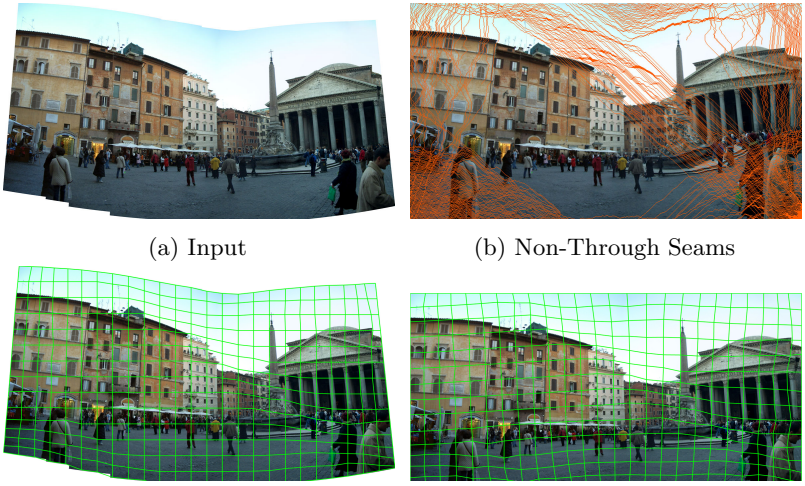
$$E_c = \left\| \begin{matrix} (u_{i,j+1} - u_{i,j}) + (v_{i+1,j} - v_{i,j}) / \cos\phi_{i,j} \\ (v_{i,j+1} - v_{i,j}) - (u_{i+1,j} - u_{i,j}) / \cos\phi_{i,j} \end{matrix} \right\|. \tag{4.41}$$

In a similar manner, the smoothness energy from Equation (4.8) is adjusted. For the details we refer to Carroll et al. [10].

The line energy, which constrains the marked lines to become straight, is independent of the input grid and we could re-use Equation (4.33) directly. But we have seen that this line energy term is non-linear, at least if the angle is a free parameter, too. Instead of using a non-linear solver, Carroll et al. solve the problem iteratively. In each step they alternately keep the direction of the line segment or the distance of the control points to the end point fixed. That way Carroll et al. have to minimize a linear energy term in each iteration. They state that the optimiziation converges after only a few iterations and therefore use the fixed number of 8 iterations.

Rectangling Panoramic Images via Warping

He et al. [30] target panorama images, which consist of several stitched photographs. Lines, which appear as curves instead of straight lines, mark a problem in that kind of images, too, but the main objective of He et al. is to obtain straight borders of the panorama image. The simplest solution to find straight borders is cropping the panoramas, but this leads to a loss of information.



(a) Input (b) Non-Through Seams

(c) Mesh Warped Backwards (d) Output with Grid Mesh

Figure 4.13: He et al. [30] first use seams [3] to make the input rectangular (b) and overlay it with a regular grid, which is warped backwards onto the input image (c) and lastly warped onto a rectangle (d).

The approach presented by He et al. [30] is a two step warp. First, they use the seam carving [3] algorithm to find so called non-through seams where pixels can be shifted left, right, up or down, until the image boarder becomes rectangular. The seams are searched in a greedy manner. In each step, the border with the longest missing pixel line is selected. The image is then temporally cropped to the height and width of that pixel line. Then, seam carving is applied to this sub-image. As shown in Figure 4.13b, the result is a rectangular image, which contains small cracks, which could be filled as in Avidan and Shamir [3]. However, He et al. suggest another approach.

He et al. overlay the found rectangular image with a regular, rectangular grid mesh. Then the grid mesh is warped backwards onto the original input image (Figure 4.13c). This is done by taking the shifts of the pixel from the seam carving step into account and eventually gives us a non-regular grid on the input image. Now we warp the grid a second time to produce the final output image (Figure 4.13d). This warp has many parallels to the case of the fish-eye images. Since the grid mesh is not rectangular we have to find another shape-preservation energy. He et al. use the method described in Zhang et al. [99] to compute a similarity measure between input and output grid cells. Further, we ask for straight borders by constraining the border vertices to have the same u or v coordinate. Additionally, straight lines should be preserved by firstly using the line detection algorithm of von Gioi et al. [90] and afterwards adding line preserving energies the same way that we have already discussed. It is easy to imagine that a user could mark additional lines, which are not detected automatically or are not straight in the input. The total energy optimization is done similarly to Carroll et al. [10]. After 10 iterations the final warp is found.

4.3.4 Content Enhancement

We have discussed several warping techniques, all of which aimed at adding depth to a scence or changing the view point, while mostly preserving the geometry of the displayed scene. However, none of

(a) Input (b) Fitted Model (c) Slider (d) Result

Figure 4.14: To reshape human bodies, Zhou et al. [100] fit a 3D body model (b) to the input image (a). The user can reshape the model with meaningful sliders (c) and this then is used to guide the warp.

them manipulates an image with the objective to change its content. Even this is possible with warping techniques.

Parametric Reshaping of Human Bodies

By looking at an image of oneself, nearly everyone once thought that it would be nice to look taller, smaller, or thinner. Modifying human bodies on images is the subject of the work by Zhou et al. [100]. In the beginning, Zhou et al. fit the shape and the pose of a human 3D body model of a database to the depicted person by using a semi-automatic algorithm [86]. Then the user can adjust several parameters, such as weight, height or girth of the body or body parts.

Zhou et al. use a set of regularly sampled points on the contour and the image border to build a Delaunay triangulation. The triangulation is used as the mesh on which the warp is computed. Three energy terms build the data term. The first energy term deals with changes along the skeletal and takes care of the length of the body

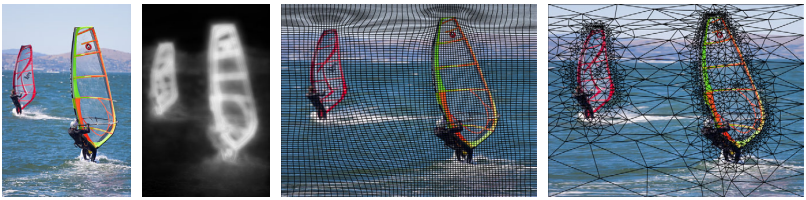
parts. The second energy term is defined perpendicular to the skeletal. It controls the girth of body parts. The skeleton is roughly determined through the body model. The third energy term that is introduced considers the length of the contours to avoid drastic changes to them. As counterweights, a smoothness and a distortion energy term are used. The smoothness term measures the difference of the transformation of neighboring triangles and the distortion energy penalizes any deformation of the triangles. The total energy is the weighted sum of all five mentioned energies, where the weights of the energies are influenced by a saliency map.

MovieReshape: Tracking and Reshaping of Humans in Videos

The work by Jain et al. [39] sounds like the natural follow up of the previously discussed paper. Indeed, there are many similarities, but the authors are different and the year of publication is the same. The two papers share the basic idea and they begin with fitting a 3D model of a human body. The difference is that in videos the poses may change over time and are therefore much more important. Again, as in Zhou et al. [100], the model can be modified by the user through a set of physically meaningful sliders. The user interacts on one frame of the sequence and gets immediate feedback. Once the user is satisfied with the result, the changes are applied to all frames of the sequence. The warp of the frames is done with the meshless *moving least squares* deformation, introduced by Müller et al. [69] and Schaefer et al. [79].

4.3.5 Finite Elements

At the beginning of this chapter, we introduced the warp as an $\mathbb{R}^2 \rightarrow \mathbb{R}^2$ mapping $M : (x, y) \rightarrow (u, v)$. We considered the finite element method to find the best possible warp. But then we used the finite differences method to approximate the derivatives of the warp and solved it for a specific set of points, the grid vertices. This works well in the case of square input grids because the differences of neighboring grid vertices represent the partial derivatives of the warp. So, $\frac{\partial u}{\partial x}$ at $(x_{i,j}, y_{i,j})$ is approximated by $u_{i+1,j} - u_{i,j}$, similar for $\frac{\partial u}{\partial y}$, $\frac{\partial v}{\partial x}$, and $\frac{\partial v}{\partial y}$.



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

Figure 4.15: Kaufmann et al. [43] double the width of the image (a) with a standard warp that has 6767 DOF (c) and their FEM framework with 1325 DOF (d). (b) shows the used saliency map.

We used this in the conformality energy in Equation (4.5) to ensure that the rectangular cell shape is preserved by the warp. Nevertheless we realized that it can become difficult to formulate such constraints for non-square grid cells [55, 10, 30]. This is a good reason to come back to the idea of the finite element method. Kaufmann et al. [43] built on that a warping framework that is completely independent of the structure of the mesh. This gives us the ability to use triangle meshes and irregular meshes that are denser in more salient regions than in other areas. We show an example in Figure 4.15, which also compares the number of degrees of freedom of a warping technique based on a regular grid and the warp by Kaufmann et al. [43].

Finite Element Image Warping

We start again from scratch to better understand the ideas of Kaufmann et al. [43] in the last warping paper that we discuss. An image warp is a function that maps an image location $\mathbf{x} = (x, y)$ onto a new location $\mathbf{u}(\mathbf{x}) = (u(\mathbf{x}), v(\mathbf{x}))$. The function itself is unknown, but it should fulfill given properties like mapping some feature points to given locations, keeping specific lines straight and usually distorting the image as little as possible. We can formulate all these desires mathematically as energy terms, such that the energy decreases when the warp better fulfills the objectives.

Now we consider the warp $\mathbf{u}(\mathbf{x})$ as a continuous function. The energy terms that we have defined in all previously discussed papers now become integrals instead of sums. For example the conformality constraint from Equation (4.5) turns into

$$E_C = \int_\Omega \left| \frac{\partial v}{\partial x}(\mathbf{x}) + \frac{\partial u}{\partial y}(\mathbf{x}) \right|^2 + \left| \frac{\partial v}{\partial y}(\mathbf{x}) - \frac{\partial u}{\partial x}(\mathbf{x}) \right|^2 d\mathbf{x}, \tag{4.42}$$

where Ω denotes the domain of the input image. The difference to Equation (4.5) is that Equation (4.42) demands for the conformality everywhere on the image while Equation (4.5) does that on the grid vertices only. The price we pay is that the problem of finding the warping function that minimizes the total energy becomes much more difficult. Since the total energy is a sum of integrals, we now have to compute integrals. Here, the finite element method steps in. It consists of two steps. First, we again split the image domain Ω in disjunct regions Ω_k, such that $\Omega = \cup_k \Omega_k$ and $\Omega_k \cap \Omega_l = \emptyset$ for $k \neq l$. Second, we define a set of basis functions $\phi_i : \Omega \to \mathbb{R}$. Then, we approximate the warping function with a linear combination of this functions,

$$\mathbf{u}(\mathbf{x}) = \big(u(\mathbf{x}), v(\mathbf{x})\big) = \left(\sum_i a_i^u \phi_i(\mathbf{x}), \sum_i a_i^v \phi_i(\mathbf{x}) \right), \tag{4.43}$$

where a_i^u and a_i^v are the weights of the function ϕ_i in the approximation of $u(\mathbf{x})$. It is now up to us, to choose a good set of functions ϕ_i. Often the ϕ_i are chosen as hat functions such that we have for each mesh vertex $\mathbf{x}_j$ one function ϕ_i with $\phi_i(\mathbf{x}_j) = 1$ if $i = j$ and $\phi_i(\mathbf{x}_j) = 0$ for all $i \neq j$. Then, we have a linear approximation of $\mathbf{u}(\mathbf{x})$ on each mesh cell and correct values on the vertices.

While in the beginning we had the function $\mathbf{u}$ as the unknown, we have now a set of scalars $\{a_i^u, a_i^v\}$ as unknowns. To give an example, we re-write the conformality energy

$$E_C = \sum_k \int_{\Omega_k} \left| \sum_i a_i^v \frac{\partial \phi_i}{\partial x} + \sum_i a_i^u \frac{\partial \phi_i}{\partial y} \right| + \left| \sum_i a_i^v \frac{\partial \phi_i}{\partial y} - \sum_i a_i^u \frac{\partial \phi_i}{\partial x} \right| d\mathbf{x}.$$

The per-element integrals, these are the integrals on each Ω_k, can be approximated using a numerical quadrature rule [43] and the coefficients $\{a_i^u, a_i^v\}$, which minimize the energy, can be found with the Newton method [66].

We motivated the use of the finite element method with the ability to also handle non-square grid cells. Indeed, this is one of the advantages of this method. Kaufmann et al. [43] use triangles as mesh cells and show the ability to have non regular meshes, which are denser in more salient regions than in others. Therefore, they are able to produce visually similar results with much fewer vertices than other algorithms with square meshes. Another advantage is the freedom to select the basis functions ϕ_i. In the classical approach the interpolation inside one mesh cell is restricted to linear or bi-linear interpolation. Depending on the warp, this may not be a good approximation to $\mathbf{u}$. With the finite element method we are able to use polynomial basis functions. Kaufmann et al. proposed the basis functions 1, x, y, x^2, xy, y^2 to approximate $\mathbf{u}$ quadratically. It is even possible to select, with some restrictions, a different set of basis functions on each grid cell.

Chapter 5

Light Fields

The interpretation of light as a field, like we know the magnetic field, goes back to the 19th century and Michael Faraday. While a magnetic field is a vector field, consisting of one direction and one magnitude on each location in the field, a *light field* describes the light traveling in every direction through every point in space. This means, at each point for all directions in a light field we have vectors representing traveling light according to ray optics, also called *geometrical optics*. Figure 5.1 visualizes this.

The term light field was established in computer graphics by Marc Levoy in 1996 [52] and gained publicity through the invention of the first Lytro[1] camera in 2012. But the concept of the Lytro camera goes back to 1908 and Gabirel Lippmann [54]. Already back then Lippmann used a two-dimensional array of microlenses to capture a light field. He called it *photographie intégrale*, which means actually "complete photography". Nevertheless, it is usually translated with *integral imaging*.

The main selling point of the Lytro camera is the possibility to refocus images after the fact and the ability of showing a scene taken from a slightly different viewpoint. These two features are possible

[1] `https://www.lytro.com`

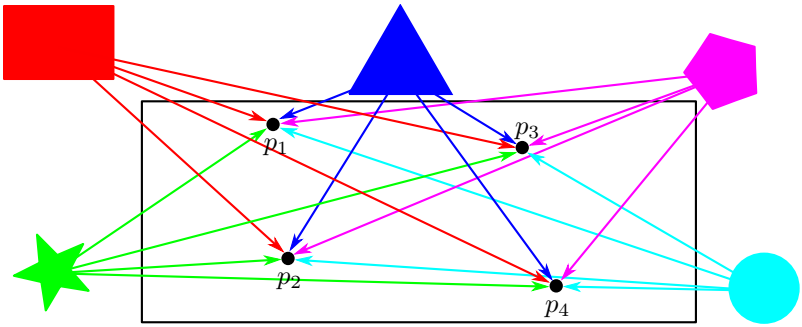


Figure 5.1: We consider the light field in the black rectangle. On each point in the field, from all directions are light rays coming. We selected four example points p_1 to p_4 and draw from each object around the light field one light ray per example point to visualize the light field.

because a light field camera system captures light rays through a microlens array which is technically the same as having an array of cameras and therefore capturing images from many slightly different positions.

In the first section of this chapter we discuss methods to describe light fields mathematically and common ways to capture light fields. Afterwards we introduce methods to compute the depth of objects seen by the light field (Section 5.2) and unveil the secret of the ability to refocus light field images after taking them. Finally, we present other applications of light field photography in Section 5.3.

5.1 Capture and Represent Light Fields

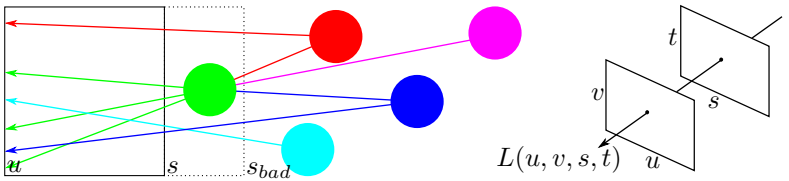
In this section we formalize the idea of light fields. We describe the light traveling with the model of geometrical optics, which describes light propagation in terms of rays that propagate along rectilinear paths in a homogeneous medium. This model works for us because it shares the assumptions we take in light field photography, too. These

are, that we have incoherent light and we deal with objects larger than the wavelength of light. Further we consider only light fields of occlusion-free spaces. The measure for the amount of light traveling along a ray is radiance. We begin with looking at the light that travels through one point in space. Then, the formal description of this light is given by the *plenoptic function*,

$$P = P(x, y, z, \theta, \phi, t, \lambda). \tag{5.1}$$

The coordinates x, y, z describe the 3D point in space where we measure the light. The two angles θ and ϕ parameterize the direction from where the incident light comes. λ is the wavelength of the light and the physical correct formulation of the color. Finally, t is the time. We try to give a more intuitive understanding of the parameters of the plenoptic function. Firstly we fix the coordinates x, y, z and the time t. We limit the angles θ and ϕ to a small range. Then summation over all λ for each θ and ϕ leads to a gray scale pinhole camera image, taken at position x, y, z and time t. If we distinguish between different λ we turn the image into a color image. Further, evaluating the plenoptic functions for the same x, y, z and angle ranges for several points in time is what a layman calls "shooting a movie with a fixed camera". We can also move the camera over time along a path by denoting its location as $x(t)$, $y(t)$, $z(t)$. Neglecting the fact that a camera operator could choose the function of the path, we do not increase the degrees of freedom by moving the camera over time. We still have information about incoming light of a small field of view at one location for each point in time. Thus, the plenoptic function represents all possible images that could be taken from all viewpoints in the light field and at any time.

The plenoptic function is the most general description of the light field, but it turned out not to be the best in practice. Most often we describe the light field of a scene from outside the scene. We do this with a mathematical model introduced by Levoy and Hanrahan [52] and Gortler et al. [22] concurrently. The model describes the light that travels through a box outside of the scene. The size of the box can be chosen, depending on the application and the scene that we

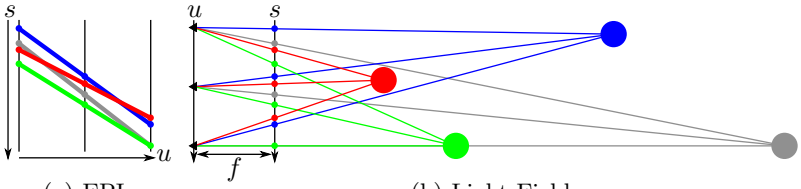


(a) Choices of the Light Field Box (b) Light Slab

Figure 5.2: (a) shows a valid (u, s) and an invalid (u, s_{bad}) choice of the two planes to parametrize a light field. (b) is a 3D illustration of a light slab.

want to capture. The only restriction is that the box has to be empty meaning it is outside of the scene, because otherwise rays may change their color while traveling through the box. As an example, we show a scene consisting of five spheres with different colors marked as circles in Figure 5.2a. We are able to capture and describe all the light, which goes through the box, in Figure 5.2a drawn in black to the left of the spheres. The light comes into the box on the side s and leaves on side u. We are free to move the box, enlarge, or squeeze it, as long none of the spheres is in the box. Otherwise, as in the example consisting of the box stretched such that it includes the green sphere, the red, blue and purple rays that enter the box on the side s_{bad}, change their color to green, when they intersect the green sphere.

There are two main advantages of this light field representation. The first is that it allows a much simpler mathematical model to describe the light field. We use a coordinate system on both ends of the box where the light rays enter and leave the box. In case of a 3D box we use the four coordinates u, v and s, t to describe the light rays. Hence, $L(u, v, s, t)$ is the light ray that enters the box at (s, t) and leaves the box at (u, v). Levoy and Hanrahan call this the *light slab* representation. Sometimes, it is also called a *4D light field* since it is represented through four parameters. An illustration of the coordinate system is shown in Figure 5.2b. As a small remark, the intuition with the light traveling through the box is only used to



(a) EPI (b) Light Field

Figure 5.3: In (b) we capture a light field of a scene consisting of four spheres. (a) shows the stack of 1D images captured along u.

explain the idea. Once the idea of the two planes that describe the light field is understood, it is easy to see that these two planes neither have to have the same size nor do they need to be parallel. Especially the size of the planes is often different in practice.

The second advantage of the light slab representation is that it is similar to how we capture light fields nowadays. Creating a light slab representation of a light field is actually very straightforward. For the moment let us focus on one point of the plane from which the light leaves our box. Then the coordinates (u, v) are fixed to one specific pair (u^*, v^*), and $L(u^*, v^*, s, t)$ describes all the light rays of the light field that go through the point (u^*, v^*). This is actually what a pinhole camera captures. If we assume that the uv- and the st-planes are parallel, then the distance between the two planes is the focal length f. The coordinates (s, t) are shifted pixel coordinates of a pinhole camera image taken on (u^*, v^*) and the center of the image is $(s - u^*, t - v^*)$. To recall the details of the pinhole camera model we refer to Figure 2.8. In conclusion, all that we need in order to capture a light field of a static scene is an ordinary camera. We move the camera along the uv-plane and take shots after equidistant steps along both axis. Figure 5.3b shows in a schematic view how we capture a light field of a scene with four spheres. To keep it simple, we show only one light ray for each sphere to each of three sample points corresponding to three camera positions on the u axis. In Figure 5.3a we show the images, which are captured by our three example cameras.

Each black vertical line represents one image and the colored dots represent the spheres on the images. The left image is the one from the top camera, the middle line corresponds to the middle camera and similarly the third line corresponds to the bottom camera.

Of course, three cameras make up a simplified example, but they may already be sufficient for simple scenes as the ones in Figure 5.3b. Connecting the corresponding dots or pixels in the images like we did in Figure 5.3a (colored lines) gives the ability to synthesize more views. For this we can draw more vertical black lines in Figure 5.3a each corresponding to a new camera or view point on u. The intersection with the colored lines then mark the pixels that show the corresponding spheres in the synthesized view. This is simplified, too since we neglected things like occlusion, shape of objects and tacitly assumed to have a Lambertian scene. There are several papers which deal with the rendering of light fields and the synthesis of novel views. The classic and most common one is by Isaksen et al. [38]. We come back to this topic in Section 5.3 and leave the discussion of Figure 5.3a with the comment that we call a cut through the light field perpendicular to the uv- and st-planes an *epipolar plane image* or shorter an *EPI*.

The presented representation of light fields became very common and it also fits our purposes best. However, we want to mention that modifications exist for the two-plane representation. Wanner and Goldlücke [93, 94] use a representation that is closer to the way the light fields are captured. They leave the uv-coordinates, which represent the camera locations used to capture the light fields, untouched. But Wanner and Goldlücke compute the st-coordinates relative to the camera location or the uv-coordinate, respectively. So, the ray $L(u^*, v^*, 0, 0)$ is the one that is perpendicular to the uv-plane and goes through the point (u^*, v^*). In the light slab representation from above this is the ray $L(u^*, v^*, u^*, v^*)$. We can come one step closer to the original plenoptic function by replacing the (x, y)-coordinates with the angles of the incoming light rays in Wanner and Goldlücke's notation.

Since we started the chapter with the Lytro camera we also want to give a brief description of how it works. At first glance a Lytro

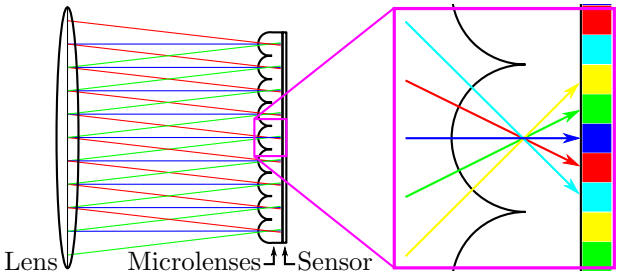
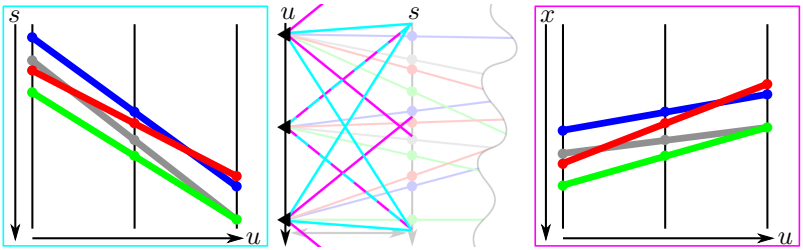


Figure 5.4: A plenoptic camera has on top of its sensor a microlens array. Each microlens covers only a few pixels, in this drawing five, which capture incoming light from different directions.

and most other *plenoptic cameras* are the same as a conventional camera. There is one aperture through which the light travels into the camera and one sensor that captures the light. The magic lies between these two things. The manufacturers places on top of the light sensor many small lenses, a so called microlens array. Each of the microlenses covers only a few pixels of the sensor (Figure 5.4). That way, it is possible to get many small images from slightly different view points, each one covering incoming light rays form different directions. Therefore, we could say that a plenoptic camera is a kind of a camera array consisting of tiny low resolution cameras. The reconstruction of an image is then performed all in software. What becomes obvious in this description is the fact that we pay for the additional features of a light field camera with a loss of resolution.

5.2 Disparity Estimation

In Section 3.4 we have already used the term *disparity estimation* when discussing mono-to-stereo conversion. There we used one view as input and computed disparities from the estimated depth of the depicted objects to being able to synthesize new views. In this section we actually are interessted in the inverse process. We use a light field



(a) EPI (b) Light Field (c) sheared EPI

Figure 5.5: Here we show two different representations of a 2D light field. In (a) we draw the light rays according to their space coordinates, (c) shows the light rays in the camera coordinate system.

as input and compute the disparities of an object that is seen by the light field. This information is then used to compute the depth of the object. We refer back to Figure 5.3 for a schematic view. In Figure 5.3b, which shows the setting from top, we see that the four differently colored circles all have a different distance to the u-line, where distance from u means depth. In the image of the leftmost camera on u (the left vertical line in Figure 5.3a), the ordering of the spheres is blue, grey, red and green. In the middle image, the red and grey spots are swapped, and on the last one, the grey is hidden by the green sphere and the red sphere became the leftmost. In summary the spheres moved with different speeds relative to the cameras. In the EPI we can make this visible by connecting the corresponding pixels with a line, as we did in Figure 5.3a. Then, the slopes of these lines are different and correspond to the depths of the objects they represent.

Earlier we stated that objects closer to the camera lead to larger disparities. Now, Figure 5.3a seems to show the opposite because the red line, which represents the closest sphere, has the flattest slope. Conversely, the grey line representing the sphere farthest away is the steepest line. To explain this we show a copy of Figure 5.3a in Figure 5.5a. The vertical black lines in Figures 5.3a and 5.5a represent the

images taken from the cameras along u. The black lines do not show the whole image but only the overlapping parts. We show the fields of view of the corresponding three example cameras in Figure 5.5b with cyan lines. Now we can see that the field of view is different for each of the images and the camera center is only for the middle camera right opposite of the center of the imaging plane on s. When we use the same field of view for all cameras, as drawn with magenta lines in Figure 5.5b, we get the sheared EPI shown in Figure 5.5c. Note that in Figure 5.5c we use x to denote the vertical axis to emphasize that we show the actual pixel coordinates. In conclusion, the colored dots that represent the spheres have now coordinates relative to the centers of the camera. Now we have the desired effect and the red sphere creates the largest disparity leading to the steepest slope for the red line. Of note, the two Figures 5.5a and 5.5c also depict the two different light field representations discussed in the previous section. Figure 5.5a shows Levoy and Hanrahan's notation and Figure 5.5c that was further used by Wanner and Goldlücke. In conclusion, only from the disparity we cannot know the absolute depth. We can only compute a relative depth that tells us which objects are closer than others and whether they are much closer or deviate only a little.

In practice, we mostly use the representation from Figure 5.5c because it is straightforward to produce such EPIs. To produce such an EPI we take all images from a row of the camera array that captured the light field and select in each image the same pixel row. These 1D images consisting of one pixel row can be combined into a new image, which is then our EPI and represents a horizontal 2D slice of a 4D light field. We can obtain also vertical slices by considering one pixel column of each image of an array column. We show an example of a horizontal EPI in Figure 5.6. It shows the same pixel row of 50 views taken from horizontally equidistant positions. The topmost pixel row stems from the leftmost camera, the bottom row from the rightmost view which leads to different axis then in Figure 5.5c: The x-axis is going from left to right, the u-axis downwards.

Unless stated differently, we assume a light field representation and EPIs as shown in Figure 5.5c for the remainder of the chapter.

Figure 5.6: This EPI consists of 50 views. Its topmost pixel row stems from the leftmost camera, the bottom row from the rightmost camera. Therefore, the u-axis points downwards and the x-axis to the right.

Our next goal is to create disparity maps for the scene captured by the light field from such EPIs. For a human viewer of the example in Figure 5.6 it is clear that the object with the white border and the brown inner part has a larger disparity and is in front of the blue and grey background. This follows directly from the slopes of the corresponding lines and is also the basic idea for all algorithms that are supposed to solve this task. We discuss three methods that compute these slopes and that also significantly influenced our own work described in Chapter 7.

Cost-Volume Filtering

Rhemann and colleagues [74] approach the problem of finding the slopes of the lines by constructing and filtering a *cost volume*. In their work they use stereo views as input and then match pixels between the two views. A stereo image pair is a special case of a light field consisting of only two views. From that view point the generalization of Rhemann's method to light fields consisting of several views is straightforward. To keep our explanation closer to the original work, we use the stereo matching terminology here.

In stereo matching we try to find for each pixel in the first view the corresponding pixel in the second view. Assuming perfect stereo images, both pixels of such a pair have the same y-coordinate. Therefore, we seek for each pixel of one view the disparity d, which tells us how many pixels we have to move in the x-direction to come to the corresponding pixel in the second view. Rhemann et al. then build up their cost volume as follows. For each triple (x, y, d) they compute a cost that tells us how expensive it is to assign the disparity

d to the pixel (x, y) of the first view. The cost is computed by the difference of the color and the color gradient in x-direction of the pixels (x, y) in the first and $(x + d, y)$ in the second view. The better these two pixels fit, the lower are the costs. Once the cost volume is built, Rhemann et al. process one xy-slice of the (x, y, d)-volume after the other. Such a slice contains the cost for one specific disparity for all pixels. Of course, this cost is different for each pixel. First, different pixels may have a different disparity since they may stem from different objects at different depths. Further, the cost for (x, y, d) and $(x, y, d + 1)$ may be very similar if the colors of the pixels around $(x + d, y)$ have similar colors in the second view. It follows that many disparities may have a low cost on one pixel. This appears in areas with low texture. However, the algorithm should assign neighboring pixels the same disparity if they belong to the same object. Instead of adding another term that creates a cost for different disparities for neighboring pixels, Rhemann et al. propose to filter each xy-slice of the cost volume. This way the costs for neighboring pixels become similar and the chance that two neighboring pixels have the lowest cost for the same disparity increases. Further, the cost computed on edges, which is usually more reliable, is propagated onto textureless regions. For choosing the filter Rhemann et al. consider that only areas belonging to the same object should be filtered. To do so Rhemann et al. use the guided filter by He et al. [31] that is an *edge aware filter* in which the filter weights are computed from the input image. This is based on the observation that neighboring pixels with similar colors are likely to belong to the same object. After filtering each slice of the cost volume independently Rhemann et al. pick for each pixel the disparity with the lowest cost.

High Resolution Light Fields

The second approach that we discuss focuses on high resolution light fields. Kim et al. [44] compute disparity maps for each of up to one hundred input images with 21 megapixels. Building a cost volume is not a feasible solution in this case because it simply demands too much memory. The advantage of this huge amount of input data is

that we have EPIs with clearly visible lines as shown in Figure 5.6. Kim et al. exploit this and reconstruct these lines. To overcome the problem of the textureless areas where lines are difficult to detect Kim et al. search first for areas with a lot of color variation along the x-axis of the EPI. Then they pick one of these pixels and test all possible disparities similarly to Rhemann et al. [74]. $E(\hat{x}, \hat{u})$ is the selected EPI pixel, where x and u denote the coordinates like in Figure 5.5c. For each disparity d Kim et al. collect all EPI pixels $\{E(\hat{x} + (\hat{u} - u)d, u) | u = 1, ..., n\}$ that correspond to the line through $E(\hat{x}, \hat{u})$ and represent disparity d. n is the number of given views and therefore the size of the EPI. For each such line they compute a score by taking the color differences of the selected pixel $(\hat{x}, \hat{u})$ to the other pixels on the line. The closer a pixel color comes to the color of the picked pixel, the higher its score is. Consequently, if a line has more pixels close to $E(\hat{x}, \hat{u})$ its score gets higher. After testing all disparities, Kim et al. assign the disparity with the highest score to the EPI pixel $(\hat{x}, \hat{u})$. Finally, they propagate the disparity to all EPI pixels that correspond to the line and have a very similar color to $E(\hat{x}, \hat{u})$. The threshold that measures the color similarity is set conservatively to not risk overwriting foreground disparities by background disparities. Once all pixels that where initially detected as being reliable are processed, Kim et al. downsample the EPI. Then newly reliable pixels are detected and processed and afterwards the EPI is downsampled again. This way, we obtain an iterative algorithm that processes each EPI independently. Altogether the memory consumption is low compared to Rhemann's method since only one EPI at the time is in the memory. Further, this method is highly parallelizeable.

Nevertheless, also Kim et al. cannot avoid using several EPIs to remove outliers in a last step. They do this with a bilateral median filter where the weights of the filter are computed from the colors of the input images. Since the filter window is small (11×11) the memory consumption stays low.

Consistent Labeling

As a third example for a recent method for disparity estimation on light fields we present the CVPR 2012 paper by Wanner and Goldlücke [93], which marked the basis for their succeeding papers [20, 94] published in CVPR 2013. In contrast to the methods before, this one uses a 4D light field as input. Similar to the two previous methods also Wanner and Goldlücke start by processing each EPI separately. Since they have a 4D light field as input they do it for all horizontal and also vertical EPIs. Wanner and Goldlücke first Gaussian filter the EPI and then compute the *structure tensor* J for each pixel of the EPI the same way that we used for corner detection in Section 2.1.1. This leads to a symmetric 2×2 matrix J. The difference between the two diagonal elements builds the first entry and two times the symmetric element of J builds the second entry of a vector that is used as the initial guess of the disparity line direction. Wanner and Goldlücke compute a reliability estimator for this direction with the elements of J.

In their work Wanner and Goldlücke then use the initial guess of the line direction on each pixel to set up an energy minimization problem that solves for the disparities of all pixels of the EPI at the same time. The energy is designed such that it is lower when the final disparity is closer to the initial guess and the initial guesses are weighted according to the reliability estimator. To avoid impossible solutions Wanner and Goldlücke add an infinite large penalty for cases in which the energy minimization suggests an impossible solution. This happens if lines of objects with smaller depth are occluded by lines of objects with larger depth, since objects closer to a camera cannot be occluded by an object that is further away.

After all EPIs are processed two depth estimates are obtained for each input view on each pixel, one from the vertical EPI and one from the horizontal EPI. To resolve this issue Wanner and Goldlücke solve another optimization problem for each view. Again, the energy is low if the final depth estimate is close to one of the two guesses. Additionally, the depth estimate for neighboring pixels has to be the same if the pixels are not on an edge. So, the cost is set high if

neighboring pixels have different depth values except the pixels are on edges. Then, the algorithm is free to choose depth values, independent of the neighbor pixels. The result of this second minimization problem is then the final depth map.

All in all, this is a very expensive method and so Wanner and Goldlücke tested their method on small data sets compared to Kim et al. [44]. Nevertheless, the produced depth maps seem to be very accurate.

5.3 Applications

In the previous section we discussed one major application of light fields: the estimation of the depth of the objects. In this section we like to explain methods to render images from light fields and to introduce a method to display a light field viewer. Today these methods are considered as "general knowledge" but back in 2000 when Isaksen et al. [38] came up with, it was a great novelty and presented at SIGGRAPH.

In the following, we sometimes consider 2D instead of 4D light fields like in Figure 5.7. Nevertheless, everything we discuss here can logically be extended to 4D.

First we discuss how we can compute new views from a light field. We assume to have a light field that is represented by two planes. Through the st-plane the rays enter the light field and leave it by going through the uv-plane. In the 2D sketch in Figure 5.7 we only see the two axes u and s. From this light field, we render two new views. One of these views is the view of camera C_1 that is actually located behind the light field. This camera view can only be rendered with the assumption that the space between the camera and the uv-plane is occlusion free. The second camera view C_2 is located inside the light field and is slightly rotated such that it is no longer parallel to the uv- and st-plane. We assume to know the 3D position of the cameras, the uv- and st-plane and also the camera matrices. Then we are able to compute a ray for each pixel of the new view of, for example, camera C_1. The ray connects the camera center and the

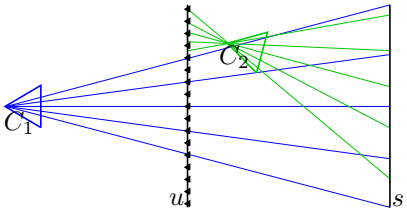


Figure 5.7: We show the rays through the light field that is defined by the lines u and s, which lead to the two camera views C_1 and C_2.

imaging plane of the camera. We then extend each ray, such that it intersects the uv- and the st-plane. The intersection points give us the four coordinates (u, v, s, t), which define each ray of the light field. All we need to do now is to look up the color or brightness of this ray and insert the information into the pixel of the new view. The same process works also for the case of camera C_2. The only difference is that we have to extend the ray from the camera center through the imaging plane also in the opposite direction to get an intersection point with the uv-plane. For both cameras, we draw five such example rays in Figure 5.7, either in blue or grey.

In practice we have to take into account that we do not know all possible rays of the light field. At least in case we captured the light field with a camera array from a real-life scene. Figure 5.7 shows such a case in which we consider the small black triangles along u to be the cameras from the array. Not all of the example rays drawn for camera C_2 hit one of these triangles. Actually, the chance that one of the desired rays (u, v, s, t) hits the center of an input camera is infinitesimal small, since the camera center is a point. The simplest solution is to pick the input camera (u^*, v^*) that is closest to the desired uv-coordinates and use the color information of the pixel representing the ray through (u^*, v^*, s, t). A much better solution is to consider the four (or in 2D two) cameras (u_i, v_i) that are closest to (u, v) and to compute the desired ray (u, v, s, t) as a weighted sum of the rays (u_i, v_i, s, t). It is also possible to consider more cameras

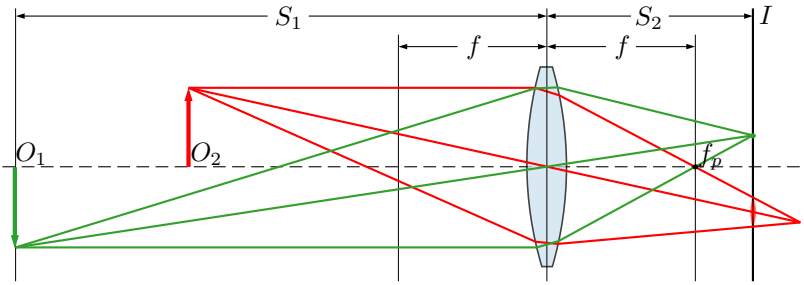


Figure 5.8: We show a thin lens with focal length f that images the object O_1 (green) at distance S_1 on the imaging plane I as S_2. The image from object O_2 (red) lies actually behind I. Therefore the object O_2 appears blurred on the image and O_2 is an out-of-focus object.

depending on the application and the density of the camera array. Isaksen et al. [38] give examples of possible weighting functions and call them *reconstruction filters*. We have the problem of a discrete number of rays also on the *st*-plane since this consists of a discrete number ob pixels. Usually on the *st*-plane the problem is smaller, since the pixel grid is denser than the camera grid on the *uv*-plane and a simple bilinear interpolation does the job.

So far we assumed pinhole cameras that have such a small aperture, the pinhole, that only one light ray from each object travels through. This way, only little light enters the camera, which either leads to a long exposure time or that we can capture only extremely bright scenes. To overcome this issue common cameras have a larger aperture, that collects more light rays, which are then bundled by a lens. Figure 5.8 illustrates a system with a *thin lens*. For two objects O_1, O_2 we show three rays each that travel through the lens and are focused by the lens onto one point, the *image point*. The outer two light rays for each point symbolize the additional light that is collected by the lens, while the middle ray also would be captured by a pinhole camera. Rays that travel along the lens axis (dashed line in Figure 5.8) focus at the

focal point f_p at a distance f (*focal length*) from the lens. In other words, the image point of an object infinitely far away is at the focal point. For objects at a finite distance S_1 we can compute the distance S_2 of the image point to the lens with the thin lens formula,

$$\frac{1}{S_1} + \frac{1}{S_2} = \frac{1}{f}. \tag{5.2}$$

Note that if $S_1 = \infty$ we have $\frac{1}{S_1} = 0$ and $\frac{1}{S_2} = \frac{1}{f}$ and therefore the equation also holds for objects at infinity.

From the thin lens formula we see that the distance S_2 varies dependent on S_1, while f is a constant for each lens. In conclusion, if we place the camera sensor at distance S_2, where a specific object with distance S_1 to the camera has its image point, other objects may have their image point at other distances. Figure 5.8 shows this scenario, where we put the imaging sensor I at the distance S_2 behind the lens such that the object O_1 (green) has its image point on it. But then the image point of object O_2 (red) is further away than I, which leads to the fact that rays from O_2 hit I on different points. These points build the *circle of confusion* and are recognizable in an image as blur, if the circle of confusion is larger than one pixel. We call everything on an image that creates a recognizable circle of confusion and becomes blurry *out of focus*. Conversely, everything else is in the *depth of field* of the camera and appears sharp in the image.

If we have a light field captured with an array of close to pinhole cameras, we can simulate cameras with apertures of nearly any size by considering several input cameras. To explain this effect we consider Figure 5.9. Again we have a light field parametrized by the uv- and st-planes and captured by a camera array that we indicate through triangles on the u-axis. The depicted scene is simple and consists of two walls at different depths. Both are colored as checkerboards, one in black and grey, the other one in green and blue squares. Now we compute the view of a new camera located in the middle of the uv plane at (u', v') in the figure drawn in red. The focal plane of the new camera should be the plane going through the black and grey wall. We parametrize the focal plane with f (horizontally) and g (vertically).

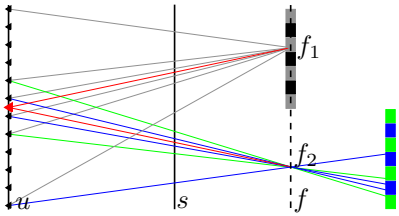


Figure 5.9: Here we show how we synthesize an image for the red camera with a large aperture focusing on the fg-plane.

As the name says the focal plane is the plane perpendicular to the pointing direction of the camera and at the depth that the camera focuses. In other words, objects on the focal plane appear sharp in the image. In our example, the black and grey wall should therefore appear sharp in the synthesized image. In order to frame the new image from several input views, we combine the rays in a way that they converge at the fg- instead of the uv-plane. To compute one pixel of the red camera we compute the ray going from (u', v') through the pixel onto the focal plane. This gives us the coordinates (f_1, g_1) on the focal plane. To determine the pixel color we compute the rays (u_i, v_i, f_1, g_1) for all neighboring cameras, where (u_i, v_i) denote the coordinates of the i-th camera. Each (u_i, v_i, f_1, g_1) ray also intersects with the st-plane and therefore is an (u, v, s, t)-ray, too. We just need to compute the st-coordinates to look up the ray's color. Again, we do this for several cameras, meaning we compute the st-coordinates for several (u_i, v_i, f_1, g_1) where the u- and v-coordinates differ, to combine these to the pixel color in the synthetic image. In case where the depicted object is in focus like on (f_1, g_1) in Figure 5.9, the result is neither impacted by the number of rays considered nor by the weighting function. All rays have the same color, in our example grey, neglecting the fact that the depicted object could be non-Lambertian. The more interesting case is when no object is on the point of the focal plane. In our example in which the wall is further away than the focal plane, we have such a point at (f_2, g_2). Luckily, the squares on

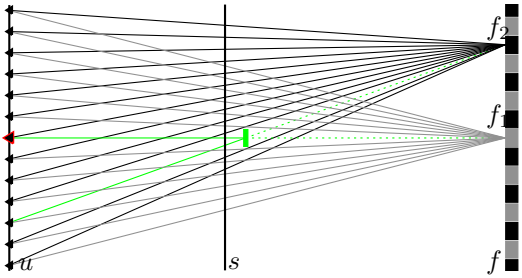


Figure 5.10: We render a new view for the red camera with focus on the gray and black plane by considering all input cameras. Then, all but one of the rays we combine for each pixel are black or gray and the green foreground object is not anymore visible in the new view.

the wall are large enough or the camera array is dense enough so that both closest input cameras see a blue spot by looking at (f_2, g_2). So by assembling the pixel that corresponds to (f_2, g_2) from the closest two cameras the pixel is still blue. This is akin to a small aperture with a large depth of field or a small circle of confusion. By increasing the aperture, which we do by taking two more input views into account, we see that the two additional rays are green. They depict already another square of the wall. When we compute a weighted sum of these four rays, the result is not a clear blue anymore and we created a blurry pixel of an out of focus object.

The same applies also to out-of-focus-objects, which are closer to the camera than the focal plane. In this case (shown in Figure 5.10), we notice an additional interesting fact. Rays which do not go through the out-of-focus-object see behind the object. We draw the object which is closer than the focal plane in green in the figure. For both example points (f_1, g_1) and (f_2, g_2) in Figure 5.10 all but one of the cameras see the background. If we now simulate the largest possible aperture, which means taking all available input views into account, then the green rays have a very low impact compared to all other rays. However, in almost all pixels that we render a ray that hits

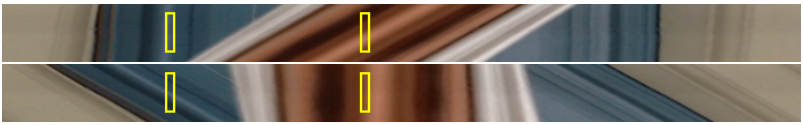


Figure 5.11: Sheared EPIs: The upper EPI we sheared such that the blue region comes into focus, i.e. is vertically aligned. In the second plot, the EPI is sheared such that the brow a area is vertically aligned.

the foreground object is involved. Thus, we end up with an image in which the foreground object is blurred over the whole image, similar to a layer of dust. The more cameras are involved and the larger the baseline of the camera array becomes, the smaller the ratio between the foreground object rays and background object rays. A larger baseline means that we simulate a larger aperture. We talk about an *infinite aperture* if the baseline is large enough for the dust to clears up to become hardly recognizable. Then, we practically made the foreground image disappear.

If we have a light field captured with a camera array and we have views sampled regularly on the uv-plane, we do not have to compute each ray separately to compute new views from the light field. We can build EPIs like the ones that we have already used to estimate the depth of objects. Assuming our EPI is a horizontal slice of our light field, then one row of the EPI corresponds to one camera. Interpolating rays from two neighboring cameras is now the same as interpolating between the corresponding two rows of the EPI. In order to simulate a larger aperture we consider more than two rows and compute a weighted sum of them, which is mathematically the same as filter these rows vertically with the according filter kernel. If we filter an EPI like the one shown in Figure 5.6 vertically with a large filter kernel, we obtain a blurry image. We can keep objects in focus by shearing the EPI according to the disparity of the object. This means the object gets the same position in all input views and this eventually leads to vertical lines in the EPI. As an example we show the EPI of Figure 5.6 twice in Figure 5.11. Once we sheared the EPI such

Figure 5.12: On top the input EPI consisting of 36 images. The one in the middle we blurred vertically with a 7 rows wide Guassian filter ($\sigma = 2.5$, yellow box). The bottom one is filtered with a box filter, considering all rows (magenta box), what we do for infinite apertures.

that the blue parts are vertically aligned and once where the brown area is vertically aligned. When filtering the sheared EPI vertically (yellow boxes), the vertically aligned areas are not changed due to the filter, independent of the filter size. As a result the objects that are represented by these rays stay in focus. On the other hand, lines which became flatter (more horizontal) through the shearing are more impacted by the filtering because more different lines are considered in such an area. Further we can control the out-of-focus blur by changing the size of the filter kernel. We show an example in which we have aligned the EPI of a light field showing a fence according to the background in Figure 5.12. In the EPI we show on the top the wires of the fence are visible as grey diagonal lines. The middle EPI we have filtered vertically with a Gaussian filter considering 7 rows (yellow box) of the EPI. The lines representing the fence wire became blurry and become out-of-focus in the single views. The bottom EPI we filtered with a box filter of the height of the EPI, which made the wires disappear totally and we have created an "infinite" aperture view. As a final note we mention that shearing the filter kernel instead of the EPI images is mathematically the same but computationally cheaper.

The last application of light fields that we discuss is a light field display. The *autostereoscopic display* makes a whole light field visible

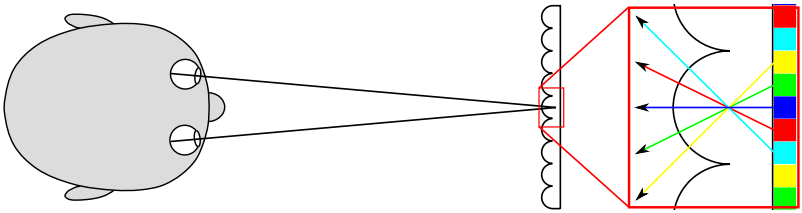


Figure 5.13: The two eyes of the viewer see the autostereoscopic display under slightly different angles and therefore receive different views emitted by the lenslets on the screen. In the example in this figure, the viewer's left eye receives the color from the red pixel and the right eye from the green pixel.

to a viewer in the sense that the viewer can independently move in the light field and always receives the correct view for both eyes. This also means that the two eyes see different views leading to a 3D perception of the displayed scene. We achieve this by inverting the process that we have previously used to capture a light field. Instead of a camera array that captures incoming light from several directions at many positions, we use a display that sends light rays from every position towards different directions. We can build such a display from a standard screen by adding a layer of lenslets on top of the screen. Each of the little lenses covers a few pixels of the screen and assembles them to one new pixel, which shows different colors depending on the viewing direction. Figure 5.13 illustrates such a display and shows how a viewer perceives the same "pixel" differently with each eye. In the 2D example of Figure 5.13 one lenslet covers five pixels and is therefore able to display five different views. We can increase the number of views by decreasing the pixels on the display or by increasing the lenslets. Like for the plenoptic camera we pay for more viewing directions with a loss of resolution.

Chapter 6

3D Conversion Using Vanishing Points and Image Warping

After the discussion of the problem statement and recently developed solutions for the task of 2D to 3D conversion in Chapter 3, we present our approach. In this chapter we introduce a paper we presented at the 3DTV-CON 2013 in Aberdeen [15].

The paper deals with single images that depict human made objects with a clear geometric structure, such as streets, bridges or buildings. Hence, none of the automatic algorithms known so far is able to compute appropriate depth maps. Therefore, we describe a technique for user assisted 2D to 3D stereo conversion that exploits the geometric structure of perspective images including vanishing points. We build on an image warping framework, as discussed in Chapter 4, and exploit constraints derived from the perspective geometry of the input to obtain a stereo image pair. In our approach a user specifies line and plane constraints, and indicates lines that intersect at vanishing points. Instead of explicitly constructing a depth map, we

warp the input image according to the user constraints to produce a stereo pair. Our approach is most suitable for scenes with large scale geometric structures such as buildings. It provides flexible user control and requires little user effort to produce visually convincing results.

6.1 Related Work

The standard industry workflow for high-quality conversion involves labor intensive manual processing including segmentation (or rotoscoping) and depth map creation [89]. Our work is related to previous academic research that strives to reduce user effort, while still providing enough flexibility to obtain convincing results. We restrict our discussion in this section to the most relevant previous work on user assisted stereo conversion, since we discussed the 2D to 3D conversion problem already in Chapter 3. External overview articles, such as the work by Smolic et al. [83], are available, too.

Harmann et al. [26] describe an early system that combines automatic depth map computation using a machine learning algorithm with user input. Several authors [25, 8, 98, 53] have proposed scribble-based interfaces that allow users to indicate the desired depth at sparse locations in video sequences. An automatic procedure then extrapolates the sparse user input to define dense per-pixel depth, and stereo views are created using depth-image-based rendering.

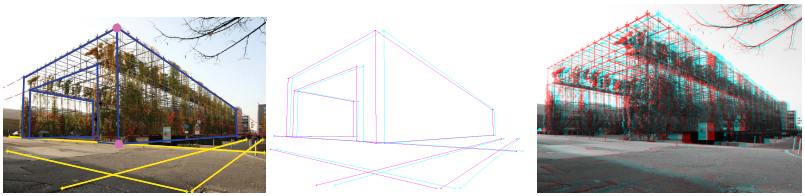
Wang et al. [91] propose a similar scribble-based user interface, which we discuss in Section 3.4.2 in depth. Further they develop a discontinuous warping technique that can create sharp depth discontinuities at object boundaries, which are discussed already in Section 4.3.2. The "depth director" system by Ward et al. [95] is based on segmentation more similar to the standard industry workflow, but it includes a variety of computer vision techniques such as automatic oversegmentation, optical flow, and structure from motion, to support user interaction. A disadvantage of scribble-based systems is that they are less suitable to generate depth maps for large scale geometric structures such as buildings, since the consistency of user scribbles

with the underlying geometry is not guaranteed.

Our approach exploits the geometric structure of perspective images including vanishing points, inspired by the seminal work by Horry et al. introducing the "tour into the picture" [35]. In contrast to this work, however, we do not explicitly construct 3D geometry, which allows us to work with more general scenes. Instead, we exploit line, plane, and vanishing points indicated by the user directly as constraints for image warping to produce a stereo pair. Our warping algorithm builds on the work by Carroll et al. [9], which we introduced in Section 4.3.1 in depth. We extend their work with constraints specifically for stereo conversion. Since we sidestep explicit depth image creation, we also avoid potential artifacts commonly associated with depth-image-based rendering.

6.2 Overview

We show an overview of our method in Figure 6.1. Given a source image, our main idea is to construct the left and right view of a stereo pair using constrained image warping. A user specifies various constraints such as straight line constraints (yellow in Figure 6.1a), which preserve linearity in the warped image, and planar region constraints (blue), which locally restrict the image warp to a homography. The user can also select sets of lines that converge in vanishing points (dotted red lines). Finally, he can place target disparity constraints at individual locations in the source image (pink). In addition to the user inputs, our system automatically enforces additional constraints specific to stereo conversion: we restrict the image warp to generate horizontal disparities, we set the target disparity of vanishing points to zero (since they are at infinity), and we enforce that the disparity along line constraints varies linearly. We feed these constraints into an optimization-based image warping algorithm to map the input view onto two new images, the left and right view respectively. Figure 6.1b shows the line and plane constraints provided by the user after warping, and Figure 6.1c shows the final stereo output. We next provide details of the image warping algorithm, and then discuss results.



(a) User input (b) Warped Constraints (c) Anaglyph Output

Figure 6.1: Overview of our method: (a) The source image with the user provided input consisting of line (yellow) and plane constraints (blue), vanishing points (intersections of dotted red lines), and disparity constraints (pink dots). (b) shows the line and plane constraints in the left and right view after warping (red-cyan anaglyph encoded). The output stereo pair (anaglyph) is shown in (c). ©Felix Frey, reproduced by permission.

6.3 Constrained Image Warping

Our constrained image warping algorithm is based on an energy minimization framework following the work by Carroll et al. [9]. We next describe our user interface, in Section 6.3.2 the mathematical formulation of the warping problem, and finally the energy minimization solver (Section 6.3.3).

6.3.1 User interface

The user interface allows a user to define constraints that describe the geometric structure of the scene. The image warper then employs the constraints to obtain the novel views of the scenes required for stereo output. The user may indicate the following constraints:

Planar Regions. Regions indicated as planar will be warped locally using a homography.

Straight Lines. The user can specify straight line segments, which will be preserved as straight during the warp. Further it is possible to mark subsegments of line constraints as inactive. This is

useful if a constrained line is partially occluded by other objects.

Vanishing Points. The user can indicate lines and edges of planar regions that are parallel in 3D. These lines define a vanishing point in the image plane. Vanishing points are fixed during image warping, since they correspond to points at infinity in 3D, and the projection of points at infinity do not change under a translation of the camera parallel to the image plane.

Line Orientation. Lines and edges of planar regions can be restricted to become vertical or horizontal after the warp.

Disparities. The user can fix a desired output disparity at any point on the image. Often it is necessary to define the disparity at only two or a few more locations. We allow users to indicate relative disparities between the fixed locations, which provides the ability to scale the disparities easily later.

6.3.2 Mathematical Formulation

We define our warp using a rectangular mesh consisting of quad faces, which is overlaid on the input image. Given the warped locations of the four vertexes of a quad, we warp the interior of the quad using bilinear interpolation, as described in Section 4.1. To compute the left and right view of the desired stereo output, we formulate an energy minimization problem that determines two deformed meshes $\mathbf{u}^l(\mathbf{x}_{i,j})$ and $\mathbf{u}^r(\mathbf{x}_{i,j})$ that best fulfill our set of constraints. Here, l and r denote the left and right view, respectively, i and j are vertex indexes, $\mathbf{x}_{i,j}$ are the locations of the undeformed mesh vertexes on the input image, and $\mathbf{u}^*(\mathbf{x}_{i,j})$, with $* \in \{l, r\}$, are the warped locations of the vertexes in the left and right output views, respectively. We also denote input coordinates by $\mathbf{x} = (x, y)$ and output coordinates in the left and right view by $\mathbf{u}^* = (u^*, v^*), * \in \{l, r\}$. Next we briefly discuss the energy terms for our constraints. In addition to the user provided constraints introduced in Section 6.3.1, we impose further constraints to ensure the output is a valid stereo pair.

Vertical Disparities

We avoid vertical disparities by penalizing differences between the v coordinates in the left and right output views, which leads to an energy term summing up over all mesh vertexes,

$$E_a = \sum_{i,j} (v^l_{i,j} - v^r_{i,j})^2. \tag{6.1}$$

User Provided Disparities

Each user specified disparity constraint is given by a location $\mathbf{x}^d = (x^d, y^d)$ and a target relative disparity Δ, where d denotes the disparity constraint. Each disparity constraint corresponds to a target location $\mathbf{u}^{l,d} = (x^d + f\Delta, y^d)$ in the left, and $\mathbf{u}^{r,d} = (x^d - f\Delta, y^d)$ in the right view, where f is a user specified global disparity scaling factor. Hence the energy term for each disparity constraint is

$$E_d = \left\|\mathbf{u}^l(\mathbf{x}^d) - \mathbf{u}^{l,d}\right\|^2 + \left\|\mathbf{u}^r(\mathbf{x}^d) - \mathbf{u}^{r,d}\right\|^2. \tag{6.2}$$

We also introduce a disparity constraint for each vanishing point, where the target disparity simply is $\Delta = 0$, that is, vanishing points remain fixed.

Note that the constrained location $\mathbf{x}^d$ is unlikely to coincide with a grid vertex. Hence we express the location as a linear combination of its surrounding quad vertexes, where we compute weights (a, b, c, d) according to Heckbert's inverse bilinear mapping [32], as we describe in Section 4.1. The corresponding output location is expressed using the same weights, as in Equation (4.1), $\mathbf{u}^*(\mathbf{x}) = a\mathbf{u}^*_{i,j} + b\mathbf{u}^*_{i+1,j} + c\mathbf{u}^*_{i+1,j+1} + d\mathbf{u}^*_{i,j+1}$.

Ratios from Vanishing Points

This and the following constraints are applied on both warps separately but on the same way. For simplicity we omit the superscripts l and r for the rest of the section.

While line constraints, as described in Equation 4.33, preserve straightness of lines, they do not penalize deformations along the line

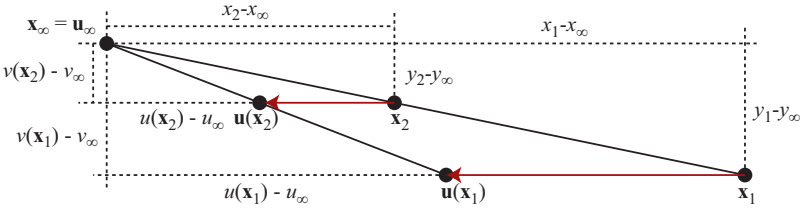


Figure 6.2: The ratios of points on a line from a vanishing point stay constant during the warp.

direction. We exploit the additional information provided by vanishing points to avoid such undesired deformations. Assume we have two points $\mathbf{x}_1 = (x_1, y_1)$, $\mathbf{x}_2 = (x_2, y_2)$ on a line with vanishing point $\mathbf{x}_\infty = (x_\infty, y_\infty)$, as shown in Figure 6.2. Let us consider the ratios $|x_1 - x_\infty|/|x_2 - x_\infty| = c_x$, and $|y_1 - y_\infty|/|y_2 - y_\infty| = c_y$. Because after the warp the line given by $\mathbf{x}_1$ and $\mathbf{x}_2$ still goes through the same vanishing point $\mathbf{u}_\infty = \mathbf{x}_\infty$, and with the intercept theorem, we can see that the ratios c_x and c_y stay constant during the warp. In our case the vanishing point never lies between $\mathbf{x}_1$ and $\mathbf{x}_2$, hence we can omit the norm. Reordering terms gives $x_1 - c_x x_2 = (1 - c_x)x_\infty$, and similarly for the y-coordinate. This leads to the energy

$$E_r = \sum_k \left(\frac{u(\mathbf{x}_k) - c_x u(\mathbf{x}_0)}{1 - c_x} - u_\infty \right)^2 + \left(\frac{v(\mathbf{x}_k) - c_y v(\mathbf{x}_0)}{1 - c_y} - v_\infty \right)^2, \tag{6.3}$$

where $\mathbf{x}_k, k \geq 0$ are locations sampled on the line segment. To sample the line regularly, we split it into intervals obtained by intersecting it with the mesh, and we use the middle point of each interval. We express these locations using bilinear interpolation from mesh vertexes as above. Note that these constraints may seem redundant with the constraint to avoid vertical disparities. With lines that are nearly horizontal, however, even small vertical disparities can lead to significant

undesired deformations. We found the ratio constraint to be necessary to avoid these in practice. Finally, observe that we divide the energy by $(1 - c_x)$ and $(1 - c_y)$. This scales the error to pixel units and makes it comparable to all the other energy terms, which measure the error in pixels, too. This is important during the optimization step, to balance the weight of the energy terms.

Additional Constraints

We implemented the remaining user constraints described in Section 6.3.1 similarly as proposed in the work by Caroll et al. [9]. For a more detailed description, we refer to Section 4.3.2. We use an energy term E_h described in Equation (4.31) to constrain the warp to a homography in planar regions. If two planar regions have a common edge, it is necessary to constrain the homographies, which gives rise to an additional term E_{hc} as formulated in Equation (4.32). As mentioned above, the straight line constraint yields the energy term E_l from Equation (4.33). This energy may also constrain the line orientation, if desired. Next there is an energy E_v that keeps lines pointing to vanishing points. Finally, there are two regularization energies as described in Section 4.1. The conformality energy E_c (Equation (4.5)) keeps the mesh rectangular, and a smoothness term E_s (Equation (4.8)) prevents abrupt changes from one mesh cell to the next.

6.3.3 Optimization

The total energy E our algorithm minimizes is a weighted sum of all the energies from the previous subsection,

$$\begin{aligned} E = {} & w_a^2 E_a + w_d^2 \sum E_d + w_r^2 \sum E_r \\ & + w_h^2 \sum E_h + w_{hc}^2 \sum E_{hc} + w_l^2 \sum E_l \\ & + w_v^2 \sum E_v + w_c^2 E_c + w_s^2 E_s, \end{aligned} \tag{6.4}$$

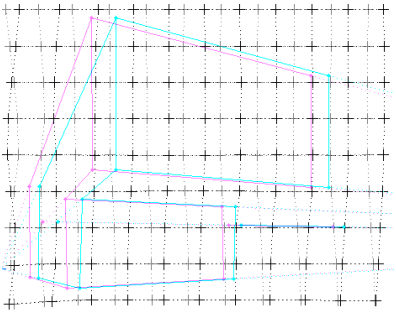
where the summations are over the number of the respective types of constraints. We also multiply each type of energy with a weight factor. The disparity constraints are the most important. Further, they act on only one grid cell, where all other constraints affect larger parts of the mesh. So, we weight this energy highest. The other user constraints are more important than the regularization terms, since the later are not meant to be hard constraints. Hence, we also weight them more heavily. Although we normalize the energy of the ratio constraint in Equation (6.3), this energy is often about ten times larger than the others. We balance this by giving ten times less weight to it. Besides, we found the best weights by experimenting and with regard to [9]. We produced all results shown here with weights $w_d = 1000$, $w_h = w_{hc} = 200$, $w_l = w_v = 100$, and $w_r = 10$ for the user given constraints, and $w_a = 20$, $w_s = 12$, and $w_c = l$ for the others.

The total energy we minimize is a least-squares problem. Because of the homography and line constraints, however, it is non-linear. We implemented a simple iterative Gauss–Newton method as described in Section 4.2.3 to solve for the minimum. We stop the minimization as soon as the total error becomes smaller than 10^{-5}. Keep in mind that we measure the energy in pixel units. In the vast majority of our experiments we reached the stopping condition after at most ten iteration steps.

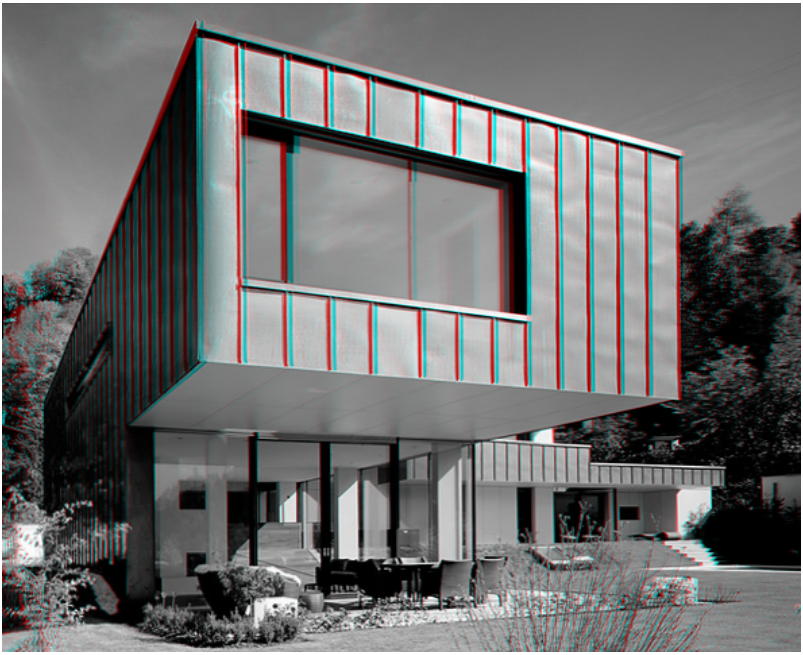
6.4 Results

We show all results in gray scale because they are more suitable for anaglyph glasses than color images. Images that need few constraints, as in Figure 6.3, take only a couple of minutes of user interaction. For more complex scenes indicating appropriate constraints may require trial and error, and our algorithm is fast enough to enable an iterative workflow. While it may be challenging for the user to set geometrically plausible disparities, our system allows a user to handle even complex scenes by specifying only a small number of disparity constraints as shown in Figure 6.4. In Figures 6.3 and 6.4 we also shift the produced images horizontally towards each other by $\Delta/2$ after warping. Hence

(a) Input with User Constraints



(b) Warped Constraints and Mesh



(c) Grayscale Output Encoded in Anaglyph

Figure 6.3: ©Angelo Kaunat, reproduced by permission.

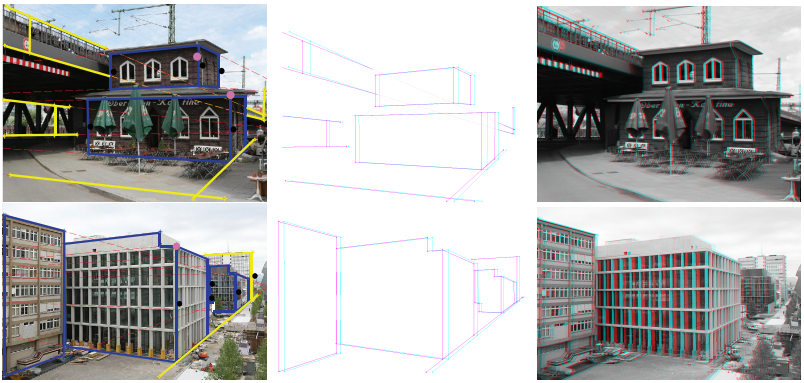


Figure 6.4: These images need many user constraints. In each we constrain five lines to be vertical (black dots). We prescribe disparities only at one (bottom) respectively two (top) points. On the bridge in the top image we use the ability to mark lines as partially hidden. This constraint guarantees that the disparity is correct along the whole bridge. ©Photos Felix Frey, reproduced by permission.

we can adjust the zero disparity plane such that the scene appears partially in front and behind the screen.

6.5 Limitations

Our algorithm is able to produce 3D images from a variety of single input images only with limited user input. In images showing many objects with round or organic shapes, however, it may be difficult to indicate the required constraints, because we rely on planes, straight lines and vanishing points. The downside of the cell-wise image warp is that it is not possible to create depth discontinuities. We illustrate this in Figure 6.5, which shows the disparity maps for three of the examples we have shown in this chapter. It is also not possible to have objects of different depths in one and the same cell. This can be seen

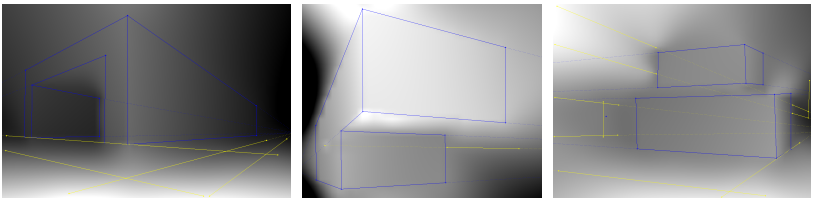


Figure 6.5: We show the implicitly computed disparity maps of the results shown in Figures 6.1, 6.3, and 6.4. We overlay the user input to indicate where the disparity discontinuities ideally would be.

in Figure 6.4 top row, where the umbrellas are warped together with the building and therefore appear at the same depth. In the future, we plan to combine our approach with scribble and segmentation based techniques to handle such cases.

Chapter 7

Hand-Held 3D Light Field Photography and Applications

Modern smartphones and tablet computers with their ever increasing computational power provide fascinating opportunities to implement computational photography applications without resorting to off-line computation. In this chapter, we describe a method for hand-held 3D light field photography, which we presented at CGI 2014 in Sydney [14]. As input we take image sequences captured with a hand-held camera along approximately linear trajectories. Capturing such data is a matter of a few seconds and does not require any extra equipment. At the core of our approach then is an efficient method to resample the input image sequence into a regularly sampled 3D light field, that is, the light field corresponds to a linear camera motion with equidistant views. This light field then opens up the possibility for a variety of further processing. First, we present a high quality algorithm for disparity estimation. Based on the disparity map, we then propose applications for digital refocusing, foreground removal, segmentation,

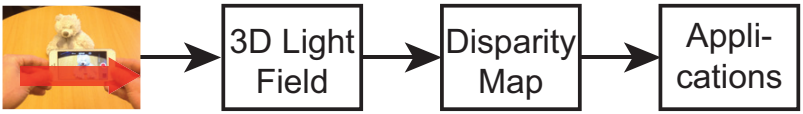


Figure 7.1: Overview of our processing pipeline.

object insertion, and multiview autostereo output.

Our approach shares similarities with recent techniques that attempt to perform multi-view 3D reconstruction [85] and 4D light field acquisition [13] on mobile devices. The main goal of multi-view reconstruction techniques is to produce full 3D models, which can then be used, for example, for 3D printing. While these techniques produce impressive results, they require several minutes of user interaction to obtain high quality reconstructions. Similarly, unstructured 4D light fields require the acquisition of many images from viewpoints distributed over a 2D domain, for example roughly on a hemisphere around an object of interest. In contrast, data capturing for our approach takes just a few seconds. The focus of our approach is not on full 3D reconstruction or image based rendering, but on providing advanced computational photography tools. In summary, we make the following contributions:

- An efficient technique for resampling sequences of images along an approximately linear camera trajectory into 3D light fields.
- A high quality disparity estimation technique based on 3D light fields.
- A technique to generate out-of-focus blur leveraging 3D light field data.
- A proof-of-concept implementation demonstrating feasibility of our approach on a mobile device.

Figure 7.1 shows an overview of our pipeline. Given an input image sequence from a hand-held camera under a roughly linear trajectory,

we first resample the data into a regularly sampled $3D$ light field (Section 7.2) and then perform disparity estimation (Section 7.3). Finally, we leverage this data for several computational photography applications (Section 7.4), including digital refocusing, foreground removal, segmentation, object insertion, and multiview autostereo output. Finally, we present results from a proof of concept application for mobile devices in Section 7.5.

7.1 Related Work

Resampling image sequences from approximately linear camera motions into 3D light fields is similar to video stabilization. Our approach is most related to the work by Liu et al. [56], which we describe in Sections 2.4 and 2.5. They proposed to use a linear analysis of feature tracks in the input video to recover a lower dimensional subspace, where the projection into the subspace is related to the camera motion. By smoothing the projection matrix they then obtain smoothed feature tracks. In contrast to their approach, we solve an optimization problem to obtain a linear camera trajectory that best approximates the input camera motion. We also resample the input images temporally to obtain a camera motion with constant speed. Similarly to their technique we render the output views using content preserving image warps [55]. As we show in Chapter 2 video stabilization can also be solved by reconstructing the 3D camera path [55], or by smoothing 2D feature trajectories under additional constraints [92]. Subspace analysis is attractive for us because it avoids the complexities and robustness issues with reconstructing the full 3D camera motion, but it provides enough information to achieve a linear camera motion at constant speed.

Our disparity estimation algorithm is inspired by the recent work of Rhemann et al. [74] and Kim et al. [44], whereas the latter represents the state-of-the art for disparity estimation from light fields. Both these works we discuss briefly in Section 5.2. Kim et al. showed that very high quality disparity estimation is possible from light fields with high spatio-angular resolution by estimating disparity scores for

single pixels. We use a similar approach to obtain initial estimates for disparity scores. Then we use an efficient edge aware filter to remove noise in our initial score volume of disparity hypotheses as proposed by Rhemann et al. While they apply the guided image filter [31] for this purpose, we are building on domain transform filtering [18], which allows us to easily include additional confidence values for the disparity hypotheses in the filtering process. We present a comparison of our approach and these techniques using standard datasets in Section 7.3, demonstrating the improved quality of our method.

Digital refocusing is one of the main applications of our framework. Ng [70] and Isaksen et al. [38] showed in their seminal work how 4D light fields can be used to refocus digital images after the fact. Unfortunately, applying the same techniques directly to 3D light fields lead to unnatural one-dimensional out-of-focus blur. In our approach, we leverage our disparity maps to combine 3D light field refocusing with an image based blur to achieve convincing results. An even simpler approach to achieve digital refocusing is to use a focus stack, which has been implemented in commercial mobile applications [72]. These techniques, however, cannot increase the defocus beyond the limits imposed by the aperture of the camera. Our approach allows for a very large synthetic aperture, and we provide additional functionality such as completely removing thin foreground objects, inspired by the work by Joshi et al. [40]. Defocus blur can also be manipulated using image processing techniques [4], but the quality of this approach is limited since it is purely image based, and it produces artifacts in particular when foreground objects are out of focus.

Beyond refocusing, our technique enables other light field processing techniques such as alpha matting [42]. We found, however, that in practice a simpler approach using edge aware filtering is more robust. Finally, the 3D light fields produced by our technique can also be used for multiview autostereo displays [54].

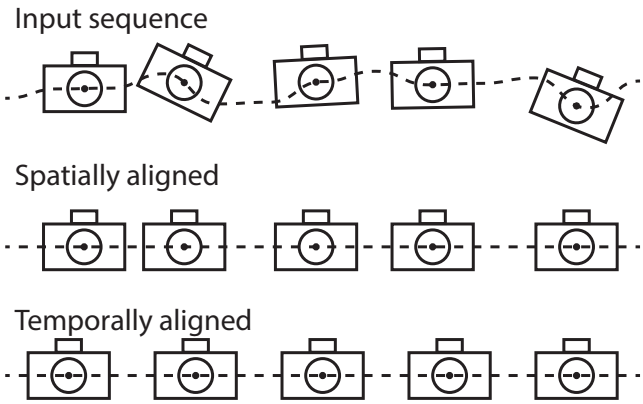


Figure 7.2: Overview of our resampling. From the input sequence (top), we first search for a horizontal camera path (middle). Then, we resample this path regularly and compute equidistant views (bottom).

7.2 Spatio-Temporal 3D Light Field Resampling

The input to our method is an image sequence from a camera sweep, similar to a sweep panorama. The sweep should be a left-to-right (or right-to-left), approximately linear camera motion without significant camera rotation. The user then picks one view as a reference image, which we use to resample the 3D light field as described below, compute a disparity map (Section 7.3), and perform our applications (Section 7.4). A camera sweep acquired using a hand-held device is unlikely to be perfectly linear, and the images usually are non-equidistant samples along the camera path. Hence we perform a linearization of the camera path in a first step (Figure 7.2, Sections 7.2.1 to 7.2.3). In a second step, we produce new views from equidistant camera positions along this linear path (Section 7.2.4).

7.2.1 Feature Trajectory Matrix

Our stabilization and resampling process is based on Liu et al.'s subspace video stabilization [56], which we describe also in Section 2.5. We begin with feature tracking and feature matching, and obtain a collection of feature trajectories $\{(x_t^i, y_t^i)\}$, where i is the feature index, and (x_t^i, y_t^i) are the coordinates of the feature on frame t. We collect the trajectories in a trajectory matrix,

$$M = \begin{bmatrix} x_1^1 & x_2^1 & \dots & x_F^1 \\ y_1^1 & y_2^1 & \dots & y_F^1 \\ & & \vdots & \\ x_1^N & x_2^N & \dots & x_F^N \\ y_1^N & y_2^N & \dots & y_F^N \end{bmatrix}, \tag{7.1}$$

where F is the number of frames of the input sequence and N the number of trajectories we found. Not all features can be tracked over the full duration of the video in general, and for missing features we set their corresponding entries in M to zero.

7.2.2 Factorization

The seminal work by Irani [37] showed that the trajectory matrix M can be approximated by a matrix with rank 9. Irani factorizes M into two matrices C and E. We discuss her method in Section 2.4 in depth. The feature coefficient matrix $C \in \mathbb{R}^{2N \times 9}$ describes the $3D$ structure of the N feature points, and the camera matrix $E \in \mathbb{R}^{9 \times F}$ represents the F camera positions and the projections of the features onto the frames. We exploit this in Section 7.2.3 where we search for a new camera matrix which describes a linear camera motion. Since CE is a full matrix, we multiply it element-wise with a binary matrix W consisting of ones where M has a non-zero entry, and zeros elsewhere. Hence, the matrix factorization we look for becomes

$$M \approx W \odot CE, \tag{7.2}$$

where $\odot$ denotes element-wise multiplication.

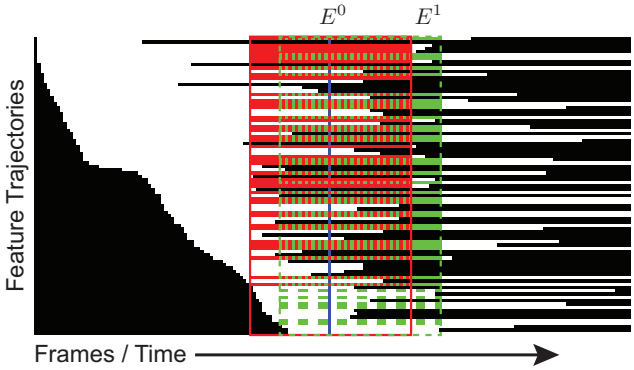


Figure 7.3: We show a part of the feature trajectory matrix. Trajectories (white) are ordered according to their first appearance. The initial factorization window is red, and a second window is green. With the C^0 matrix entries from the first window (green dotted lines) and the additional frames of these trajectories (solid part of green lines) in the next window we compute E^1. Next we compute coefficients for trajectories that span the second window, but did not span the previous one completely (dashed green lines).

We incrementally factorize M with the moving factorization method by Liu et al. [56], which we describe in Section 2.5. In our approach, we select our initial window such that the reference frame F_m is in its center. The initial window is depicted in red in Figure 7.3. We then collect all trajectories that span the whole window in a trajectory matrix M^0, which we decompose using SVD. By truncating the resulting matrices to 9 rows resp. columns and distributing the square roots of the 9 largest eigenvalues to the left and right matrices we get a camera matrix E^0 and a coefficient matrix C^0. Next, we move the window forward as depicted in green in Figure 7.3, and we search again for the trajectories that span the whole window. Since now we have some trajectories that spanned the previous window, too, we already have coefficients in C^0 for them. These cases are depicted with green dotted lines in Figure 7.3. With these coefficients we can compute the missing entries for the camera matrix E^1, which corresponds to the frames that are not covered by the previous factorization windows. The camera matrix is then complete for the current window, and we can compute the feature coefficients C^1 for the new trajectories that completely cover the current factorization window (and have not been computed before). We mark these trajectories with green dashed lines in Figure 7.3. Finally, we repeat this process forward and backward in time until all frames are processed. For a more formal description we refer to Section 2.5.2 or Liu et al. [56].

In the process above, we compute the coefficients in C for each feature with the knowledge of only one factorization window, although most feature tracks extend beyond a single window. The restriction to single windows may fail if, for example, the camera moves only very little during this window and does not constrain C enough. Therefore, we verify the validity of the coefficients of each feature by checking if the difference between the approximation using the factorization and the input feature location ever exceeds 3 pixels. If this test fails, we recompute the feature coefficients by taking into account the whole feature trajectory and test the factorization error again. In the end, we keep only trajectories for which the approximation never differs more than 3 pixels.

7.2.3 Linear Camera Motion

To construct a 3D light field we require a linear camera path and completely horizontal feature trajectories. In addition, the camera and the features should stay as close as possible to the input. Hence we seek trajectories with constant y-coordinates, and the x-coordinates along the linearized camera path should stay as close as possible to the input. Remember that we factorized the trajectory matrix into a feature coefficient matrix C and a camera matrix E. Further, we can split the coefficient matrix into submatrices C_x and C_y, which give rise to the x- and y-coordinates of the feature trajectories, respectively. With that in mind, we now search for a new matrix $\hat{E}$, such that $C_y\hat{E}$ is row-wise constant. Since our desired camera motion is linear, the columns of $\hat{E}$ representing the cameras in each frame must follow a linear model. This is, $\hat{E} = E^sT + E^b$, where E^s and E^b are both column vectors of height 9, and T is a row vector of length F. Intuitively, the vector T marks the points in time each frame was captured. Our goal is now to determine the unknowns E^s, E^b, and T.

We further reduce the degrees of freedom of the system by holding the reference frame F_m fixed. As a consequence, the y-coordinate for all trajectories is given by that frame. We then create a matrix δM containing the differences of the feature coordinates in the trajectory matrix M to the location in the reference frame F_m. Note that the column m of δM is all zero. We conclude that $T_m = 0$ and $M_m = W_m \odot CE^b$. The subscript m denotes the m-th column of M and W respectively, and the m-th entry of T. It follows that E^b is equal to the m-th column of E. Hence, our problem reduces to the minimization

$$\arg\min_{T,E^s} \|C_y(E^sT)\| + \alpha \|\delta M_x - W_x \odot C_x(E^sT)\| . \qquad (7.3)$$

The first term pushes the y-coordinates towards the ones in F_m. The second term keeps the x-coordinates where they were on the input frames and prevents the system from returning the trivial solution, and α is a factor to balance the two terms. We usually obtained best results with $\alpha = 1$.

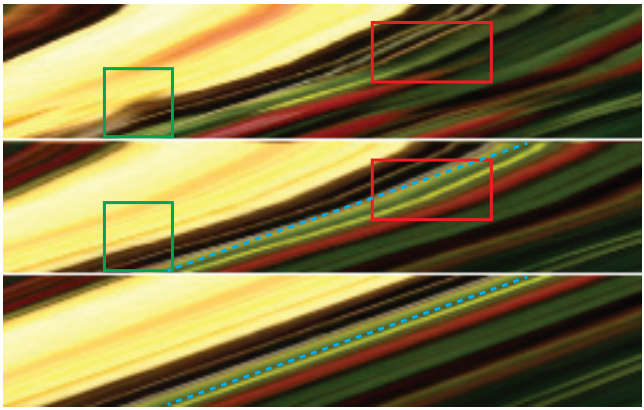


Figure 7.4: We show the EPI of the input sequence on top. Lines may become thinner or wider (green box) or may disappear (red box). In the middle is the EPI after the linearization of the camera path. The structure of the light field is now clearly visible. Still, the lines are curved as the comparison to the blue line shows. In the bottom EPI the lines became straight after temporal resampling.

7.2.4 Rendering of Output Views

We finally compute the output feature locations and render the views of the regularly sampled $3D$ light field. With T and E^s given, and $\{(x_m^i, y_m^i)\}$ the feature locations on the reference frame, the feature locations on frame j on a perfectly linearly moving camera are

$$\{(x_m^i, y_m^i) + (C_x^i E^s T_j, C_y^i E^s T_j)\}, \tag{7.4}$$

where T_j denotes the j-th entry of T. Still, it is possible that the camera changes speed along its linear trajectory. This leads to curved lines in the EPI as we show in Figure 7.4. To avoid this, we manipulate T. We set

$$\Delta t = \min(|\min(T)|, |\max(T)|)/n \tag{7.5}$$

where n is an arbitrary number of views we want to create on each side of the reference frame. For the l-th output image, with $l \in \{-n, ..., n\}$, we compute its time $t = l\Delta t$, and we use t to compute the new x-coordinates. For the y-coordinates we use the location on the reference frame F_m directly. This leads to the output feature locations $\{(x_m^i, y_m^i) + (C_x^i E^s t, 0)\}$.

We render the output view by searching for t's next smaller and larger entry in T. We warp the corresponding two input frames with the content-preserving warp from Liu et al. [55], and linearly blend the two warped frames to produce the output image. After rendering all $2n$ images our $3D$ light field is complete and the EPIs show straight lines as we show in Figure 7.4.

7.3 Disparity Map

For most of our applications we need a disparity map for our reference image. We compute this disparity map using the 3D light fields that we obtain as described in the previous section. We first construct a score volume that holds a score for a set of disparity hypotheses at each pixel, where larger scores indicate higher quality matches (Section 7.3.1). We then filter each disparity slice of the score volume using a structure preserving filter to increase the robustness of our initial score estimates (Section 7.3.2). We assign a disparity to each pixel with a winner-takes-all strategy over the filtered disparity hypotheses at each pixel. Finally, we apply a bilateral median filter to get our output disparity map.

7.3.1 Score Volume Computation

We construct our score volume using the stabilized images I_j from the previous section as input. For each disparity hypothesis from a predetermined set of hypotheses, we shift all the images horizontally with respect to the hypothesized disparity. We then compare the pixels in the shifted images with the reference image I_m and compute a score for each pixel. We adopt the similarity measurement from

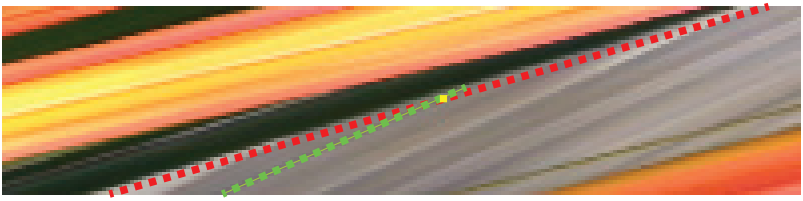


Figure 7.5: The accumulated score of the red disparity hypothesis is larger than the one of the green one because we find more pixels with non-zero scores along the red line. On the other hand, the green hypothesis has fewer but higher non-zero scores. Our normalization gives preference to the correct hypothesis in green.

Kim et al. [44] using an Epanechnikov kernel. Additionally, we also take into account the horizontal image gradients.

More precisely, we define the initial score for pixel (x, y) and disparity hypothesis d as

$$S(x,y,d) = \sum_{j \neq m} K(I_j(x_s, y) - I_m(x,y)) \cdot K(\nabla_x I_j(x_s, y) - \nabla_x I_m(x,y)), \tag{7.6}$$

where j is the index of the input image, $x_s = x + (j - m)d$ is the shifted pixel position under disparity d, and ∇_x is the horizontal image gradient. The similarity kernel K is the Epanechnikov kernel $K(z) = 1 - \|z/h\|^2$ if $\|z/h\| < 1$ and 0 otherwise. We set the threshold h to $h = 9$, where pixel values are in the range $[0, 255]$.

We observe, however, that in regions containing occlusions as in Figure 7.5 the raw score from Equation (7.6) is biased towards the foreground disparity. For the pixel marked in yellow the correct disparity corresponds to the green ray, which belongs to the background. Along this ray, however, we have fewer non-zero scores in the sum of Equation (7.6) because of the occlusion by the foreground in some of the views. Hence the sum of the scores of the disparity of the foreground, drawn in red, is higher, although the value of the individual

scores in Equation (7.6) are smaller. To avoid this effect we include a normalization step in our approach.

We first define a confidence measure $C(x, y)$, which captures at each pixel (x, y) the ratio by which the highest score outperforms the average score, that is,

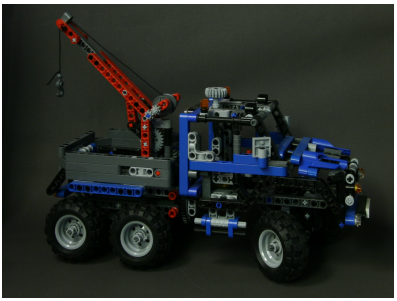
$$C(x, y) = \frac{\max_d(S(x, y, d))}{\frac{1}{D}\sum_d S(x, y, d)}. \tag{7.7}$$

This ratio indicates how unique the maximum score is with respect to the average score. In an ideal case the score is non-zero only for a single disparity hypothesis and the confidence takes on the value D, the number of disparity hypotheses. The confidence goes to 1 as the maximum score gets closer to the average.

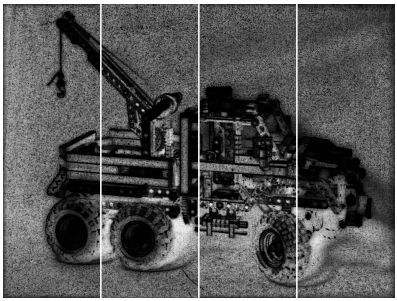
Situations as in Figure 7.5 lead to low confidence values, because the scores of the disparity hypotheses of the yellow pixel exhibit several peaks instead of a single one. Therefore, if the confidence is low we divide each score by its corresponding number of non-zero values in the sum in Equation (7.6). This favors disparity hypotheses with fewer, but higher scores, and allows us to more robustly detect the background disparity. If confidence is high there is likely a single peak in the scores and the normalization is not necessary. It may even be counterproductive, since it reduces the prominence of the peak. Hence in this case we normalize all scores for a pixel by the same factor, which is the number of non-zero values in the highest score in Equation (7.6). We obtained all our results with a threshold of $D/4$ on the confidence.

7.3.2 Score Volume Filtering

The purpose of the score volume filtering step is to reduce the noise in our initial per pixel score estimate described above. We apply an edge preserving filter to the (x, y)-slice of each disparity hypothesis similar as proposed by Rheman et al. [74]. Instead of the guided image filter, however, we use the domain transform filter (DTF) introduced by Gastal and Oliveira [18], whose computational complexity is linear

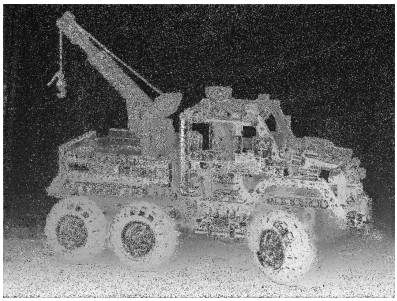


(a) Reference Image

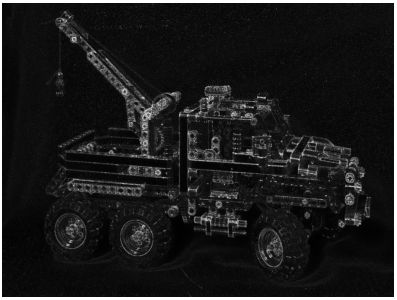


(b) Score Volume Slice

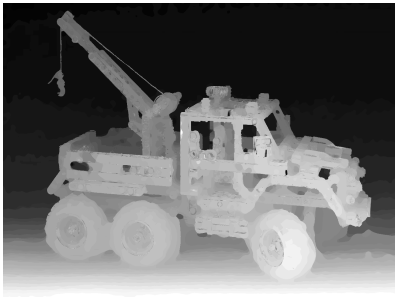
(c) Unnormalized



(d) Normalized



(e) Our confidence map



(f) Final Result

Figure 7.6: Steps of the disparity map creation: (b) parts of four slices of the score volume S for disparity hypothesis -1.5, 0, 1.5 and 3 (from left to right), (c) the disparity with maximum score, (d) the disparity with maximum but normalized scores, (e) the confidence map.

in the number of pixels to be filtered and independent of the filter support size, similar as for the guided filter. The most attractive property of the DTF for our problem is that its support adaptively shrinks or expands according to the image structure. In particular, in highly uniform areas where disparity estimation is notoriously difficult, the filter takes on a large support. In regions with rich structure, in contrast, the filter support shrinks. Intuitively, the DTF weights pairs of pixels by their distance (according to some metric) along a path connecting them in the image. This is similar to geodesic filtering, and indeed the domain transform approach can be interpreted as an iterative approximation of geodesic filtering. Here we focus on our extensions of DTF for filtering our score volumes. Please see the original publication [18] for more details.

We first give a simplified explanation of the basic DTF in 1D. Assume the input is a 1D function $I(x) : R \to R$. The DTF weight for two neighboring pixels at locations x and $x + h$ is defined as $g(|ct(x) - ct(x + h)|)$, where g is a filter kernel, and $ct : R \to R$ is the *domain transform function*, which is at the core of the approach. The main idea is to define ct in a way such that the absolute value $|ct(x) - ct(x + h)|$ is related to a l_1 distance in 2D between the two 2D points given by the pixels and their function values. This l_1 distance is defined as $\sigma_s h + |\sigma_r(I(x) - I(x + h))|$, where σ_s and σ_r are filter parameters similar to the spatial and range parameters of the bilateral filter. The key observation is that if these 2D distances are large, ct "scales up" the argument $|ct(x) - ct(x + h)|$ to the filter, leading to a quick fall-off of filter weights, and preserving the structure in the input. The opposite happens for small distances. Generalizing to color images with three r, g, b channels, one can show that the above constraints on ct lead to the definition

$$ct(u) = \int_0^u 1 + \frac{\sigma_s}{\sigma_r} \sum_{k \in r,g,b} |I'_k(x)| \, dx, \tag{7.8}$$

where I'_k is the derivative of the k-th color channel. In addition, 2D images can be filtered by iterating over several 1D passes.

In our application, we filter the disparity hypotheses scores ob-

tained in the previous section using the color image of the reference view as a "guide" to define the domain transform function, which is similar to cross-bilateral filtering. We observed, however, that we can improve the quality of our filtered output by including the confidence C, Equation (7.7), from the previous Section. The intuition for including the confidence in the DTF is that if we found a clear winner among the disparity hypotheses at a pixel, meaning we get a high confidence value, the filter does not need to extend further. On the other hand, if we have low confidence in the winning disparity hypothesis, the filter should expand until we accumulated enough evidence.

We include the confidence into the DTF as an additional channel in the guide, forcing the filter support to stop where we have enough confidence. We achieve this by plugging the logarithm of Equation (7.7) into Equation (7.8),

$$ct(u) = \int_0^u 1 + \frac{\sigma_s}{\sigma_r} \sum_{k \in \{r,g,b\}} |I'_k(x)| + \frac{\sigma_s}{\sigma_c} \log(C(x))\, dx. \qquad (7.9)$$

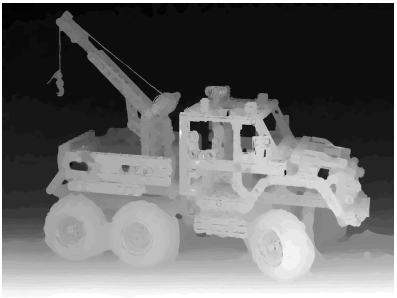
Due to the non-linearity of our confidence estimate, we use $\log(C)$ as an upper bound on the confidence of the filtered score volume that will be accumulated by the filter. We can easily show that using the logarithm guarantees that the filter support never accumulates more than the user specified confidence σ_c. We use $\sigma_r = 178.5$, set σ_s to one fifth of the image width, and $\sigma_c = \log(D)$ to produce all our results.

After cost volume filtering, we select the disparity with the highest score in a winner-takes-all manner. We finally apply a bilateral median filter to remove remaining spike noise within a 9×9 block. To compute this weighted median, we calculate its bilateral weights [88] according to the corresponding colors in the reference image. Then a histogram is created using the computed weights as accumulation factor of the neighboring disparities. The median value of this histogram is assigned to the pixel's disparity. We compare our approach to two other recent methods [93, 44] in Figure 7.7.

(a) Reference Image

(b) Wanner and Goldlücke [93]

(c) Kim et al. [44]



(d) Ours

Figure 7.7: Our result compared to other recent methods: (a) the reference view, (b) result of Wanner and Goldlücke [93], (c) result of Kim et al. [44], (d) our final disparity map.

7.4 Applications

In this section we present several applications of our reconstructed 3D light fields, most of them relying on disparity maps constructed as described above.

7.4.1 Refocusing using Synthetic Apertures

Shallow depth of field effects, as often used in professional portrait photography for example, are beyond the reach of devices like smart phones because of size restrictions on the optical design. Light fields acquired by translating a camera, however, make it possible to simulate synthetic apertures whose size is only limited by the range of camera translations. Light fields also facilitate digital refocusing after the fact, that is, changing the focal depth after image acquisition. We exploit our 3D light fields to achieve refocusing using potentially large synthetic apertures.

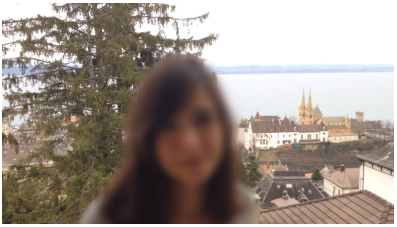
Given a 4D light field, it is straightforward to simulate a synthetic aperture by simply filtering over its two angular dimensions, where the filter represents the shape and extent of the desired aperture. The main challenge we need to overcome is that in our 3D light fields we only have one angular dimension, restricting synthetic apertures to horizontal 1D slits. We solve this problem by observing that we can model any separable 2D aperture as a superposition of vertical 1D apertures over the 1D angular domain of our 3D light fields. Hence, we use a two step procedure to obtain synthetic 2D apertures. First, for each view in our 3D light field we approximate the effect of the vertical 1D aperture. In the second step, we filter these processed views over the angular domain of the light field.

We leverage our disparity maps to compute the vertical 1D synthetic apertures using a depth-aware blur. We assume a two layer model consisting of a foreground and a background layer at each pixel, where the foreground contains all neighboring pixel closer to the camera, and the background all other pixels. We compute the colors for both layers separately, and blend them using alpha compositing. We obtain the depth-aware blur by splatting each foreground pixel to its

(a) Reference Image

(b) Vertical 1D Aperture

(c) Foreground in Focus



(d) Background in Focus

(e) Reference Image

(f) Foreground in Focus

Figure 7.8: We create our synthetic aperture in two steps. First we enlarge the aperture vertically only (b). This we do for several views along the horizontal camera path. Summing them up extends the aperture horizontally (c). To get (d) we apply the same procedure by focusing on the background. (f) shows an example where the different levels of blur are recognizable. The person in foreground appears sharp, the people behind are less blurred than the building and the trees further away. (e) is the input image of (f).

vertical neighbors, where the splat size is given by the difference of the pixel's disparity to the disparity corresponding to the desired focal distance, and we use a 1D Gaussian splat kernel. More precisely, we splat the color of pixel q to a vertical neighbor p using the Gaussian weight

$$G(p, q, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{\|p-q\|^2}{2\sigma^2}}, \tag{7.10}$$

where the variance

$$\sigma_q = \frac{a\,|d(q) - d_f| + 1}{\sqrt{2\log(255)}} \tag{7.11}$$

is defined by the difference of the disparity $d(q)$ of pixel q to the disparity d_f of the object in focus and the user given aperture size a.

We compute the foreground color $F(p)$ of a pixel p by accumulating the splat contributions of all foreground pixels q, that is, pixels with larger disparities than p,

$$F(p) = \frac{\sum_{\{q|d(q)>d(p)\}} G(p, q, \sigma_q) I(q)}{W(p)}, \tag{7.12}$$

where we normalize by the sum of the weights

$$W(p) = \sum_{\{q|d(q)>d(p)\}} G(p, q, \sigma_q). \tag{7.13}$$

Note that the normalization weight $W(p)$ can be considered as an opacity value. We similarly compute the background color using all background pixels, that is, pixels with the same or smaller disparities than p. Note that here we calculate the filter size according to the disparity of p for all background pixels. We do this because p itself belongs to the background, and it should not be splatted with colors from pixels which are behind it when p itself is in focus. Then,

$$B(p) = \frac{\sum_{\{q|d(q)\leq d(p)\}} G(q, p, \sigma_p) I(q)}{\sum_{\{q|d(q)\leq d(p)\}} G(q, p, \sigma_p)}. \tag{7.14}$$

Finally, we composite the foreground and background using alpha blending with $\alpha = \min(1, W(p))$,

$$V(p) = \alpha F(p) + (1 - \alpha)B(p), \tag{7.15}$$

where we clamp foreground coverage to one. We show an example in Figure 7.8b.

Note that to apply the depth-aware blur to each light field view, we need a disparity map for each one. Instead of recomputing disparity maps for each view, we simply propagate the disparities from the reference image by following them to the other views. Hence, we propagate the disparity d of pixel (x, y) on the reference view m to pixel $(x+d(i-m), y)$ on the i-th disparity map. For pixels that receive several disparity values we keep the largest one, since this is the one belonging to the frontmost object. On the other hand, gaps appear in background regions that were occluded in the reference view. We fill these holes with the lower disparity of its left respectively right border.

Once we computed all the vertically blurred views V_i, we shift them according to the in-focus disparity d_f and compute a weighted sum

$$I_{synthApp}(p) = \sum_i G(i, m, \sigma) V_i(p_s) \tag{7.16}$$

as the output image, where $p_s = (x_s, y)$ with $x_s = x + (i - m)d_f$. We use again the Gaussian weights $G(i, m, \sigma)$, where i is the index of the view, m is the index of the reference image and $\sigma = (a + 1)/\sqrt{2\log(255)}$.

7.4.2 Further Applications

In this section we illustrate the usefulness of our processing pipeline by discussing further computational photography applications.

Foreground Removal

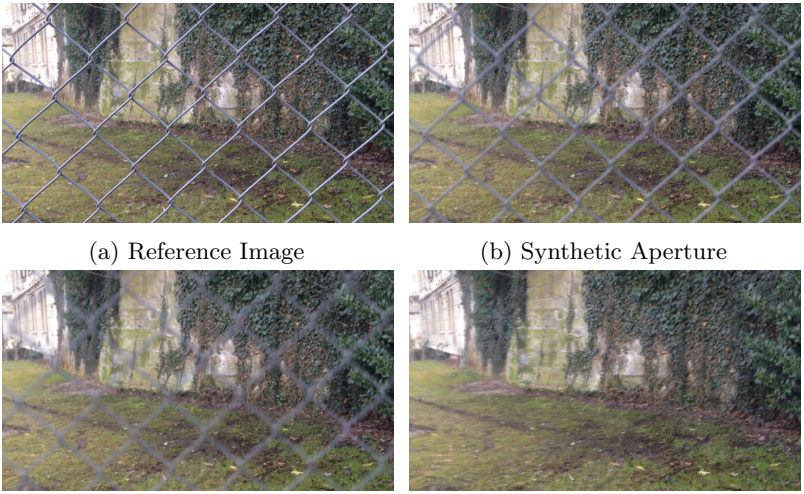
We can automatically remove thin foreground obstacles by exploiting our light field data and disparity map. This is useful to remove

unwanted objects that may spoil a shot, as illustrated in Figure 7.9. Our approach is inspired by previous work that exploits light fields to "see through" foreground objects that partially occlude the scene behind [40]. The main idea is that digitally refocusing on a background layer using a very large synthetic aperture makes the foreground almost transparent. Since we have a disparity map at our disposal in addition to the light field, we are even able to completely disregard foreground objects based on their disparity when digitally refocusing on the scene behind. We simply mask out the disparity map using a threshold given by the disparity of the obstacle. Then we refocus the light field on the background and integrate only where the mask is non-zero. We apply the same disparity propagation to the non-central light field views as in Section 7.4.1.

Segmentation and Alpha Matting

We can use our disparity map to segment foreground objects by thresholding the disparities. The user sets the threshold simply by selecting the desired object. In addition, we obtain an alpha matte by filtering the resulting binary segmentation mask with the guided image filter as proposed by He et al. [31]. The filtering step produces a "guided feathering" effect where alpha values preserve detailed image structures while smoothly blending between foreground and background. Although algorithms for alpha matting using light fields have been proposed [42], we found that these approaches are less robust and more sensitive to parameter settings and scene characteristics.

We can also use the resulting segmentation and alpha matte to generate a selective gray scale effect where the selected region stays colorful while we convert the rest of the scene to a gray scale image, as shown in Figure 7.11. Leveraging the disparity map, we can further provide functionality to insert new objects in the scene while respecting occlusions and performing alpha compositing with the foreground and background.



(a) Reference Image (b) Synthetic Aperture

(c) Large Synthetic Aperture (d) Infinite Aperture

Figure 7.9: We show the user selected reference image in (a). In (b) and (c) we applied our synthetic aperture focusing on the background with aperture size 5 and 10, respectively. In (d) we show the result of our "infinite aperture" by removing the fence.

(a) Reference Image (b) Alpha Matte (c) Text Inserted

Figure 7.10: An application of our computed alpha matte: For the reference image (a) an alpha matte is generated for the foreground (b). The object is inserted into the scene and blended with the foreground (c).

(a) Reference Image (b) Grayscale

Figure 7.11: An application of image segmentation: pixels with a disparity value below a threshold are converted to gray scale.

Multiview Autostereo Output

With the method from Section 7.2.4 we are able to render views from any point on the camera baseline. Hence it is straightforward to produce the appropriate views for autostereoscopic displays or lenticular prints. We adjust the zero-disparity plane to focus on desired scene elements by horizontally shifting the created views, where we read the required shift directly from the disparity map.

7.5 Mobile Application

To demonstrate the feasibility of a mobile app targeting advanced computational photography we implemented digital refocusing with synthetic apertures on iOS. The app lets the user record short movies and then processes the video frames as explained in Section 7.2. The user can then refocus the image as described in Section 7.4.1 using a touch gesture.

The iOS implementation shares most of the underlying source code with its desktop sibling, which keeps the porting effort at a minimum.

To improve performance on the mobile device we vectorized the math-libraries using ARM NEON, perform more complex operations asynchronously to avoid freezing the user interface, and use an OpenGL

	iPhone 5			
Movie	Preprocess	Disparity	Warp	Refocus
Fence	17.9s	44.7s	4.3s	8.8s
Rava	30.0s	41.4s	3.6s	5.7s
Yasmin	11.4s	41.4	4.1s	12.6s
	iPad Air			
	Preprocess	Disparity	Warp	Refocus
Fence	8.8s	18.3	2.2s	3.8s
Rava	14.1s	16.1	1.6s	2.5s
Yasmin	5.9s	16.5	1.8s	6.1s

Table 7.1: Results for the 2D synthetic aperture as explained in Section 7.4.1.

ES 2 based off-screen renderer to increase the performance of our image-based warper from Section 7.2.4. Last but not least, we tuned all quality settings for speed to minimize the runtime complexity when computing synthetic apertures aimed at mobile device screen resolutions. This includes the number of tracked features[1], the number of rendered views (10), and the input frame resolution (720×1280).

We benchmarked our prototype on two devices, an iPhone 5 powered by Apple's ARM-v7s A6, and an iPad Air powered by Apple's ARM-v8 64bit A7. The results are shown in Table 7.1. Apparently, preprocessing the input material is the most time consuming part, notably feature detection, whereas refocusing is relatively quick. It is thus advisable to use as few frames as possible, and then to store the preprocessed data for later reuse. This enables us to provide a similar experience as with the Lytro light field picture files.

[1] Using `cv::goodFeaturesToTrack()`

(a) The EPI we used for the disparity estimation.

(b) Reference Image

(c) Disparity Map

(d) Focus on Teddy

Figure 7.12: The branch in the top right corner of the reference image (b) moved with the breeze. Despite linearization, the EPI in (a) does not show straight lines for this area and the disparity map computation fails (c). Therefore, the branch stays sharp in the refocused image (d).

7.6 Limitations and Conclusions

Our whole pipeline relies on a faithful linearization of the input image sequence. The algorithm fails when this step does not succeed. This may happen due to the lack of feature points, or if the input video is just to shaky. In the later case, the warp may produce views with visible distortions. Such distortions may also lead to bad EPIs which may make it difficult to compute a good depth map. The same problem we face with moving content. We show an example in Figure 7.12, where the branch in the top right corner moved slightly with the breeze. This is clearly visible in the EPI, Figure 7.12a, and leads to a wrong disparity estimation, visible in the disparity map in Figure 7.12c. Further, more low-level performance optimization is needed to provide a desirable level of interactivity on current mobile devices.

However, we presented a method for hand-held 3D light field photography and described several computational photography applications enabled by our framework. The main advantage of our approach

over previous techniques for capturing light fields using hand held devices is that it requires only a simple and short user interaction, making it practical for casual users. Our work hinges on a novel technique for spatio-temporal resampling of image sequences from approximately linear camera paths into regularly sampled 3D light fields. We also developed a novel disparity estimation technique leading to state-of-the-art results on standard datasets. Finally, we introduced a digital refocusing approach using synthetic apertures that leverages our light field data and disparity maps, and several other applications. We believe our approach opens exciting avenues for further computational photography applications on mobile devices.

Chapter 8

Conclusions

We finish this thesis with a short summary of the topics we covered and highlight the contributions of our own research. We also give a brief outlook of possible future work.

Video Stabilization. We have explained the standard stabilization pipeline, which first matches feature points across frames, stabilizes them and uses these to guide the rendering of the new frames. We discussed two techniques, structure from motion and subspaces, in detail and mentioned several other methods. Further we also presented methods for stereo video stabilization and found that some of the standard stabilization methods are directly applicable to stereo videos. The biggest challenge in this area are videos with a large amount of moving content. None of the discussed methods can stabilize such a video and only one paper treats moving objects separately in the rendering step. Therefore, we see the largest space for improvement in future in this area.

2D to 3D Conversion. We have learnt why humans perceive their environment in 3D and how we can display images in a way that the viewer perceives depth of a depicted scene. Despite the rendering of

the stereo views the most difficult problem is to find out the depth in the scene of a monocular image. We have presented techniques that solve this problem through user input, but also some that use machine learning methods. Mainly in the latter case we think we are just at the beginning of the development and we will see that future work will go far beyond what machine learning algorithms can do nowadays.

Image Warping. We have presented the standard warping technique that bases on rectangular grid cells and discussed many different applications of this technique. At the end of the chapter we also discussed a method that generalizes warping techniques. This generalization in combination with the versatility of image warping leaves no doubt that we will see more applications to photos and videos.

Light Fields. We have covered the basics of light fields, their acquisition, representation and also a method to display a light field. Further we presented applications and methods that compute new views from a light field and how we can estimate the depth of objects from a light field. Since light field photography just entered the consumer market, we expect progress in the whole area of light field processing, even though the currently most vivid area is the depth estimation.

3D Conversion Using Vanishing Points and Image Warping

Conclusions. We have presented a warping-based method that is able to convert mono images into geometrically correct stereo images. It heavily relies on a strong and simple geometry in the scene and on user input. Nevertheless the user input is restricted to the provision of only some planar areas, faces and vanishing points and thus can be done in only a few minutes. The limitations of the method are twofold. First, the warp that we use is not able to produce depth discontinuities. Second, on scenes consisting of round or organic shaped objects, it may be difficult to indicate planar areas and straight lines. Then it becomes impossible to find the vanishing points.

Future Work. As a future work we therefore see the use of a warping technique that can produce depth discontinuities and the use of additional tools for the user to provide the system with depth information in the absence of planar areas and straight lines. Further, it would be interesting to remove the user input totally from the system with line detection methods and the automatic computation of vanishing points.

Hand-Held 3D Light Field Photography and Applications

Video Linearization. With our video linearization we can turn a video sequence taken with a swipe of a hand-held camera into a regularly sampled 3D light field. We use the image warping technique to render the new views. This has the downside that images that build the light field, are actually slightly distorted and we can only tolerate small deviations from a horizontal camera path. Despite this, it suites our need as we have shown with several examples.

Disparity Estimation. We combined and extended two methods which estimate the disparity of rays in an EPI. We have shown the robustness of the methods with many real-live scenes, and the accuracy of our method out-performs other state-of-the art algorithms. But for our case it is a problem that methods which estimate the disparity of lines in the EPI can not handle moving content, because this gives no straight lines in the EPI. Furthermore, time-wise the disparity map estimation is the bottle neck in our pipeline and owing to this fact we are not yet able to create a pleasant user experience.

Large Aperture Simulation. Besides other applications we have presented a method to compute photos with a narrow depth of field and this method profits from the many views we have thanks to the light field. Despite the fact that our out-of-focus blur looks more realistic than many others, it is still distinguishable from the bokeh effect created with real lenses. Additionally, we are able to remove thin foreground objects completely.

Future Work. To overcome the issue about moving content or a too shaky input video, the video linearization and depth estimation steps could be combined. In an iterative approach it seems to be possible to compute disparities and fill the wholes in the stabilized frames with information from other frames at the same time. This would make warping obsolet and would lead to a higher quality light field. Further, moving content could be recognized and removed from the initial disparity estimation. Afterwards, the disparity estimation from the static part of the scene can be propagated to the moving objects.

The linearization and disparity estimation could be made faster by the use of fewer frames. Intermediate frames could then be computed directly from the EPI to still reach the same quality of the out-of-focus blur. The blur itself could be made more realistic by the use of other blur kernels.

Bibliography

[1] Sameer. Agarwal, Noah Snavely, Ian Simon, Steven M. Seitz, and Richard Szeliski. Building rome in a day. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 72–79, Sept 2009.

[2] Adrien Auclair, Nicole Vincent, and Laurent D. Cohen. Using point correspondences without projective deformation for multi-view stereo reconstruction. In *Image Processing, 2008. ICIP 2008. 15th IEEE International Conference on*, pages 193–196, Oct 2008.

[3] Shai Avidan and Ariel Shamir. Seam carving for content-aware image resizing. *ACM Trans. Graph.*, 26(3), July 2007.

[4] Soonmin Bae and Frédo Durand. Defocus magnification. *Computer Graphics Forum*, 26(3):571–579, 2007.

[5] Luca Ballan, Gabriel J. Brostow, Jens Puwein, and Marc Pollefeys. Unstructured video-based rendering: Interactive exploration of casually captured videos. *ACM Trans. Graph.*, 29(4):87:1–87:11, July 2010.

[6] Jeffrey S. Beis and David G. Lowe. Shape indexing using approximate nearest-neighbour search in high-dimensional spaces. In *Computer Vision and Pattern Recognition, 1997. Proceedings., 1997 IEEE Computer Society Conference on*, pages 1000–1006, Jun 1997.

[7] Chris Buehler, Michael Bosse, and Leonard McMillan. Non-metric image-based rendering for video stabilization. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 2, pages II–609–II–614 vol.2, 2001.

[8] Xun Cao, Zheng Li, and Qionghai Dai. Semi-automatic 2d-to-3d conversion using disparity propagation. *Broadcasting, IEEE Transactions on*, 57(2):491–499, June 2011.

[9] Robert Carroll, Aseem Agarwala, and Maneesh Agrawala. Image warps for artistic perspective manipulation. *ACM Trans. Graph.*, 29(4):127:1–127:9, July 2010.

[10] Robert Carroll, Maneesh Agrawal, and Aseem Agarwala. Optimizing content-preserving projections for wide-angle images. *ACM Trans. Graph.*, 28(3):43:1–43:9, July 2009.

[11] Gaurav Chaurasia, Sylvain Duchene, Olga Sorkine-Hornung, and George Drettakis. Depth synthesis and local warps for plausible image-based navigation. *ACM Trans. Graph.*, 32(3):30:1–30:12, July 2013.

[12] Jiawen Chen, Sylvain Paris, Jue Wang, Wojciech Matusik, Michael Cohen, and Frédo Durand. The video mesh: A data structure for image-based three-dimensional video editing. In *Computational Photography (ICCP), 2011 IEEE International Conference on*, pages 1–8, April 2011.

[13] Abe Davis, Marc Levoy, and Fredo Durand. Unstructured light fields. *Computer Graphics Forum*, 31(2pt1):305–314, 2012.

[14] Daniel Donatsch, Siavash Arjomand Bigdeli, Philippe Robert, and Matthias Zwicker. Hand-held 3d light field photography and applications. *The Visual Computer*, 30(6-8):897–907, 2014.

[15] Daniel Donatsch, Nico Färber, and Matthias Zwicker. 3d conversion using vanishing points and image warping. In *3DTV-*

Conference: The True Vision-Capture, Transmission and Dispaly of 3D Video (3DTV-CON), 2013, pages 1–4, Oct 2013.

[16] Olivier Faugeras, Thierry Viéville, Eric Theron, Jean Vuillemin, Bernard Hotz, Zhengyou Zhang, Laurent Moll, Patrice Bertin, Herve Mathieu, Pascal Fua, Gérard Berry, and Catherine Proy. Real-time correlation-based stereo : algorithm, implementations and applications. Research Report RR-2013, 1993.

[17] Daniel Frey. 3d video stabilization. Bachelor's thesis, University of Bern, 2011.

[18] Eduardo S. L. Gastal and Manuel M. Oliveira. Domain transform for edge-aware image and video processing. *ACM Trans. Graph.*, 30(4):69:1–69:12, July 2011.

[19] Stas Goferman, Lihi Zelnik-Manor, and Ayellet Tal. Context-aware saliency detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 34(10):1915–1926, Oct 2012.

[20] Bastian Goldluecke and Sven Wanner. The variational structure of disparity and regularization of 4d light fields. In *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, pages 1003–1010, June 2013.

[21] Amit Goldstein and Raanan Fattal. Video stabilization using epipolar geometry. *ACM Trans. Graph.*, 31(5):126:1–126:10, September 2012.

[22] Steven J. Gortler, Radek Grzeszczuk, Richard Szeliski, and Michael F. Cohen. The lumigraph. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '96, pages 43–54, New York, NY, USA, 1996. ACM.

[23] Marcelo Guimares Lima. Oscar rejlander - the two ways of life. `http://photographyhistory.blogspot.com/2012/02/oscar-rejlander-two-ways-of-life.html`, 2012.

[24] M. Guttmann, L. Wolf, and D. Cohen-Or. Semi-automatic stereo extraction from video footage. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 136–142, Sept 2009.

[25] Moshe Guttmann, Lior Wolf, and Daniel Cohen-Or. Semi-automatic stereo extraction from video footage. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 136–142, Sept 2009.

[26] Philip V. Harman, Julien Flack, Simon Fox, and Mark Dowley. Rapid 2d-to-3d conversion. volume 4660, pages 78–86, 2002.

[27] Chris Harris and Mike Stephens. A combined corner and edge detector. 1988.

[28] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, New York, NY, USA, 2000.

[29] James Hays and Alexei A. Efros. Scene completion using millions of photographs. *Commun. ACM*, 51(10):87–94, October 2008.

[30] Kaiming He, Huiwen Chang, and Jian Sun. Rectangling panoramic images via warping. *ACM Trans. Graph.*, 32(4):79:1–79:10, July 2013.

[31] Kaiming He, Jian Sun, and Xiaoou Tang. Guided image filtering. In *Proceedings of the 11th European Conference on Computer Vision: Part I*, ECCV'10, pages 1–14, Berlin, Heidelberg, 2010. Springer-Verlag.

[32] Paul S. Heckbert. *Fundamentals of Texture Mapping and Image Warping*. Master's thesis, 1989.

[33] C. Hernandez, G. Vogiatzis, and R. Cipolla. Probabilistic visibility for multi-view stereo. In *Computer Vision and Pattern Recognition, 2007. CVPR '07. IEEE Conference on*, pages 1–8, June 2007.

[34] Derek Hoiem, Alexei A. Efros, and Martial Hebert. Automatic photo pop-up. *ACM Trans. Graph.*, 24(3):577–584, July 2005.

[35] Youichi Horry, Ken-Ichi Anjyo, and Kiyoshi Arai. Tour into the picture: Using a spidery mesh interface to make animation from a single image. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '97, pages 225–232, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[36] Takeo Igarashi, Tomer Moscovich, and John F. Hughes. As-rigid-as-possible shape manipulation. *ACM Trans. Graph.*, 24(3):1134–1141, July 2005.

[37] Michal Irani. Multi-frame correspondence estimation using subspace constraints. *Int. J. Comput. Vision*, 48(3):173–194, July 2002.

[38] Aaron Isaksen, Leonard McMillan, and Steven J. Gortler. Dynamically reparameterized light fields. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '00, pages 297–306, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[39] Arjun Jain, Thorsten Thormählen, Hans-Peter Seidel, and Christian Theobalt. Moviereshape: Tracking and reshaping of humans in videos. *ACM Trans. Graph.*, 29(6):148:1–148:10, December 2010.

[40] Neel Joshi, Shai Avidan, Wojciech Matusik, and David Kriegman. Synthetic aperture tracking: Tracking through occlusions. In *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pages 1–8, Oct 2007.

[41] Neel Joshi, Sing Bing Kang, C. Lawrence Zitnick, and Richard Szeliski. Image deblurring using inertial measurement sensors. *ACM Trans. Graph.*, 29(4):30:1–30:9, July 2010.

[42] Neel Joshi, Wojciech Matusik, and Shai Avidan. Natural video matting using camera arrays. *ACM Trans. Graph.*, 25(3):779–786, July 2006.

[43] Peter Kaufmann, Oliver Wang, Alexander Sorkine-Hornung, Olga Sorkine-Hornung, Aljoscha Smolic, and Markus Gross. Finite element image warping. *Computer Graphics Forum*, 32(2pt1):31–39, 2013.

[44] Changil Kim, Henning Zimmer, Yael Pritch, Alexander Sorkine-Hornung, and Markus Gross. Scene reconstruction from high spatio-angular resolution light fields. *ACM Trans. Graph.*, 32(4):73:1–73:12, July 2013.

[45] Kalin Kolev, Maria Klodt, Thomas Brox, and Daniel Cremers. Continuous global optimization in multiview 3d reconstruction. *International Journal of Computer Vision*, 84(1):80–96, 2009.

[46] Johannes Kopf, Michael F. Cohen, and Richard Szeliski. First-person hyper-lapse videos. *ACM Trans. Graph.*, 33(4):78:1–78:10, July 2014.

[47] Ilya Kostrikov, Esther Horbert, and Bastian Leibe. Probabilistic labeling cost for high-accuracy multi-view reconstruction. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pages 1534–1541, June 2014.

[48] Matthias Kunter, Sebastian Knorr, Andreas Krutz, and Thomas Sikora. Unsupervised object segmentation for 2d to 3d conversion. volume 7237, pages 72371B–72371B–10, 2009.

[49] Manuel Lang, Alexander Hornung, Oliver Wang, Steven Poulakos, Aljoscha Smolic, and Markus Gross. Nonlinear disparity mapping for stereoscopic 3d. *ACM Trans. Graph.*, 29(4):75:1–75:10, July 2010.

[50] Jehee Lee and Sung Yong Shin. General construction of time-domain filters for orientation data. *Visualization and Computer Graphics, IEEE Transactions on*, 8(2):119–128, Apr 2002.

[51] Anat Levin, Dani Lischinski, and Yair Weiss. Colorization using optimization. *ACM Trans. Graph.*, 23(3):689–694, August 2004.

[52] Marc Levoy and Pat Hanrahan. Light field rendering. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '96, pages 31–42, New York, NY, USA, 1996. ACM.

[53] Miao Liao, Jizhou Gao, Ruigang Yang, and Minglun Gong. Video stereolization: Combining motion analysis with user interaction. *IEEE Transactions on Visualization and Computer Graphics*, 18(7):1079–1088, 2012.

[54] Lippmann, G. Épreuves réversibles donnant la sensation du relief. *J. Phys. Theor. Appl.*, 7(1):821–825, 1908.

[55] Feng Liu, Michael Gleicher, Hailin Jin, and Aseem Agarwala. Content-preserving warps for 3d video stabilization. *ACM Trans. Graph.*, 28(3):44:1–44:9, July 2009.

[56] Feng Liu, Michael Gleicher, Jue Wang, Hailin Jin, and Aseem Agarwala. Subspace video stabilization. *ACM Trans. Graph.*, 30(1):4:1–4:10, February 2011.

[57] Feng Liu, Yuzhen Niu, and Hailin Jin. Joint subspace stabilization for stereoscopic video. In *Computer Vision (ICCV), 2013 IEEE International Conference on*, pages 73–80, Dec 2013.

[58] Shuaicheng Liu, Lu Yuan, Ping Tan, and Jian Sun. Bundled camera paths for video stabilization. *ACM Trans. Graph.*, 32(4):78:1–78:10, July 2013.

[59] Wan-Yen Lo, Jeroen van Baar, Claude Knaus, Matthias Zwicker, and Markus Gross. Stereoscopic 3d copy & paste. *ACM Trans. Graph.*, 29(6):147:1–147:10, December 2010.

[60] H. Christopher Longuet-Higgins and Kvetoslav Prazdny. The interpretation of a moving retinal image. *Proceedings of the Royal*

Society of London. Series B. Biological Sciences, 208(1173):385–397, July 1980.

[61] David G. Lowe. Object recognition from local scale-invariant features. In *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, volume 2, pages 1150–1157 vol.2, 1999.

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

[63] Bruce D. Lucas, Takeo Kanade, et al. An iterative image registration technique with an application to stereo vision. 1981.

[64] Sheng-Jie Luo, I-Chao Shen, Bing-Yu Chen, Wen-Huang Cheng, and Yung-Yu Chuang. Perspective-aware warping for seamless stereoscopic image cloning. *ACM Trans. Graph.*, 31(6):182:1–182:8, November 2012.

[65] Q.-T. Luong and T. Viville. Canonical representations for the geometries of multiple projective views. *Computer Vision and Image Understanding*, 64(2):193 – 229, 1996.

[66] Kaj Madsen, Hans Bruun Nielsen, and Ole Tingleff. Methods for non-linear least squares problems (2nd ed.), 2004.

[67] Marco Manzi. Weiche schatten in echtzeit-applikationen. Bachelor's thesis, University of Bern, 2010.

[68] Hans Peter Moravec. *Obstacle Avoidance and Navigation in the Real World by a Seeing Robot Rover*. PhD thesis, Stanford, CA, USA, 1980. AAI8024717.

[69] Matthias Müller, Bruno Heidelberger, Matthias Teschner, and Markus Gross. Meshless deformations based on shape matching. *ACM Trans. Graph.*, 24(3):471–478, July 2005.

[70] Ren Ng. Fourier slice photography. *ACM Trans. Graph.*, 24(3):735–744, July 2005.

[71] Yuzhen Niu, Wu-Chi Feng, and Feng Liu. Enabling warping on stereoscopic images. *ACM Trans. Graph.*, 31(6):183:1–183:7, November 2012.

[72] Nokia. Refocus app. `https://refocus.nokia.com/`, February 2014.

[73] Patrick Pérez, Michel Gangnet, and Andrew Blake. Poisson image editing. *ACM Trans. Graph.*, 22(3):313–318, July 2003.

[74] Christoph Rhemann, Asmaa Hosni, Michael Bleyer, Carsten Rother, and Margrit Gelautz. Fast cost-volume filtering for visual correspondence and beyond. In *IEEE CVPR*, CVPR '11, pages 3017–3024, Washington, DC, USA, 2011. IEEE Computer Society.

[75] Sébastien Roy and Ingemar J Cox. A maximum-flow formulation of the n-camera stereo correspondence problem. In *Computer Vision, 1998. Sixth International Conference on*, pages 492–499. IEEE, 1998.

[76] Ashutosh Saxena, Sung H. Chung, and Andrew Y. Ng. Learning depth from single monocular images. In Y. Weiss, B. Schölkopf, and J.C. Platt, editors, *Advances in Neural Information Processing Systems 18*, pages 1161–1168. MIT Press, 2006.

[77] Ashutosh Saxena, Jamie Schulte, and Andrew Y. Ng. Depth estimation using monocular and stereo cues. In *Proceedings of the 20th International Joint Conference on Artifical Intelligence*, IJCAI'07, pages 2197–2203, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc.

[78] Ashutosh Saxena, Min Sun, and Andrew Y. Ng. Make3d: Learning 3d scene structure from a single still image. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 31(5):824–840, May 2009.

[79] Scott Schaefer, Travis McPhail, and Joe Warren. Image deformation using moving least squares. *ACM Trans. Graph.*, 25(3):533–540, July 2006.

[80] S.M. Seitz, B. Curless, J. Diebel, D. Scharstein, and R. Szeliski. A comparison and evaluation of multi-view stereo reconstruction algorithms. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 1, pages 519–528, June 2006.

[81] J. Shi and C. Tomasi. Good features to track. In *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR '94., 1994 IEEE Computer Society Conference on*, pages 593–600, Jun 1994.

[82] Brandon M. Smith, Li Zhang, Hailin Jin, and Aseem Agarwala. Light field video stabilization. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 341–348, Sept 2009.

[83] Aljoscha Smolic, Peter Kauff, Sebastian Knorr, Alexander Hornung, Matthias Kunter, Marcus Muller, and Manuel Lang. Three-dimensional video postproduction and processing. *Proceedings of the IEEE*, 99(4):607–625, April 2011.

[84] Noah Snavely, Steven M. Seitz, and Richard Szeliski. Photo tourism: Exploring photo collections in 3d. *ACM Trans. Graph.*, 25(3):835–846, July 2006.

[85] Petri Tanskanen, Kalin Kolev, Lorenz Meier, Federico Camposeco, Olivier Saurer, and Marc Pollefeys. Live metric 3d reconstruction on mobile phones. In *Computer Vision (ICCV), 2013 IEEE International Conference on*, pages 65–72, December 2013.

[86] C.J. Taylor. Reconstruction of articulated objects from point correspondences in a single uncalibrated image. In *Computer Vision and Pattern Recognition, 2000. Proceedings. IEEE Conference on*, volume 1, pages 677–684 vol.1, 2000.

[87] Carlo Tomasi and Takeo Kanade. *Detection and tracking of point features*. School of Computer Science, Carnegie Mellon Univ. Pittsburgh, 1991.

[88] Carlo Tomasi and R. Manduchi. Bilateral filtering for gray and color images. In *Proceedings of the Sixth International Conference on Computer Vision*, ICCV '98, pages 839–846, Jan 1998.

[89] Andrew P. Van Pernis and Matthew S. DeJohn. Dimensionalization: converting 2d films to 3d. *Proc. SPIE*, 6803:68030T–68030T–5, 2008.

[90] Rafael Grompone von Gioi, Jeremie Jakubowicz, Jean-Michel Morel, and Gregory Randall. Lsd: A fast line segment detector with a false detection control. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(4):722–732, 2010.

[91] Oliver Wang, Manuel Lang, M. Frei, Alexander Hornung, Aljoscha Smolic, and Markus Gross. Stereobrush: Interactive 2d to 3d conversion using discontinuous warps. In *Proceedings of the Eighth Eurographics Symposium on Sketch-Based Interfaces and Modeling*, SBIM '11, pages 47–54, New York, NY, USA, 2011. ACM.

[92] Yu-Shuen Wang, Feng Liu, Pu-Sheng Hsu, and Tong-Yee Lee. Spatially and temporally optimized video stabilization. *IEEE Trans. Vis. and Comp. Graph.*, 19(8):1354–1361, August 2013.

[93] Sven Wanner and Bastian Goldluecke. Globally consistent depth labeling of 4d light fields. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 41–48, June 2012.

[94] Sven Wanner, Christoph Straehle, and Bastian Goldluecke. Globally consistent multi-label assignment on the ray space of 4d light fields. In *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, pages 1011–1018, June 2013.

[95] Ben Ward, Sing Bing Kang, and Eric P. Bennett. Depth director: A system for adding depth to movies. *Computer Graphics and Applications, IEEE*, 31(1):36–48, 2011.

[96] Jian Wei, Benjamin Resch, and Hendrik Lensch. Multi-view depth map estimation with cross-view consistency. In *Proceedings of the British Machine Vision Conference*. BMVA Press, 2014.

[97] Charles Wheatstone. Contributions to the physiology of vision. part the first. on some remarkable, and hitherto unobserved, phenomena of binocular vision. *Philosophical Transactions of the Royal Society of London*, 128:371–394, 1838.

[98] Xi Yan, You Yang, Guihua Er, and Qionghai Dai. Depth map generation for 2d-to-3d conversion by limited user inputs and depth propagation. In *3DTV Conference: The True Vision - Capture, Transmission and Display of 3D Video (3DTV-CON), 2011*, pages 1–4, May 2011.

[99] Guo-Xin Zhang, Ming-Ming Cheng, Shi-Min Hu, and Ralph R. Martin. A shape-preserving approach to image resizing. *Computer Graphics Forum*, 28(7):1897–1906, 2009.

[100] Shizhe Zhou, Hongbo Fu, Ligang Liu, Daniel Cohen-Or, and Xiaoguang Han. Parametric reshaping of human bodies in images. *ACM Trans. Graph.*, 29(4):126:1–126:10, July 2010.

Declaration of consent

on the basis of Article 28 para. 2 of the RSL05 phil.-nat.

Name/First Name:

Matriculation Number:

Study program:

Bachelor ☐ Master ☐ Dissertation ☐

Title of the thesis:

Supervisor:

I declare herewith that this thesis is my own work and that I have not used any sources other than those stated. I have indicated the adoption of quotations as well as thoughts taken from other authors as such in the thesis. I am aware that the Senate pursuant to Article 36 para. 1 lit. r of the University Act of 5 September, 1996 is authorised to revoke the title awarded on the basis of this thesis. I allow herewith inspection in this thesis.

Place/Date

Signature

Curriculum Vitae

Personal Information

Name	Daniel Donatsch
Citizenship	Swiss
Date of birth	January 26, 1981
Place of birth	Männedorf, Switzerland

Education

10/2009–7/2015 *Ph.D., Computer Science*
Computer Graphics Group, University of Bern
Bern, Switzerland
Thesis : *Computational Tools for Stereo and Light Field Photography*
Adviser : Prof. Matthias Zwicker

10/2001–2/2008 *Diploma in Mathematics*
Swiss Federal Institute of Technology (ETH)
Zürich, Switzerland
Thesis : *Smallest Enclosing Balls with Additional Constraints*
Advisor : Prof. Bernd Gärtner

8/1996–2/2001 *Matura*
Mathematisch-Naturwissenschaftliches Gymnasium
Zürich, Switzerland

Professional Experience

5/2008–7/2009 Web-Application Developer
Quatico Solutions AG
Zürich, Switzerland

10/2001–5/2008 Self-Employed Web Designer