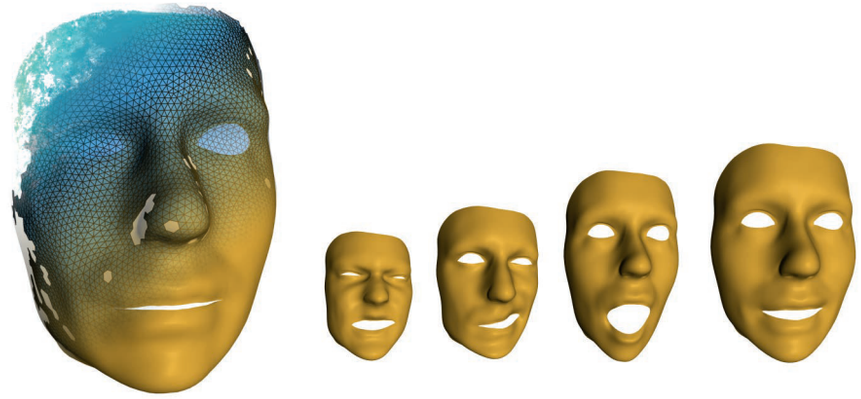
Reconstructing Dynamic Morphable Models of the Human Face

Dominik Sibbing

Selected Topics in Computer Graphics

Dominik Sibbing

Reconstructing Dynamic Morphable Models of the Human Face



Band 15

Herausgeber: Prof. Dr. Leif Kobbelt

Computer Graphics and Multimedia

Visual Computing Institute

RWTH AACHEN UNIVERSITY

Reconstructing Dynamic Morphable Models of the Human Face

Von der Fakultät für Mathematik, Informatik und Naturwissenschaften der
RWTH Aachen University zur Erlangung des akademischen Grades
eines Doktors der Naturwissenschaften genehmigte Dissertation

vorgelegt von Diplom-Informatiker
Dominik Sibbing
aus Gronau, Deutschland

Berichter: Universitätsprofessor Dr. Leif Kobbelt
Universitätsprofessor Norman I. Badler, Ph.D.

Tag der mündlichen Prüfung: 21.07.2016

Abstract

In this thesis we developed new techniques to detect, reconstruct and track human faces from pure image data. It is divided into two parts. While the first part considers static faces only, the second part deals with dynamic facial movements. For static faces we introduce a new facial feature localization method that determines the position of facial features relative to segments that were uniformly distributed in an input image. In this work we introduce and train a compact codebook that is the foundation of a voting scheme: Based on the appearance of an image segment this codebook provides offset vectors originating form the segments center and pointing towards possible feature locations. Compared to state-of-the-art methods, we show that this compact codebook has advantages regarding computational time and memory consumptions without losing accuracy. Leaving the two-dimensional image space, in the following chapter we introduce and compare two new 3D reconstruction approaches that extracts the 3D shape of a human face from multiple images. Those images were synchronously taken by a calibrated camera rig. With the aim of generating a large database of 3D facial movements, in the second part of this thesis we extend both systems to reconstruct and track human faces in 3D from videos taken by our camera rig. Both systems are completely image based and do not require any kind of facial markers. By carefully taking all requirements and characteristics into account and discussing single steps of the pipeline, we propose our facial reconstruction system that efficiently and robustly deforms a generic 3D mesh template to track a human face over time. Our tracking system preserves temporal and spatial correspondences between reconstructed faces. Due to this fact we can use the resulting database of facial movements, showing different facial expressions of a fairly large number of subjects, for further statistical analysis and to compute a generic movement model for facial actions. This movement model is independent from individual facial physiognomies. In the last chapter we introduce a new marker-less 3D face tracking

approach for 2D video streams captured by a single consumer grade camera. Our approach tracks 2D facial features and uses them to drive the evolution of our generic motion model. Here, our major contribution lies in the formulation of a smooth deformation prior which we derive from our generic motion model. We show that derived motions can be mapped back onto the individual facial shape, which leads to a reconstruction of the facial performance as seen in the video sequence. Additionally we show that it is possible to map the motion to another facial shape to drive the facial performance of a different (virtual) character. We demonstrate the effectiveness of our technique on a number of examples.

Acknowledgments

This work was made possible by the great support and contributions from numerous people. I take this opportunity to express my deepest gratitude and respect to any person who provided their help of whatever kind.

First of all, I would like to thank my adviser Prof. Leif Kobbelt, who allowed me to pursue my doctoral studies at the Computer Graphics Group of RWTH Aachen University. During the past years I profited a lot from his inspiration, professional support and his continued guidance. I thank Prof. Norman I. Badler for acting as second referee for this thesis and for giving me the opportunity to visit his lab for nine month. Besides being able to conduct many insightful discussions, where he was willing to share his enormous scientific experience, I see this stay as a valuable life experience, which I will always keep in memory. All former and current members of the Computer Graphics Group made me enjoy living and working in Aachen a lot. In particularly I would like to thank David Bommes, Jan Möbius, Henrik Zimmer, Martin Habbecke, Torsten Sattler, Hyun-Chul Choi, Robin Tomcin, Darko Pavic, Lars Krecklau. Kai Ingo Esser, Marcel Campen, Ellen Dekkers, Hans-Christian Ebke, Bastian Leibe and Ming Li for their collaborative work, being co-authors and co-operators, for their help proof-reading the papers and for all fruitful discussions and any technical help. I thank all performers and in particular the skilled actresses and actors of the *Freie Musical-Ensemble Münster e.V.*, who cooperatively helped me to create the video footage. Not only for her helpful commitment before and during the capture sessions, but especially for her endless patience and support I thank my lovely wife Viola. Finally I express my deepest gratitude to my family, my sister Carina and my parents Reiner and Gisela, who encouraged and supported me over my whole life.

Contents

1. Introduction

There is a mechanism deeply ingrained in the human visual system, that responses extremely sensitive to (human) faces. This mechanism lets us, e.g., recognize the face of a familiar person, even if that person is still far away. It also lets us instantaneously identify the emotional state of a person by shortly looking at his or her face. These abilities, which were a cornerstone in human evolution and which belong to the first skills a baby learns in life, were and still are of particular importance in any daily social interaction essential in human lives. We can only behave appropriate towards other persons (superior, spouse, friends, etc.), if we immediately recognize that person and rate his or her emotional state.

Due to this sensitivity of our visual system towards (human) faces, the creation of realistic 3D facial animations is one of the hardest challenges of today's computer graphics. But simultaneously, it is also one of the most important disciplines of computer graphics. This is because the success of substantially all movie or computer game productions raise and fall with their characters, which are expected to show more and more complex behaviors like being, e.g., cheerful, profound, kind, beastly, mean, sad or tender. In many today's movies the characters aren't real humans anymore (like, e.g, in films of the relatively young genre "animation movie", in fantasy and science-fiction films etc.), but they show the emotional states of humans and it is especially important that these emotional states appear believable to the audience. Seeing the generation of 3D facial animations from a mathematical point of view, one would think that the *emotional acceptance* follows a simple linear relation and is proportional to the degree of *realism* of the 3D model. But surprisingly, psychological studies show that this is not the case for the human visual system. The phenomena is called the *uncanny valley* [Mor70] and is sketched in Figure 1.1. Although just composed by a few circles and lines the emotional acceptance of a simple smiley is surprisingly high. For 2D cartoonish drawings the acceptance

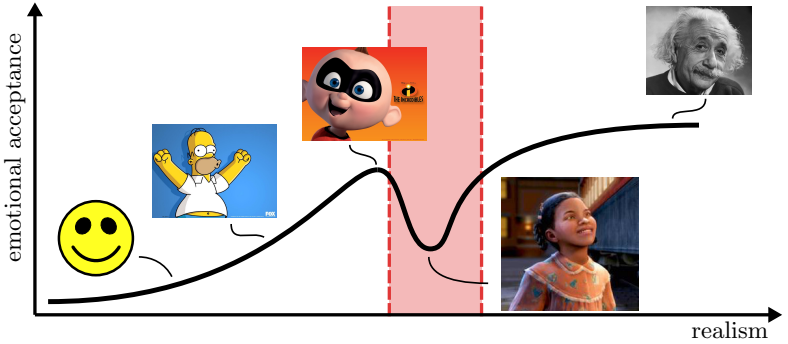


Figure 1.1.: Illustration of the uncanny valley. There is a level of realism (red area), where the emotional acceptance actually drops and where the human visual system perceives the happy face of the girl as unnatural. Images ©: Twentieth Centrury Fox Film Corporation, Pixar Animation Studios/Disney Company, Castle Rock Entertainment.

is even higher and increases further if we add more details as it was done for the baby Jack-Jack from the Pixar movie "The Incredibles". But when adding more realistic details, something strange happens as it can be seen in the face of a little girl from the movie "The Polar Express". Instead of an increased emotional credibility the character becomes scary to us, which results in an actually drop of the emotional acceptance. We can overcome this *uncanny valley*, if more realistic details are added. This highest amount of realism is, e.g., achieved by showing a photo of a real person.

To avoid this problematic zone, todays animation systems clearly define for which purpose they are designed and stay rather in the region left (cartoonish animations) or in the region right (realistic facial animations) of the *uncanny valley*. While the acceptance of cartoonish animations are mostly dependent on the artistic skills of the animations experts and designers, realistic animations require well developed technical equipment and algorithms to capture realistic facial movements. Those capture systems all work with the same principle. An actor performs in front of some scanning device which (partially) captures and reconstructs the geometry of his face and his facial movements in 3D space. Then the result can be used as it is (e.g. for a computer game where the

actor just needs to be digitized) or it is used to drive the facial performance of another character, probably having a complete different facial shape than the actor himself (e.g. the face of a dragon in a fantasy movie).

Systems that capture the facial performance of an actor, can roughly be divided into marker-based and marker-less systems. When marker-based systems are employed, the face of the actor is covered with several points, which are easily detectable by the scanning device. Then, the facial performance of the complete surface of the actor is approximated by the reconstructed 3D trajectories of these marker points or *landmarks*. Such trajectories can then be used to deform a dense reconstruction of a facial surface and thereby create an animation. The biggest problem with such systems is, that there are no measurements for the regions between the landmarks. Often algorithms come into play, which simulate the physical properties of human skin, such that the regions between the landmarks move in a plausible way. Although these algorithms perform quite well, they are often doomed to trick the human visual system, which perceives the constructed movements as unnatural.

Marker-less systems try to overcome these problems and densely reconstruct the shape and the facial movements of an actor. Thereby they have the potential to reconstruct fine details like wrinkles or little beauty marks, which makes the resulting, animated 3D model much more realistic and vivid.

In this thesis we focus on a marker-less animation system, whose scanning device is purely image based. Therefore, our approaches are based on methods from computer vision to reconstruct and track 3D facial surfaces captured in 2D images. Although future camera systems will augment the color channels with an additional depth value, which makes the reconstruction much more easier, this thesis focuses on pure color images, since they can be taken at a considerably higher frame-rate. For facial animation this is an advantage, since many facial movements (micro expressions, saccades etc.) happen within fractions of milliseconds. Nevertheless, they are still noticed by the human visual system and, e.g., play a central role to decide whether another person is telling the truth.

Realistic facial animations are not only important for the entertainment industry. Many psychological studies, which, e.g., investigate brain disorders like schizophrenia and autism, analyze the response of a subject to a presented fa-

cial animation [WGP*09, MDR*07, RMK*14, JJC*14]. In order to decrease the bias introduced by the recurring presentation of the same actor or to objectively change the intensity of the stimulus, psychologists are interested in a dynamic morphable face model of a human face to freely adjust shape parameters (which actor performs) and deformation parameters (which expression is performed in which intensity). In this thesis we develop such a dynamic morphable face model and use it to reconstruct facial movements from videos taken by a single camera.

Thesis outline and contributions. In general, the reconstruction and tracking of 3D objects from 2D images is a hard problem arising in the field of computer vision. It requires various optimization techniques and image based vision techniques such as 3D stereo reconstruction and video tracking. Therefore, we start with an exhaustive overview of the important techniques used throughout this thesis and briefly describe their principles in a comprehensible way in Chapter 2. In Section 2.4 we make a contribution to the field of camera calibration and present a very practical approach to accurately calibrate a rig of given cameras, which is the basic requirement for each 3D stereo reconstruction. This was motivated by the fact that we needed to reconstruct different facial movements from many persons and in order to guarantee a constant accuracy, we repeated the calibration, whenever we started a new capture session. We close Chapter 2 with a brief introduction to geometric modeling, which is fundamental to our 3D tracking approaches and the construction of the dynamic face model.

Before entering the main part of this thesis in Chapter 3 we briefly discuss related work in facial performance capture. Parts of this chapter have previously appeared in [SHK09, SHK11, SK15, CSK15, SK].

The thesis is divided into two main parts. Part I considers static faces showing a neutral facial expression. In Chapter 5 we encounter the problem of initializing the shape of a generic *face template*, such that it roughly approximates a human face in a neutral pose. This initialization problem can be solved by triangulating facial features automatically detected in the input images. In Chapter 4 we first describe a new technique to automatically detect such facial features. We apply a data-driven approach which estimates the location of a facial feature by analyzing the content of a small image region. Since the ap-

proach generates a *compact codebook*, the main contributions of this work are the reduction of the memory consumption and the reduction of the time to detect facial features. We experimentally show, that this approach leads to high detection rates, even if the face is partially occluded. Parts of this chapter have previously appeared in [CSK15]. In Section 5 we describe and compare our two reconstruction systems that we used to reconstruct static neutral faces. The first system was previously presented in [SHK09, SHK11], while [SK] presents the core aspects of the second system. Therefore, parts of this chapter appear in the respective publications.

Part II deals with dynamic facial expressions. We start this part in Chapter 6 by integrating the capability to track facial shapes from videos taken by a calibrated camera rig. Parts of this chapter presenting the first prototypical system previously appeared in [SHK09, SHK11], while parts concerning the second system appear in [SK]. The main contribution of both tracking systems is a new pipeline to deform a generic face template such that temporal and spatial correspondences between reconstructions are maintained during the tracking and between individual facial shapes. By empirically comparing both systems we clearly motivate the final system design, which is capable to robustly reconstruct a large database of facial expressions from different individuals in a reasonable time. We propose a number of applications, where the resulting reconstructions can be used, but the key contribution is the generation of a large database of 3D reconstructions of facial expressions. This database maintains temporal and spatial correspondences, that can be used to construct our dynamic morphable face model in Chapter 7. The main contributions of this section is a new formulation of facial movements, that decouples the facial movements from the individual facial shapes. As we will see, we can use our new model to capture the performance of faces filmed by a single camera. Parts of Chapter 7 have previously appeared in [SK15, SK].

2. Foundations

This chapter discusses the mathematical and algorithmic background needed throughout this thesis.

Since for many proposed techniques we encounter the problem of minimizing a scalar valued function, we will first explain the basic principles of least squares optimization and the Levenberg Marquard Algorithm [NW06]. The second part of the mathematical basics are Statistical Models, which are particularly important to represent large data in a compact form. Such large data are e.g. the geometric representation of facial movements performed by different actors. In this thesis we employ the Principal Component Analysis and Gaussian Mixture Models, which are briefly discussed in Section 2.2.

The proposed facial reconstruction approaches (cf. Chapter 5 and 6) mainly depend on two components. The first component is image tracking, which is based on the Lukas Kanade Tracker [BM04] to estimate the 2D movement of single pixels from one frame to the next frame of a video stream. In Section 2.3 we will present this approach in detail and how we can reformulate the objective function to improve its efficiency. Active Appearance Models [CET98, CET01, MB04] are essential to find and track facial features (cf. Chapter 7). Due to its close relation to Lucas Kanade Tracking we additionally explain its principles in Section 2.3. The second important component of the facial reconstruction pipeline involves multi-view stereo reconstruction [HZ03], which is described in more detail in Section 2.5. To understand these algorithms it is essential to know about the fundamentals of the image forming process. Therefore we will introduce the used camera model in Section 2.4 and propose a practical and accurate method to calibrate a given set of cameras (cf. Section 2.4.3).

We conclude the theoretical foundations by presenting mesh deformation techniques and show how the deformation of one object can be mapped to another object. Later in Chapter 7 we use this technique to decouple the facial

movements from the underlying facial shape and derive a statistical model of the facial movements itself.

Notation In this thesis we denote scalars as lower-case, italic type, non-bold letters (e.g., $s, t, \alpha, \ldots$). Vectors or points in higher dimensions are denoted as lower-case, bold letters (e.g. $\mathbf{a}, \mathbf{f}, \mathbf{x}, \ldots$) and matrices are written as upper-case, regular letters as, e.g., M, T or P. If needed, we will introduce other notations in the respective parts of this thesis.

2.1. Least Squares Optimization

From our perspective we mainly use least squares optimization to find a set of parameter values $\mathbf{x} \in \mathbb{R}^n$ minimizing a scalar valued function $E(\mathbf{x}) \in \mathbb{R}$, denoted as the energy function

$$E(\mathbf{x}) = \frac{1}{2} \sum_{i=1}^{m} f_i(\mathbf{x})^2 = \frac{1}{2} \mathbf{f}(\mathbf{x})^T \mathbf{f}(\mathbf{x}).$$

The vector valued function $\mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}), \ldots, f_m(\mathbf{x}))^T \in \mathbb{R}^m$ is called the residuals, and is in our case often interpreted as a set of differences between measurements and a mathematical model. An example for $\mathbf{x}$ are the $n = 2$ parameter of a 2D line. In this case the residuals could be the differences of a set of m 2D samples and the closest points on the line represented by $\mathbf{x}$.

In general the minimum of this energy function can be found by setting the derivative $\nabla E(\mathbf{x}) = 0$, where the n dimensional vector

$$\nabla E(\mathbf{x}) = \begin{bmatrix} \frac{\partial E(\mathbf{x})}{\partial x_1} \\ \vdots \\ \frac{\partial E(\mathbf{x})}{\partial x_n} \end{bmatrix} = \begin{bmatrix} \left(\frac{\partial \mathbf{f}(\mathbf{x})}{\partial x_1}\right)^T \\ \vdots \\ \left(\frac{\partial \mathbf{f}(\mathbf{x})}{\partial x_n}\right)^T \end{bmatrix} \cdot \mathbf{f}(\mathbf{x}) = \nabla \mathbf{f}(\mathbf{x})^T \cdot \mathbf{f}(\mathbf{x})$$

is called the gradient of $E(\mathbf{x})$ and $J(\mathbf{x}) := \nabla \mathbf{f}(\mathbf{x}) \in \mathbb{R}^{m \times n}$ is the Jacobian of the residuals.

In the special case where $\mathbf{f}(\mathbf{x})$ is linear in $\mathbf{x}$, the residuals can be expressed by the linear relation $\mathbf{f}(\mathbf{x}) = A\mathbf{x} + \mathbf{b}$, where $A \in \mathbb{R}^{m \times n}$ and $\mathbf{b} \in \mathbb{R}^m$. Then the optimal parameters, which minimize the energy function, can analytically

be found by setting the gradient $\nabla E(\mathbf{x})$ to zero, leading to the linear system $A^T \cdot (A\mathbf{x} + \mathbf{b}) = 0$ or as written in a more common form, that is called the normal equation [NW06]

$$A^T A\mathbf{x} = -A^T b.$$

In the more general case, where the functions $\mathbf{f}(\mathbf{x})$ are nonlinear in $\mathbf{x}$, it is often quite involved to compute an analytical solution. Moreover, in practical applications like, e.g. multi-view stereo, it is not even possible to compute the global optimal solution. For such problems common approaches locally find some improvements $\mathbf{d} \in \mathbb{R}^n$, which are supposed to reduce the energy when added to the parameters $\mathbf{x}$, i.e. $E(\mathbf{x}+\mathbf{d}) < E(\mathbf{x})$. Computing the second order Taylor expansion of $E(\mathbf{x}+\mathbf{d})$ results in an quadratic function for the unknown improvements $\mathbf{d}$

$$E(\mathbf{x}+\mathbf{d}) = E(\mathbf{x}) + \mathbf{d}^T \nabla E(\mathbf{x}) + \frac{1}{2}\mathbf{d}^T \nabla^2 E(\mathbf{x})\mathbf{d}$$

This function then can be minimized by solving the linear system

$$\nabla^2 E(\mathbf{x})\mathbf{d} = -\nabla E(\mathbf{x}) \tag{2.1}$$

Now the only difficulty is to compute the Hessian $\nabla^2 E(\mathbf{x}) \in \mathbb{R}^{n\times n}$ at position $\mathbf{x}$. Since the gradient of $E(\mathbf{x})$ can alternatively be expressed as $\nabla E(\mathbf{x}) = \sum_{i=1}^m \nabla f_i(\mathbf{x}) \cdot f_i(\mathbf{x})$ the reapplication of the gradient operator leads to the Hessian of the energy function

$$\nabla^2 E(\mathbf{x}) = \nabla \mathbf{f}(\mathbf{x})^T \cdot \nabla \mathbf{f}(\mathbf{x}) + \sum_{i=1}^{m} f_i(\mathbf{x}) \cdot \nabla^2 f_i(\mathbf{x}) \tag{2.2}$$

Observe that both terms in this equation have the right dimension: While the first term is computed from the product of a $n \times m$ and a $m \times n$ matrix, the second term is the sum of scalar values multiplied with the Hessians of the residual values $f_i(\mathbf{x})$, which is defined as

$$\nabla^2 f_i(\mathbf{x}) = \begin{bmatrix} \frac{\partial^2 f_i(\mathbf{x})}{\partial x_1 \partial x_1} & \cdots & \frac{\partial^2 f_i(\mathbf{x})}{\partial x_1 \partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f_i(\mathbf{x})}{\partial x_n \partial x_1} & \cdots & \frac{\partial^2 f_i(\mathbf{x})}{\partial x_n \partial x_n} \end{bmatrix} \in \mathbb{R}^{n\times n}$$

Equation 2.1 leads to a simple iterative algorithm to find a minimum of the objective energy function by successively adding local improvements to the parameter values $\mathbf{x}$. The *Newton Method* starts with an initial solution $\mathbf{x}_0$ and iteratively computes $\mathbf{x}_i$ by applying the following update rule

$$\begin{aligned} \mathbf{d} &= -[\nabla^2 E(\mathbf{x}_{i-1})]^{-1} \cdot \nabla E(\mathbf{x}_{i-1}) \\ \mathbf{x}_i &= \mathbf{x}_{i-1} + \mathbf{d} \end{aligned}$$

The Hessian $\nabla^2 E(\mathbf{x})$ is invertible if $\mathbf{x}$ is close to the extremal point. Once the optimum is reached, the gradient $\nabla E(\mathbf{x})$ vanishes, and the update $\mathbf{d}$ evaluates to zero. In practice the iterations are stopped if $||\mathbf{d}||$ is smaller than a certain threshold. Geometrically we can interpret this approach as follows: We locally estimate a vector $\mathbf{d}$ pointing towards the extremal point and move along its direction to lower the energy function. Since this directional vector is only tangential to the curve of the energy function, the algorithm could oscillate around the extremal point. Such oscillations can be reduced by introducing a damping constant $\lambda \in (0, 1)$ which reduces the step-size of the update [NW06]

$$\mathbf{x}_i = \mathbf{x}_{i-1} + \lambda \cdot \mathbf{d}$$

In practice the computation of the Hessian in 2.2 is not always possible or can at least be quite involved. Assuming the second derivative of the residuals $f_i(\mathbf{x})$ is small, we can omit the second term of Equation 2.2 and approximate $\nabla^2 E(\mathbf{x})$ as

$$\nabla^2 E(\mathbf{x}) = J(\mathbf{x})^T J(\mathbf{x})$$

This approximation leads to the *Gauss-Newton Method* where the update is computed by solving the linear system

$$J(\mathbf{x}_{i-1})^T J(\mathbf{x}_{i-1}) \cdot \mathbf{d} = -J(\mathbf{x}_{i-1})^T \mathbf{f}(\mathbf{x}_{i-1}) \tag{2.3}$$

and added to the previous solution for the parameters $\mathbf{x}$. According to [NW06] the convergence of this method is similar to the *Newton Method* but can be unstable if the solution is far from the extremal point. Especially this approach has the same disadvantage as the *Newton Method* since due to the linear approximation, the energy $E(\mathbf{x} + \mathbf{d})$ is not guaranteed to be smaller than the energy $E(\mathbf{x})$. In the worst case both approaches can diverge in practical application, and the minimum is never found.

The *Levenberg Marquard Algorithm* overcomes this problem and first computes a possible improvement $\mathbf{d}$ by solving the augmented system

$$[J(\mathbf{x}_{i-1})^T J(\mathbf{x}_{i-1}) + \lambda I] \cdot \mathbf{d} = -J(\mathbf{x}_{i-1})^T \mathbf{f}(\mathbf{x}_{i-1}) \tag{2.4}$$

where $\lambda \in \mathbb{R}$ is a regularization factor and I the $n \times n$ identity matrix. Observe that if λ is close to zero, this system is equivalent to the system used in the *Gauss-Newton Method*, which quickly converges in the vicinity of the extremal point. If λ is large the influence of the identity matrix dominates the system and the update is close to the negative gradient $\mathbf{d} = -J(\mathbf{x}_{i-1})^T \mathbf{f}(\mathbf{x}_{i-1})$. This update behavior is desired where the solution is far from the extremal point and where the inverse of the Hessian $J^T J$ is unstable to compute. The *Levenberg Marquard Algorithm* smoothly adapts the regularization factor to switch between the two cases and accepts only updates which actually improve the energy function. This translates to the following update rule:

$$\mathbf{x}_i, \lambda = \begin{cases} \mathbf{x}_{i-1} + \mathbf{d}, & \lambda/10 & \text{if } E(\mathbf{x}_i + \mathbf{d}) < E(\mathbf{x}_i) \\ \mathbf{x}_{i-1}, & \lambda \cdot 10 & \text{else} \end{cases}$$

In practical scenarios a good starting value is $\lambda = 10^{-3}$, while decreasing respectively increasing λ by a factor / divisor of 10 leads to stable convergence behavior [HZ03]. The requirement that the energy function can only decrease and the fact that the system matrix is symmetric and due to the regularization term positive definite, make this approach a robust and efficient optimization method for least squares optimization. Throughout this thesis it will be the first choice method for this kind of problems.

2.2. Statistical Models

Statistical models are often used to represent large data in a compact form. In this thesis we majorly use Principal Component Analysis and Gaussian mixture models for this aim and detail its principals in this section.

2.2.1. Principal Component Analysis

The aim of the Principal Component Analysis (PCA) is to find the most meaningful basis representing an observed set of possibly noisy data. PCA analyzes

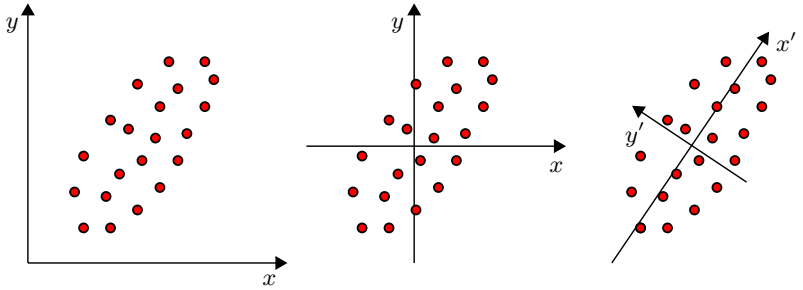


Figure 2.1.: PCA first shifts the original data (left), such that the mean of the data is the origin of a new coordinate system (middle). Then it finds a new basis (right) representing the data. In practical scenarios the data often contains some redundancy and it is possible to omit basis vectors (y' in this case), without significantly loosing expressiveness.

the covariances of this data and computes a sorted set of orthogonal axes, i.e. a new basis, where the largest variance observed in the data lies along the first axis, the second largest along the second axis and so on. At first transforming the data into this new basis, is just about finding a new representation of it. In practical scenarios it is often the case, that the data contains some redundancy, meaning that there are some axes where the variance along them is only marginal. An example of such a data set is depicted in Figure 2.1. Here we observe that the major variance of the 2D point distribution is along the $\mathbf{x}'$-axis, while there is nearly no variance along the $\mathbf{y}'$-axis (cf. right image of Figure 2.1). When projecting all points onto the new basis $[\mathbf{x}', \mathbf{y}']$ and omitting $\mathbf{y}'$, we only loose a small amount of variance but are able to represent the data in a very compact form, in this example by a single scalar value.

Lets describe this idea more precisely. The input for a PCA is a set of d-dimensional vectors or data points

$$X = [\mathbf{x}_1, \ldots, \mathbf{x}_n] \in \mathbb{R}^{d \times n}$$

representing n observations. We first center the data by subtracting the mean vector $\bar{\mathbf{x}} = \frac{1}{n} \sum_{i=1}^{n} \mathbf{x}_i$ from all data points (cf. Figure 2.1 middle)

$$\mathbf{x}_i \leftarrow \mathbf{x}_i - \bar{\mathbf{x}}$$

From now on we consider the matrix X to represent the centered data where the mean was subtracted from each column. We then assemble the covariance matrix

$$C = \frac{1}{n-1} X X^T \in \mathbb{R}^{d \times d}$$

In this form the covariance matrix can be used to compute the variance $\mathbf{d}^T C \mathbf{d}$ of the scattered data with respect to a specific direction $\mathbf{d} \in \mathbb{R}^d$, where the directional vector $\mathbf{d}$ is assumed to be normalized, i.e., $||\mathbf{d}|| = 1$. For the covariance matrix associated to the example in Figure 2.1, we would expect a high value for $\mathbf{d}^T C \mathbf{d}$, if $\mathbf{d} = \mathbf{x}'$ and a low value if $\mathbf{d} = \mathbf{y}'$. In order to identify those directions $\mathbf{d}$ which reveal the largest variances, we compute the Eigendecomposition $C = V \Sigma V^T$ [QSS07], where $V = [\mathbf{v}_1, \ldots, \mathbf{v}_d]$ is an orthonormal matrix, storing the Eigenvectors column wise and $\Sigma = \text{diag}(\lambda_1, \ldots, \lambda_d)$ is the diagonal matrix containing the Eigenvalues with $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_d$. Multiplying the covariance matrix with an Eigenvector will result in the same Eigenvector scaled by the respective Eigenvalue

$$C \mathbf{v}_i = \lambda_i \mathbf{v}_i$$

and we can easily see that the Eigenvalue λ_i represents the variance along the direction $\mathbf{v}_i$, since $\mathbf{v}_i^T C \mathbf{v_i} = \lambda_i$. The key to reduce the dimensionality is to notice that the smaller the Eigenvalue, and with it the variance along the respective Eigenvector, the less important is that Eigenvector and we can omit it in our new basis. This allows us to identify the k largest Eigenvalues and define a reduced basis as $V' = [\mathbf{v}_1, \ldots, \mathbf{v}_k]$. A point $\mathbf{x}$ in our centered data set can then be approximated by a k dimensional vector $\mathbf{y} = V'^T \mathbf{x}$ as

$$\mathbf{x} \approx V' \mathbf{y}$$

The question is how to choose the number of Eigenvectors in our basis? Common approaches select the amount of Eigenvectors such that a certain percentage (e.g. 95%) of the global variance is covered by them. Then, k is the smallest integer such that

$$0.95 \leq \frac{\sum_{i=1}^{k} \lambda_i}{\sum_{i=1}^{d} \lambda_i} \tag{2.5}$$

In practical settings the observed data can often be of high dimensions. In the case of modeling the shape of human faces one point is an individual face

represented by the concatenation of thousands of 3D vertex positions. On the other hand the amount of data points is often very small. In the former case it is experimentally quite hard to collect much more than hundred individual faces. In such cases $C \in \mathbb{R}^{d \times d}$ is a large matrix and solving the resulting Eigenproblem, where most Eigenvalues evaluate to zero, is computationally too complex in practice. To tackle this problem we start with the original solution where $\mathbf{v}$ is an Eigenvector for the covariance matrix C. Then the following equations hold

$$\begin{aligned} & C\mathbf{v} && = \lambda\mathbf{v} \\ \Leftrightarrow \quad & \tfrac{1}{n-1} X X^T \mathbf{v} && = \lambda\mathbf{v} \\ \Leftrightarrow \quad & \left(\tfrac{1}{n-1} X^T X\right) X^T \mathbf{v} && = \lambda X^T \mathbf{v} \end{aligned}$$

If we define $\tilde{C} = \frac{1}{n-1} X^T X$ and $\tilde{\mathbf{v}} = X^T \mathbf{v}$ we see that for all $\mathbf{v}$ being Eigenvectors for C, $\tilde{\mathbf{v}}$ are Eigenvectors for the much smaller system $\tilde{C} \in \mathbb{R}^{n \times n}$. This allows us to computationally improve the PCA for problems where the number of examples is much smaller than the number of dimensions: We simply compute the Eigendecomposition $\tilde{C} = \tilde{V}\tilde{\Sigma}\tilde{V}^T$ of this reduced system and deduce from its solution the Eigenvectors V of the covariance matrix C using the relation

$$\begin{aligned} & \tilde{V} && = X^T V \\ \Leftrightarrow \quad & \tilde{V}^T \tilde{V} V^T && = \tilde{V}^T X^T V V^T \\ \Leftrightarrow \quad & V && = X\tilde{V} \end{aligned}$$

In order to reduce the number of basis vectors we can again use Equation 2.5 to find a small k such that most of the global variance is covered.

2.2.2. Gaussian Mixture Models

Before we introduce Gaussian Mixture Models and describe how its model parameter can be estimated from a given set of random variables using the so called *Expectation Maximization* (EM) Algorithm [Bil98], we first go over some

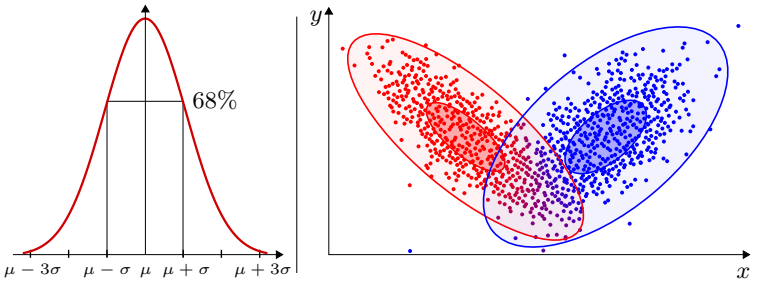


Figure 2.2.: Left: One dimensional normal distribution. Right: Gaussian Mixture Model with two clusters (red and blue). The mean of each cluster are the centers of the red and blue ellipsoids. The inner ellipsoid outlines the region within the standard deviation, while the outer ellipsoid outlines the region within three times the standard deviation. Each sample (represented as point) color codes the responsibility. Only in the overlapping region the responsibilities of the red and the blue cluster mix.

basic terminology using the example of the one dimensional normal distribution [Bis06]:

$$\mathcal{N}(x|\mu,\sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

This continuous probability density has its maximum at the *expectation* or *mean* value μ and drops to around 68% of that maximum at a distance $\pm\sigma$ to the mean, where σ is called the *standard deviation* (cf. left of Figure 2.2).

The normal distribution is often used to model the natural behavior of real valued random variables. Given a set of such variables $x_1, \dots x_N$ and a normal distribution defined by the parameters μ and σ we can compute the *likelihood function* [Bis06] to compute how well this model fits to the observed data. It is determined by multiplying the density values evaluated from the individual random variables as

$$\mathcal{L}(\mu,\sigma|x) = \prod_{i=1}^{N} \mathcal{N}(x_i|\mu,\sigma)$$

Alternatively the likelihood function can also be feed to an optimization framework which then computes the best model parameters to explain the observed

random variables x_i. In this scenario practical approaches rather optimize the logarithmic likelihood (log-likelihood) function, whose maximum equals the maximum of the likelihood function, since the logarithm is a monotonic increasing function. In case of the normal distribution the log-likelihood function is

$$\log \mathcal{L}(\mu, \sigma|x)) = \sum_{i=1}^{N} \log\left(\frac{1}{\sqrt{2\pi}\sigma}\right) - \frac{(x_i - \mu)^2}{2\sigma^2} \tag{2.6}$$

The main advantage in using the log-likelihood function is, that it is much easier to differentiate, since the products turn into simple sums. Differentiating Equation 2.6 w.r.t to μ and σ and setting the result to zero evaluates to

$$\begin{aligned} \frac{\partial}{\partial \mu} \log \mathcal{L}(\mu, \sigma|x)) &= \sum_{i=1}^{N} \frac{x_i - \mu}{\sigma^2} = 0 \\ \frac{\partial}{\partial \sigma} \log \mathcal{L}(\mu, \sigma|x)) &= \sum_{i=1}^{N} -\frac{1}{\sigma} + \frac{(x_i - \mu)^2}{\sigma^3} = 0 \end{aligned}$$

from which we can deduce the closed form solution of the optimal model parameters of the normal distribution to explain the observed random variables: Solving both equations for μ respective σ^2 leads to the well known formulas for the mean value and the *variance*, which is equal to the square of the standard deviation

$$\mu = \frac{1}{N}\sum_{i=1}^{N} x_i \qquad \text{and} \qquad \sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2$$

While the normal distribution is defined by a single exponential function, a Gaussian Mixture Model is a weighted sum of multiple normal distributions and its treatment within this framework is a bit harder. First of all random variables $\mathbf{x} \in \mathbb{R}^D$ are of higher dimensions and therefore their normal distribution is modeled by the *multivariate normal distribution*

$$\mathcal{N}(\mathbf{x}|\mu, \Sigma) = \frac{1}{\sqrt{(2\pi)^D |\Sigma|}} \cdot e^{-\frac{1}{2}(\mathbf{x}-\mu)^T \Sigma^{-1} (\mathbf{x}-\mu)} \tag{2.7}$$

where the mean $\mu \in \mathbb{R}^D$ is similarly computed as in the one dimensional case:

$$\mu = \frac{1}{N}\sum_{i=1}^{N} \mathbf{x}_i$$

In the one dimensional case the squared distances $(x-\mu)^2$ between a point x and the mean are weighted by the inverse of the variance, which means that for low variances the exponential function quickly drops to a low value when x departs from the mean. In the multivariate case the variances are distinct between different dimensions and to achieve here the same behavior of the exponential function, the squared distances $(x-\mu)^2$ are exchanged by the squared Mahalanobis distances $(\mathbf{x}-\mu)^T\Sigma^{-1}(\mathbf{x}-\mu)$ [Han05, Bis06], where the covariance matrix $\Sigma \in \mathbb{R}^{D\times D}$ is obtained by

$$\Sigma = \frac{1}{N}\sum_{i=1}^{N}(\mathbf{x}_i-\mu)(\mathbf{x}_i-\mu)^T$$

Similarly to the one dimensional case, the mean and the covariance matrix are computed by setting the differential of the log-likelihood function w.r.t. μ and Σ to zero. While the deduction of the mean exactly follows the procedure in the one dimensional case the computation of the covariance matrix is more involved, since it requires the matrix differential calculus. A complete proof for the deduction of the covariance matrix can be found in [MN99].

The second complication comes from the afore mentioned fact that a Gaussian Mixture Model is a superposition of K multivariate normal distributions $\mathcal{N}_k(\mathbf{x}) = \mathcal{N}(\mathbf{x}|\mu_\mathbf{k}, \Sigma_k)$ each having their own mean $\mu_\mathbf{k}$ and covariance matrix Σ_k. Each of the K *clusters* are weighted by parameters π_k with $0 \leq \pi_k \leq 1$ and $\sum_{k=1}^{K}\pi_k = 1$, such that the probability density for a given random variable $\mathbf{x}$ is given by

$$p(\mathbf{x}) = \sum_{k=1}^{K}\pi_k\mathcal{N}_k(\mathbf{x})$$

In the right part of Figure 2.2 we show a mixture model with two clusters (red and blue). To find the best parameters π_k, μ_k and Σ_k explaining a set of N random variables $\mathbf{x}_1, \ldots, \mathbf{x}_N$ it is again necessary to maximize the log-likelihood function

$$\log\prod_{i=1}^{N}p(\mathbf{x}_i) = \sum_{i=1}^{N}\log\left[\sum_{k=1}^{K}\pi_k\mathcal{N}_k(\mathbf{x}_i)\right] \tag{2.8}$$

When differentiating Equation 2.8 with respect to μ_k and setting the result to zero we obtain the following equation

$$0 = -\sum_{i=1}^{N} \underbrace{\frac{\pi_k \mathcal{N}_k(\mathbf{x}_i)}{\sum_{j=1}^{K} \pi_j \mathcal{N}_j(\mathbf{x}_i)}}_{:=r_{ki}} \Sigma_k(\mathbf{x}_i - \mu_k) \tag{2.9}$$

Since the normal distributions $\mathcal{N}_k(\mathbf{x}_i)$ also depend on the locations of the other means μ_k and the covariances Σ_k, solving this equation for μ_k is problematic and any direct solution rather unstable. The Expectation-Maximization solves this problem by separating the optimization into two steps. In the *expectation* step (E-step) it softly assigns each sample $\mathbf{x}_i$ to a specific cluster k by computing values r_{ki} (cf. Equation 2.9). For each sample this value describes the posterior probability that $\mathbf{x}_i$ belongs to cluster k or that the cluster k is responsible for explaining the observation $\mathbf{x}_i$. Therefore the values r_{ki}, which are color coded for each sample in Figure 2.2, are also denoted as *responsibilities*. Once these responsibilities have been determined, they appear as constants in, e.g., Equation 2.9. Then, in the *maximization* (M-step) the EM-Algorithm solves for optimal parameters π_k, μ_k and Σ_k. For example, the optimal value for the mean μ_k of cluster k is obtained by multiplying Equation 2.9 from the left side with the inverse of the covariance and rearranging the terms such that

$$\mu_k = \frac{1}{N_k} \sum_{i=1}^{N} r_{ki} \mathbf{x}_i$$

The normalization factor $N_k = \sum_{i=1}^{N} r_{ki}$ is interpreted as the effective number of samples associated to the cluster k. Similar derivations [Bis06] lead to optimal values for the remaining parameters of the Gaussian Mixture Model, which are also computed during the M-step as

$$\begin{aligned} \pi_k &= \frac{N_k}{N} \\ \Sigma_k &= \frac{1}{N_k} \sum_{i=1}^{N} r_{ki} (\mathbf{x}_i - \mu_k)(\mathbf{x}_i - \mu_k)^T \end{aligned}$$

The EM-Algorithm stops when either the parameters do not change or the log-likelihood converged to a certain value. In general the log-likelihood has many

local minima and its convergence behavior depends on the number of clusters and their initial conditions. To train a Gaussian Mixture Model in practice, the EM-Algorithm is often performed multiple times with different, randomly set initial conditions. Then the result with the highest likelihood score is accepted as the final Gaussian Mixture Model.

2.3. Image based Tracking

In this section we briefly recall the principles of the Lucas Kanade Tracking which iteratively improves an image warp to align a template image with an observed image. We also recap the principles of the inverse compositional image alignment, which is an alternative formulation of the Lucas Kanade Tracker, where the role of the template and the observed image are exchanged and which is much more efficient to compute. Active Appearance Models (AAMs) are based on this tracking approach and before embedding them in our image based face tracker (cf. Chapter 7), we briefly explain how to efficiently implement AAMs.

2.3.1. Lukas Kanade Tracking

The input for this image alignment approach is a template image T and a comparison image I. The aim is to find parameters $\mathbf{p} \in \mathbb{R}^n$ of a warping function $\mathbf{W}(\mathbf{x};\mathbf{p}) : \mathbb{R}^2 \to \mathbb{R}^2$ that maps pixel positions $\mathbf{x} \in \mathbb{R}^2$ of the template image to pixel positions in the comparison image, such that the objective function

$$E(\mathbf{p}) = \mathbf{f}(\mathbf{p})^T \mathbf{f}(\mathbf{p})$$

is minimized in the least squares sense. The residuals $\mathbf{f} = [f_1, \ldots, f_m]^T$ measure m intensity differences between pixels $\mathbf{x}$ of the template image and pixels $\mathbf{W}(\mathbf{x};\mathbf{p})$ of the comparison image:

$$f_i = I(\mathbf{W}(\mathbf{x}_i;\mathbf{p})) - T(\mathbf{x}_i) \tag{2.10}$$

The warping function $\mathbf{W}(\mathbf{x};\mathbf{p})$ is uniquely identified by its parameters $\mathbf{p} \in \mathbb{R}^n$ and differs significantly for different use cases. In the case where a 2D point needs to be tracked in two consecutive frames of a video sequence, the

warping function can simply be characterized by a 2D displacement of the point position, i.e. $\mathbf{W}(\mathbf{x};\mathbf{p}) = \mathbf{x} + [p_1, p_2]^T$. In this case the number m of measurements can be determined by the cardinality of the set of pixels within a rectangular window around $\mathbf{x}$. The surfel reconstruction approach described in Section 2.5.3 uses homographies to map pixels from the template image to a 3D surface element (surfel) and from there to the comparison image, leading to a more complicated warping function. Active Appearance Models (AAMs) use an even more complex warping function, which depends on a learned deformation model for a set of triangles characterizing the 2D shape of an object in image space. Notice that the pixel positions $\mathbf{x}_i$ are constants in the objective function and determined once before the optimization starts. Hence, we will use the abbreviatory notations $\mathbf{W}(\mathbf{x}_i, \mathbf{p}) = \mathbf{W}_i(\mathbf{p})$ and $T(\mathbf{x}_i) = T_i$ where appropriate.

Additive Image Alignment. Observe that even if $\mathbf{W}$ would be a linear function for the parameters $\mathbf{p}$, the image function $I(\mathbf{W}(\mathbf{x};\mathbf{p}))$ is highly non linear. Thus, the Additive Image Alignment [BM04] is a *Gauss-Newton Method* which iteratively computes improvements $\Delta\mathbf{p}$ until the warped comparison image is nearly identical to the template image. Following the deduction in Section 2.1 a first order Taylor expansion of the residuals $\mathbf{f}(\mathbf{p} + \Delta\mathbf{p}) \approx \mathbf{f}(\mathbf{p}) + J\Delta\mathbf{p}$ leads to the linear system

$$J^T J \cdot \Delta\mathbf{p} = -J^T \mathbf{f} \tag{2.11}$$

The Additive Image Alignment iteratively finds improvements $\Delta\mathbf{p}$ by solving this equation and adds them to the current parameter estimate, such that $\mathbf{p} \leftarrow \mathbf{p} + \Delta\mathbf{p}$. The rows of the Jacobian $J = \frac{\partial \mathbf{f}}{\partial \mathbf{p}}$ are thereby computed by consequently applying the chain rule to Equation 2.10

$$J_{i,\cdot} = \frac{\partial f_i}{\partial \mathbf{p}} = \nabla I_i \frac{\partial \mathbf{W}_i(\mathbf{p})}{\partial \mathbf{p}}$$

where the image gradient ∇I_i is evaluated at the pixel position $\mathbf{W}_i(\mathbf{p})$, and the partial derivative of the warping function is evaluated at the point $\mathbf{p}$. The Taylor expansion for the Additive Image Alignment can thus be stated as

$$f_i \approx I(\mathbf{W}_i(\mathbf{p})) + \nabla I_i \frac{\partial \mathbf{W}_i(\mathbf{p})}{\partial \mathbf{p}} \Delta\mathbf{p} - T_i \tag{2.12}$$

Since in each iteration we need to recompute a (large) set of image gradients and the partial derivatives of the warping function, the Additive Image Alignment is computationally quite involved.

Compositional Image Alignment. The Additive Image Alignment linearizes Equation 2.10 by computing the first order Taylor expansion of $f_i(\mathbf{p} + \Delta\mathbf{p})$ at the point $\mathbf{p}$ and computes an additive offset $\Delta\mathbf{p}$ for the parameters $\mathbf{p}$. Instead of this, the Compositional Image Alignment searches for an incremental warping function $\mathbf{W}(\mathbf{x}, \Delta\mathbf{p}')$ and updates the warp by composing the old warping function with that incremental warp

$$\mathbf{W}(\mathbf{x}, \mathbf{p}) \leftarrow \mathbf{W}(\mathbf{x}, \mathbf{p}) \circ \mathbf{W}(\mathbf{x}, \Delta\mathbf{p}') = \mathbf{W}(\mathbf{W}(\mathbf{x}, \Delta\mathbf{p}'), \mathbf{p})$$

The parameters $\Delta\mathbf{p}'$ of this incremental warping function are found by minimizing the linearized residuals in the least squares sense, which are computed from the first order Taylor expansion of

$$f_i = I(\mathbf{W}(\mathbf{W}_i(0 + \Delta\mathbf{p}'), \mathbf{p})) - T(\mathbf{x}_i)$$

at point $\mathbf{p} = 0$. Minimizing these residuals, again is equivalent to solving a linear system of the form of Equation 2.11. The only difference regards the Jacobian J, where we need to apply the chain rule once more due to the concatenation of the warping functions

$$J_{i,\cdot} = \frac{\partial f_i}{\partial \mathbf{p}} = \nabla I_i \frac{\partial \mathbf{W}(\mathbf{W}_i(0), \mathbf{p})}{\partial \mathbf{x}} \frac{\partial \mathbf{W}_i(0)}{\partial \mathbf{p}}$$

Under the assumption $\mathbf{W}(\mathbf{x}, 0) = \mathbf{x}$, the Jacobian can be written in a more simpler form, leading to the Taylor expansion of the residuals for the Compositional Image Alignment

$$f_i \approx I(\mathbf{W}_i(\mathbf{p})) + \nabla I_i \frac{\partial \mathbf{W}_i(\mathbf{p})}{\partial \mathbf{x}} \frac{\partial \mathbf{W}_i(0)}{\partial \mathbf{p}} \Delta\mathbf{p}' - T_i \tag{2.13}$$

The core statement of Baker and Matthews [BM04] about the Additive and the Compositional Approach is that both approaches are equivalent. This can be seen when directly comparing the linearized residuals of the additive and

the compositional approach. If $\Delta\mathbf{p}$ minimizes 2.12 and $\Delta\mathbf{p}'$ minimizes 2.13, then

$$\frac{\partial \mathbf{W}_i(\mathbf{p})}{\partial \mathbf{p}}\Delta\mathbf{p} = \frac{\partial \mathbf{W}_i(\mathbf{p})}{\partial \mathbf{x}}\frac{\partial \mathbf{W}_i(0)}{\partial \mathbf{p}}\Delta\mathbf{p}'$$

since these terms are the only differences between both formulations. Since further the update of the warping function evaluates to

$$\mathbf{W}_i(\mathbf{p}+\Delta\mathbf{p}) \approx \mathbf{W}_i(\mathbf{p})) + \frac{\partial \mathbf{W}_i(\mathbf{p})}{\partial \mathbf{p}}\Delta\mathbf{p}$$

in the additive case and to

$$\mathbf{W}(\mathbf{W}(\mathbf{x},\Delta\mathbf{p}'),\mathbf{p}) \approx \mathbf{W}_i(\mathbf{p})) + \frac{\partial \mathbf{W}_i(\mathbf{p})}{\partial \mathbf{x}}\frac{\partial \mathbf{W}_i(0)}{\partial \mathbf{p}}\Delta\mathbf{p}'$$

in the compositional case, we directly notice that both updates lead to the same change of the warping function, i.e., to the same alignment result. Thus, both formulations, the additive and the compositional approach, can be used to solve the same alignment problem.

The compositional approach has the nice advantage over the additive approach, that the expression $\frac{\partial \mathbf{W}_i(0)}{\partial \mathbf{p}}$ is a constant and needs to be evaluated only once before the optimization starts. This property is one key ingredient for the Inverse Compositional Image Alignment, which aims to reformulate the problem of image alignment, such that it can efficiently be solved.

Inverse Compositional Image Alignment. The key idea of this approach is to exchange the role of the template T and the comparison image I during the optimization. Lets consider a simple example first where the u component of pixels $\mathbf{x} = [u, v]^T$ in the template image encodes the brightness value $T(\mathbf{x}) = u$. Lets further assume that the comparison image shows the template image shifted to the right by a constant offset d, i.e. $I(x) = u - d$. Then the optimal warping function minimizing $I(\mathbf{W}(\mathbf{x})) - T(\mathbf{x})$ is $\mathbf{W}(\mathbf{x}) = [u + d, v]^T$, meaning pixels of T are shifted to the right to find the correct lookup position in I. Exchanging the role of both images means, we search for a warp $\mathbf{W}'$ in T such that $I(\mathbf{x}) - T(\mathbf{W}'(\mathbf{x}))$ is minimized. Obviously this warp is the inverse warp $\mathbf{W}'(\mathbf{x}) = \mathbf{W}^{-1}(\mathbf{x}) = [u - d, v]^T$, which shifts pixels of I to the left to find the correct lookup position in T.

As stated before, the residuals in Equation 2.10 are highly non linear and an optimal warp can only be found incrementally. Hence, Baker and Matthews [BM04] adapt the Compositional Image Alignment and exchange the role of the template and the comparison image by reformulating the residuals as

$$f_i = T(\mathbf{W}_i(\Delta\mathbf{p})) - I(\mathbf{W}_i(\mathbf{p}))$$

Similar as before the parameters $\Delta\mathbf{p}$ are found by minimizing the linearized residuals

$$f_i = T_i + J_{i,\cdot}\Delta\mathbf{p} - I(\mathbf{W}_i(\mathbf{p}))$$

which is equivalent to solving the linear system of Equation 2.11. This formulation has the big advantage that the Jacobian

$$J_{i,\cdot} = \nabla T_i \frac{\partial \mathbf{W}_i(0)}{\partial \mathbf{p}} \tag{2.14}$$

is independent from the current estimate of the warp, since T_i is evaluated at the fixed reference positions $\mathbf{x}_i$ and that the partial derivative of the warping function is constant since it is always computed at the point $\mathbf{p} = 0$. Therefore, the Jacobian and the factorization of the matrix $J^T J$ needs to be computed only once at the beginning of the optimization, which increases the performance of this approach significantly.

The approach assumes that the warping function can be inverted. If this is the case we can find the new warping function similar to the above, simple example by composing the inverse of the warp $\mathbf{W}_i(\Delta\mathbf{p})$ with the current estimate of the warp leading to the following update rule

$$\mathbf{W}_i(\mathbf{p}) \leftarrow \mathbf{W}(\mathbf{W}_i^{-1}(\Delta\mathbf{p}), \mathbf{p}) \tag{2.15}$$

To summarize the Inverse Compositional Image Alignment we precompute the Jacobian J (cf. Equation 2.14) and factorize the matrix $J^T J$. Then the algorithm iterates the following steps:

1. Compute the error image $\mathbf{f} = \mathbf{T} - \mathbf{I}(\mathbf{W}(\mathbf{p}))$
2. Solve $J^T J\Delta\mathbf{p} = -J^T\mathbf{f}$
3. Compute $\mathbf{W}_i^{-1}(\Delta\mathbf{p})$ and update the warp according to Equation 2.15

2.3.2. Active Appearance Models

Real-world instances of 3D deformable objects often are highly manifold. Their appearance depend not only on the current position and orientation, but also on their specific shape, the color of the surface and the illumination condition. Human faces are an example of such deformable objects, where their shape strongly varies for different facial expressions and their appearance depend on the look of e.g. the skin and how it is illuminated. In spite of this large variability, it is often possible to identify some extraordinary feature points, always observable in each deformed instance of the object. For human faces these features are, e.g., the corners of the eyes, the tip of the nose, the contour of the mouth and when projected to an image the contour of the face w.r.t the background.

2D Active Appearance Models (AAMs), originally introduced by Cootes et al. [CET98], represent the projected shape of an object as a 2D triangular mesh $M = (\mathbf{s}, T)$, where T is a set of triangles connecting n vertices with vertex positions

$$\mathbf{s} = (x_1, y_1, \ldots, x_n, y_n)^T$$

A triplet $t = (i, j, k) \in T$ defines a single triangle connecting the vertices $(x_i, y_i)^T$, $(x_j, y_j)^T$ and $(x_k, y_k)^T$. The topology, i.e. the set of all triangles, of the 2D mesh is fixed for a specific AAM and usually designed such that edges between vertices follow some contours, like lips, eyelids or the border of the face (cf. Figure 2.3).

Shape variations. An AAM is built from a large database of images. For each image in that training database the vertices of M have to be placed according to the features and contours of the observed object. This results in shapes $\mathbf{s}$ for each training image, which are used to fill the columns of a data matrix. The first step in building an AAM is to use PCA (cf. Section 2.2.1) to compute the average shape $\bar{\mathbf{s}}$ and an orthonormal basis $S \in \mathbb{R}^{2n \times k_s}$ with small k_s, such that a 2D shape can be represented by a linear combination

$$\mathbf{s} = \bar{\mathbf{s}} + S\mathbf{p} \tag{2.16}$$

The parameters $\mathbf{p} \in \mathbb{R}^k$ are called shape parameters of the AAM. To stabilize the computation of the PCA and in order to obtain pure deformation parame-

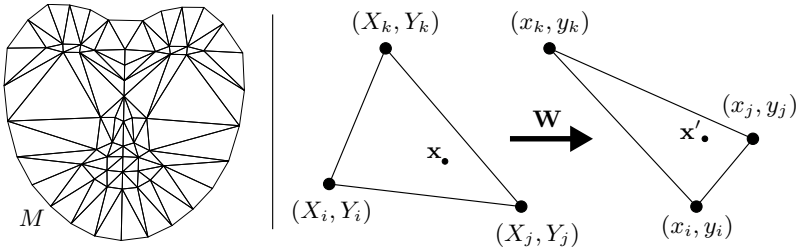


Figure 2.3.: Left: 2D triangular mesh. Right: Mapping of a point $\mathbf{x}$ from the reference to a deformed triangle.

ters, the rigid movement of the observed 2D shapes are factored out. In [MB04] Matthews and Baker suggest to use a Procrustes analysis [Goo91] to register all shapes to a common shape template.

Appearance variations. In the second step of the training phase we compute an appearance model by analyzing the content of the images in the training database. For each image the appearance of the object could be represented by concatenating the intensity values of all pixels, which lie inside the triangles of M. Unfortunately the number and ordering of pixels lying inside those shapes may strongly vary between examples and therefore this representation cannot be used for a statistical analysis like PCA. To make the appearance vectors usable for statistical analysis, the average shape

$$\bar{\mathbf{s}} = (X_1, Y_1, \ldots, X_n, Y_n)^T$$

is interpreted as a reference and rasterized. Then a rasterized point $\mathbf{x} = (x, y)^T$ lies in a specific triangle $t = (i, j, k)$ where it has the barycentric coordinates $(\alpha_{t,\mathbf{x}}, \beta_{t,\mathbf{x}}, \gamma_{t,\mathbf{x}})^T$. Assuming the positions of the reference vertices of the triangle t are $(X_i, Y_i)^T$,$(X_j, Y_j)^T$ and $(X_k, Y_k)^T$, the barycentric coordinates of $\mathbf{x}$, which are constant for each rasterized point, are given by

$$\begin{pmatrix} \alpha_{t,\mathbf{x}} \\ \beta_{t,\mathbf{x}} \\ \gamma_{t,\mathbf{x}} \end{pmatrix} = \begin{bmatrix} X_i & X_j & X_k \\ Y_i & Y_j & Y_k \\ 1 & 1 & 1 \end{bmatrix}^{-1} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

In an observed training image this triangle might have a difference shape, encoded by the vertex positions $(x_i, y_i)^T$,$(x_j, y_j)^T$ and $(x_k, y_k)^T$. In this deformed triangle the warped point $\mathbf{x}' = \mathbf{W}(\mathbf{x})$ has the same barycentric coordinates as in the reference shape and we can compute $\mathbf{x}'$ as

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{bmatrix} x_i & x_j & x_k \\ y_i & y_j & y_k \end{bmatrix} \cdot \begin{pmatrix} \alpha_{t,\mathbf{x}} \\ \beta_{t,\mathbf{x}} \\ \gamma_{t,\mathbf{x}} \end{pmatrix} = B_{t,\mathbf{x}} \cdot \begin{pmatrix} x_i \\ y_i \\ x_j \\ y_j \\ x_k \\ y_k \end{pmatrix}$$

where we rearranged terms at the right side of the equation and defined

$$B_{t,\mathbf{x}} = \begin{bmatrix} \alpha_{t,\mathbf{x}} & 0 & \beta_{t,\mathbf{x}} & 0 & \gamma_{t,\mathbf{x}} & 0 \\ 0 & \alpha_{t,\mathbf{x}} & 0 & \beta_{t,\mathbf{x}} & 0 & \gamma_{t,\mathbf{x}} \end{bmatrix}$$

as the matrix encoding the barycentric coordinates of $\mathbf{x}$ in the triangle t (cf. Figure 2.3). By introducing a vertex selection matrix for each triangle t as

$$N_t = \begin{array}{c} \begin{array}{cccccccccc} \cdots & x_i & y_i & \cdots & x_j & y_j & \cdots & x_k & y_k & \cdots \end{array} \\ \left[\begin{array}{cccccccccc} & 1 & 0 & & & & & & & \\ & 0 & 1 & & & & & & & \\ & & & & 1 & 0 & & & & \\ & & & & 0 & 1 & & & & \\ & & & & & & & 1 & 0 & \\ & & & & & & & 0 & 1 & \end{array}\right] \end{array} \in \mathbb{R}^{6\times 2n} \tag{2.17}$$

we can further simplify the notation and express this mapping as an expression linear in a given shape $\mathbf{s}$:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = B_{t,\mathbf{x}} \cdot N_t \cdot \mathbf{s} \tag{2.18}$$

Rasterizing the reference shape gives a unique and ordered set of m pixel positions. When additionally the mapping from the reference shape to the deformed shape is given, the appearance observed in each training image can be expressed by concatenating intensity values, which are linearly interpolated at the warped pixel positions. These m dimensional appearance vectors are used to fill the columns of a data matrix which is again feed to a PCA to

extract the mean appearance $\bar{\mathbf{T}}$ and an orthonormal basis $T \in \mathbb{R}^{m \times k_t}$ such that we can express an appearance vector by the linear combination

$$\mathbf{T} = \bar{\mathbf{T}} + T\mathbf{q}$$

The vector $\mathbf{q} \in \mathbb{R}^{k_t}$ contains the so called appearance parameters.

Object tracking with AAMs. Once an AAM was built, i.e. $\bar{\mathbf{s}}, S, \bar{\mathbf{T}}$ and T are determined during the training phase, it can e.g. be used for tracking. For this task we want to reconstruct shape parameters $\mathbf{p}$ and appearance parameters $\mathbf{q}$ for a given input image. The optimal appearance parameter then reconstruct an appearance vector $\mathbf{T}$ that matches the concatenated image intensities $\mathbf{I}$ interpolated in the input image at warped pixel positions which are determined by the shape parameters. Following the notation in Section 2.3 the residual function can be formulated as

$$\mathbf{f} = \bar{\mathbf{T}} + T\mathbf{q} - \mathbf{I}(\mathbf{W}(\mathbf{p})) \tag{2.19}$$

where the warping function $\mathbf{W}(\mathbf{x}; \mathbf{p})$ maps rasterized pixel positions $\mathbf{x}$ to positions in the input image an can be obtained by combining Equation 2.16 with Equation 2.18:

$$\mathbf{W}(\mathbf{x}; \mathbf{p}) = B_{t,\mathbf{x}} \cdot N_t \cdot (\bar{\mathbf{s}} + S\mathbf{p}) \tag{2.20}$$

Of course, minimizing this energy function can be done with one of the straight forward optimization techniques presented in Section 2.1, but in order to make the tracking suitable for real-time applications, Matthews and Baker [MB04] formulate it as an Inverse Compositional Image Alignment (cf. Section 2.3). This requires to minimize an energy term, whose parameters occur only in the warping function $\mathbf{W}$. The formulation in Equation 2.19 does not fulfill this yet, because it additionally depends on the appearance parameters $\mathbf{q}$ occurring in the middle term.

To solve this Matthes and Baker [MB04] consider the orthonormal matrix T as base of a subspace for all possible residuals $\mathbf{f}$. When defining the orthonormal complement to this base as $T_\perp$ we can express a residual vector as a linear combination $\mathbf{f} = T\mathbf{a} + T_\perp\mathbf{b}$. Where $\mathbf{a}$ and $\mathbf{b}$ are coefficients obtained by

projecting the residual function $\mathbf{f}$ on the respective base of the linear subspace. Given this separation we can rewrite the energy function as follows

$$\begin{aligned} E &= \mathbf{f}^T\mathbf{f} = ||\mathbf{f}||^2 \\ &= \mathbf{a}^T \underbrace{T^T T}_{Id}\mathbf{a} + \mathbf{a}^T \underbrace{T^T T_\perp}_{0}\mathbf{b} + \mathbf{b}^T \underbrace{T_\perp^T T}_{0}\mathbf{a} + \mathbf{b}^T \underbrace{T_\perp^T T_\perp}_{Id}\mathbf{b} \\ &= \mathbf{a}^T\mathbf{a} + \mathbf{b}^T\mathbf{b} \\ &= ||T^T\mathbf{f}||^2 + ||T_\perp^T\mathbf{f}||^2 \end{aligned} \tag{2.21}$$

When projecting the residuals $\mathbf{f}$ (cf. Equation 2.19) onto the orthogonal complement $T_\perp$ of the base T the middle term $T\mathbf{q}$ vanish and the second term of Equation 2.21 will be independent of the appearance parameters $\mathbf{q}$. In the first term of Equation 2.21 the residuals are projected onto the base T, which leads to

$$T^T\mathbf{f} = T^T[\bar{\mathbf{T}} - \mathbf{I}(\mathbf{W}(\mathbf{p}))] + \mathbf{q}$$

Finally we notice, that for any shape parameters $\mathbf{p}$ which minimize the second term of Equation 2.21, there is a unique set of appearance parameters $\mathbf{q} = -T^T[\bar{\mathbf{T}} - \mathbf{I}(\mathbf{W}(\mathbf{p}))]$, for which the first energy term evaluates to zero. This allows to formulate the whole tracking approach as a two step algorithm: First we search for shape parameters minimizing the second energy term $||T_\perp^T\mathbf{f}||^2$ and then we compute optimal appearance vectors to finally reconstruct the observed input image.

Optimizing the shape. Since now the second term in Equation 2.21 is independent of the appearance parameters, its optimization can be formulated as an Inverse Compositional Image Alignment problem. For a moment we will consider the general optimization task, where the residuals are not restricted to the linear subspace spanned by the vectors contained in $T_\perp$:

$$f_i = \bar{T}_i(\mathbf{W}_i(\mathbf{\Delta p})) - I(\mathbf{W_i}(\mathbf{p}))$$

From Section 2.3 we know that for the Inverse Compositional Image Alignment we first needs to compute the Jacobian, which is constant throughout the optimization. Multiplying the image gradients, evaluated in the average

appearance image $\bar{\mathbf{T}}$ with the derivative of Function 2.20 with respect to the parameters $\mathbf{p}$, yields the rows of the Jacobian

$$J_{i,\cdot} = \nabla \bar{T}_i \cdot B_{t,\mathbf{x}} \cdot N_t \cdot S$$

The optimal warping function then can iteratively be found by following the steps described in Section 2.3:

1. Compute the error image $\mathbf{f} = \bar{\mathbf{T}} - \mathbf{I}(\mathbf{W}(\mathbf{p}))$
2. Solve $J^T J \Delta\mathbf{p} = -J^T \mathbf{f}$
3. Update the warp: $\mathbf{W}(\mathbf{x}; \mathbf{p}) \leftarrow \mathbf{W}(\mathbf{W}^{-1}(\mathbf{x}; \Delta\mathbf{p}); \mathbf{p})$

In order to restrict the residuals to the linear subspace spanned by the vectors in $T_\perp$, it is sufficient to restrict the Jacobian J to this subspace, since in Step 2 of the Inverse Compositional Image Alignment the residuals are projected onto the columns of the Jacobian, and if these columns are restricted to a linear subspace, $\mathbf{f}$ will be too. Projecting the columns $J_{\cdot,j}$ to the linear subspace spanned by $T_\perp$, means that we need to subtract from each column, the portions of all k_t vectors stored in T column wise:

$$J_{\cdot,j} \leftarrow J_{\cdot,j} - \sum_{i=1}^{k_t} [T_{\cdot,i} \cdot J_{\cdot,j}] \cdot T_{\cdot,i}$$

Updating the warp. To obtain the new warping function, Step 3 of the algorithm composes the current estimate $\mathbf{W}(\mathbf{x}; \mathbf{p})$ of the warping function with the inverse $\mathbf{W}^{-1}(\mathbf{x}; \Delta\mathbf{p})$ of an incremental warping function computed from the improvements $\Delta\mathbf{p}$. Before we go into detail, we look at the first order Taylor expansion $\mathbf{W}(\mathbf{x}; \Delta\mathbf{p}) \approx \mathbf{x} + \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \Delta\mathbf{p}$ of the incremental warping function and observe that $\mathbf{W}^{-1}(\mathbf{x}; \Delta\mathbf{p}) \approx \mathbf{W}(\mathbf{x}; -\Delta\mathbf{p})$, since

$$\mathbf{W}(\mathbf{W}(\mathbf{x}; -\Delta\mathbf{p}); \Delta\mathbf{p}) \approx \mathbf{W}(\mathbf{x}; -\Delta\mathbf{p}) + \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \Delta\mathbf{p} \approx \mathbf{x} - \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \Delta\mathbf{p} + \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \Delta\mathbf{p} = \mathbf{x}$$

leads to the identity function.

Computing the composed warping function $\mathbf{W}(\mathbf{W}(\mathbf{x}; -\Delta\mathbf{p}); \mathbf{p})$ means to find new vertex locations defining a deformed version of the mesh M, since then the barycentric mappings from triangles in the reference shape $\bar{\mathbf{s}}$ to the deformed shape $\mathbf{s}$ is the new warp we searching for. For this we first deduce new vertex positions from our shape model 2.16 and the improvements $\Delta\mathbf{p}$ as

$$\mathbf{s}' = (x'_1, y'_1, \ldots, x'_n, y'_n)^T = \bar{\mathbf{s}} - S\Delta\mathbf{p}$$

where $\mathbf{x}'$ is one vertex of the incrementally deformed reference shape. These vertices need further to be warped with the current estimate $\mathbf{W}(\mathbf{x}', \mathbf{p})$ to finally define the vertex positions of the deformed mesh. Since $\mathbf{x}'$ can lie anywhere in the reference mesh (even outside all triangles), it is not obvious, which triangle we should use to map these points (remember for the mapping we need barycentric coordinates w.r.t. a specific triangle). Matthews and Baker [MB04] suggest to use all triangles surrounding a reference vertex $\mathbf{X}$ and argue that this introduces an additional smoothing term to stabilize the optimization. If the triangles adjacent to $\mathbf{X}$ are $t_1, \ldots, t_L$, the point $\mathbf{x}'$ has different barycentric coordinates $B_{t_l,\mathbf{x}'}$ with respect to the triangles t_l adjacent to the reference vertex X. Then the final position $\mathbf{x}''$ can be computed by averaging the warped positions:

$$\mathbf{x}'' = \frac{1}{L} \sum_{i=1}^{L} B_{t_l,\mathbf{x}'} \cdot N_{t_l} \cdot \mathbf{s}$$

After assembling the new deformed shape of the mesh as $\mathbf{s} = (x_1'', y_1'', \ldots, x_n'', y_n'')^T$ the new shape parameters defining this estimate of the warp can simply be obtained by projecting $\mathbf{s}$ into the PCA base of our shape model, which leads to the update $\mathbf{p} \leftarrow S^T \cdot (\mathbf{s} - \bar{\mathbf{s}})$ of the shape parameters, from which we can continue the the shape optimization.

2.4. Camera Model

The outcome of an image strongly depends on the camera and the lens which was used. In this Section we present a mathematical model describing this image forming process and explain how we can efficiently extract the parameters of that model to mimic the physical behavior of real-world cameras (camera calibration). Only if we know the camera model and its parameters we can "reverse" the image forming process and reconstruct 3D structures as seen in 2D images.

2.4.1. Pinhole Camera

The pinhole camera model (cf. Figure 2.4) mathematically describes an ideal camera with an aperture of size zero. That means all light which emerges from

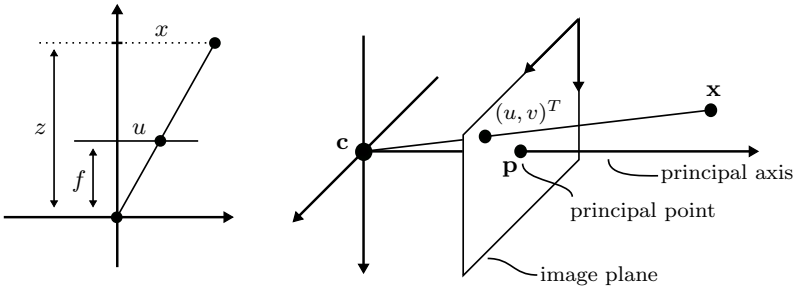


Figure 2.4.: Pinhole camera model. Left: The theorem of intersecting lines shows, that the u coordinate of the projection of x depends on its distance z and the focal length f of the camera. Right: Projection of a 3D point x onto the image plane.

a specific point in three dimensional space travels through one point, called the optical center, before it hits the image plane. The focal length f of such an ideal camera is the distance between the image plane and the optical center and has a direct influence on the size of the projected scene. Using the theorem of intersecting lines the relation of a point $(x, y, z)^T \in \mathbb{R}^3$ and its projection $(u, v)^T \in \mathbb{R}^2$ onto the image plane is given by

$$(u, v)^T = (f\frac{x}{z}, f\frac{y}{z})^T \tag{2.22}$$

and we see that the size of an object is inverse proportional to its distance z to the optical center (cf. left image in Figure 2.4).

Homogenous Coordinates. Most applications of computer graphics and computer vision use the so called *homogenous* or *projective coordinates* to represent such projections as simpler mappings which are linear in the point coordinates $(x, y, z)^T$. A two dimensional point $(u, v)^T$ can be written in homogenous coordinates as a three dimensional vector $(ku, kv, k)^T$ which represents the same point $(u, v)^T$ for all $k \neq 0$. The concept of homogenous coordinates also works in higher dimensions where an n dimensional point is represented by an $n + 1$ dimensional vector. In order to compute the Euclidean representation from a homogenous vector (ku, kv, k) one needs to perform a *de-homogenization*, where

the homogenous vector is divided by the homogenous component, such that the last component evaluates to one: $(ku, kv, k) \rightarrow (u, v, 1)$. Using homogenous coordinates we can express the projection and, as we will see later, arbitrary affine transformations like rotations, scaling and translations as simple linear mappings, thus the afore mentioned projective mapping can be expressed by the multiplication of a four dimensional vector with a 3×4 projection matrix

$$\begin{pmatrix} fx \\ fy \\ z \end{pmatrix} = \begin{bmatrix} f & & & 0 \\ & f & & 0 \\ & & 1 & 0 \end{bmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Then, the de-homogenization, i.e. in this case the division by z, of the resulting vector leads to the coordinates computed in Equation 2.22.

Intrinsic camera parameters. The 3×4 projection matrix can be split into an *intrinsic camera matrix* $K \in \mathbb{R}^{3\times3}$ and another 3×4 *extrinsic camera matrix*, which we later use to model the position and rotation of the camera in 3D space. In this simple version of a projective mapping the projection is split as

$$\begin{bmatrix} f & & & 0 \\ & f & & 0 \\ & & 1 & 0 \end{bmatrix} = \underbrace{\begin{bmatrix} f & & \\ & f & \\ & & 1 \end{bmatrix}}_{:=K} \cdot \begin{bmatrix} 1 & & & 0 \\ & 1 & & 0 \\ & & 1 & 0 \end{bmatrix}$$

where the extrinsic matrix is just the identity matrix, performing no rotation and translation of the 3D point, and where the intrinsic camera matrix only depends on the focal length. This projection computes image coordinates with respect to the principal point, which is the intersection of the optical axis with the image plane. In practice the location of the principal point on the image plane is different for each lens and camera. Therefore two more intrinsic parameters (p_x, p_y) are introduced which model the location of the principal point w.r.t. the origin of the image plane, which is typically located at the top left corner of the image. For some older cameras the pixels of the sensors do not have a quadratic shape, which leads to a different focal length for the x and y axis. In order to model this as an intrinsic parameter a scaling factor α is

introduced. Together with the principal point the intrinsic matrix of a pinhole camera can be described as

$$K = \begin{bmatrix} f & 0 & p_x \\ 0 & \alpha f & p_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2.23}$$

Note that most cameras use sensors whose pixels have a quadratic shape. Throughout this thesis we only use such cameras, where we can set the scaling factor to $\alpha = 1$.

Extrinsic camera parameters. The above example assumes that the optical center of the pinhole camera lies in the origin of a world coordinate system and that the coordinate axes of the camera is equal to the axes of the world coordinate frame. In general the mapping of points $\mathbf{x}_w = (x_w, y_w, z_w)^T$ in world coordinates to points $\mathbf{x}_c = (x_c, y_c, z_c)^T$ in camera coordinates is defined by an affine map $\mathbf{x}_c = R \cdot (\mathbf{x}_w - \mathbf{c})$, where $R \in \mathbb{R}^{3\times3}$ encodes the rotation of the camera frame w.r.t. the world coordinate system and $\mathbf{c} \in \mathbb{R}^3$ is the position of the optical center in world coordinates (cf. right image in Figure 2.4). Using homogenous coordinates this affine transformation can simply be written as

$$\mathbf{x}_c = [R|\mathbf{t}] \cdot \begin{pmatrix} \mathbf{x}_w \\ 1 \end{pmatrix}$$

where $\mathbf{t} = -R \cdot \mathbf{c}$ is a convenient abbreviation for the translational part. Together with the intrinsic camera matrix K the whole projective mapping is given by

$$\begin{pmatrix} a \\ b \\ c \end{pmatrix} = \underbrace{K[R|\mathbf{t}]}_{:=P} \cdot \begin{pmatrix} \mathbf{x}_w \\ 1 \end{pmatrix} \tag{2.24}$$

where the intrinsic and extrinsic parameters are all combined to form one projection matrix $P = [Q|\mathbf{q}] = [KR|K\mathbf{t}] \in \mathbb{R}^{3\times4}$. After multiplying a point in homogenous coordinates with the projection matrix P, the image coordinates of the projected point are obtained by the de-homogenization $(u, v)^T = (a/c, b/c)^T$.

Back projection. The projection matrix P allows it to map 3D points to points on the 2D image plane. For many computer vision applications it is important to handle the reverse direction, i.e. the construction of a ray from a given pixel or image point (for this we assume the intrinsic and extrinsic camera parameters to be known). Using homogenous coordinates a point $(u, v)^T$ on the image plane is represented as an arbitrary scalable 3D vector $\lambda \cdot (u, v, 1)^T$, which is proportional to the projection of a 3D point $\mathbf{x}_w = (x_w, y_w, z_w)^T$ lying on the viewing ray through that image point:

$$\lambda \cdot \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = Q \cdot \mathbf{x}_w + \mathbf{q} = Q \cdot (\mathbf{x}_w - \mathbf{c}) = K \cdot R \cdot (\mathbf{x}_w - \mathbf{c})$$

From this we can directly see, that any point $\mathbf{x}$ in 3D space lying on the viewing ray can be computed from the 2D point position as

$$\mathbf{x} = \mathbf{c} + \lambda \cdot Q^{-1} \cdot (u, v, 1)^T \tag{2.25}$$

Notice that for the special case, where λ equals the z-component of the vector $R \cdot (\mathbf{x}_w - \mathbf{c})$, we can directly reconstruct the position $\mathbf{x}_w$ given its projection. This is e.g. useful when reconstructing a point cloud from a given depth map, where the distances w.r.t. to the camera center is stored in each pixel of an image.

2.4.2. Real Cameras

The mathematical formulation of the pinhole camera model assumes that light travels along a straight line through one point, the optical center, before it hits the image plane. This means that the aperture of such a camera is infinitely small, such that physically no light will pass through it. Real cameras therefore must have apertures of a certain extend to capture enough light in short periods of time. This on the other hand is problematic, since light bends at the border of the aperture and the effect of diffraction becomes visible in the image. In order to compensate for such effects, complex lens systems are needed, but it is very costly to build perfect lens shapes, such that these lens system typically introduce other distortions. Chromatic aberration, e.g., is an effect where the refraction of light depends on its wavelength. The most significant distortions

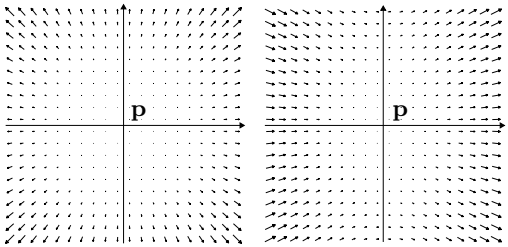


Figure 2.5.: Left: Example of a vector field describing radial lens distortion. Right: Vector field, that exemplary describes tangential lens distortion.

are the so called barrel and pincushion distortions [JW76] which deform the whole image such that straight lines in object space are curved lines in the image.

Mathematically these distortions can be separated into radial and tangential distortions, like it is done in the Brown's distortion model [Bro66]. This model basically adds two vectors to the projected point position $(u, v)^T$ obtained from the standard pinhole camera model. Radial distortion vectors $\mathbf{d}_r \in \mathbb{R}^2$ depend on a set of radial distortion parameters $k_1, k_2, k_3, \ldots$ and can be computed at a point $(u, v)^T$ as

$$\mathbf{d}_k(u, v) = \begin{pmatrix} r_u \\ r_v \end{pmatrix} \cdot (k_1 \cdot ||\mathbf{r}||^2 + k_2 \cdot ||\mathbf{r}||^4 + k_3 \cdot ||\mathbf{r}||^6 + \ldots)$$

where $\mathbf{r} = (r_u, r_v)^T = (u - p_x, v - p_y)^T$ is the vector from the principal point $\mathbf{p}$ to the pixel $(u, v)^T$. In most applications the number of radial distortion parameters is restricted to three parameters. The left image of Figure 2.5 shows a vector field, computed from one example of a radial distortion model, where pixels far away from the principal point also show large distortion vectors. Tangential distortion vectors $\mathbf{d}_t \in \mathbb{R}^2$ also influence the resulting image significantly. In practical scenarios they are modeled by two tangential parameters t_1, t_2, but similar to the radial parameters, more parameters can be used:

$$\mathbf{d}_t(u, v) = \begin{pmatrix} t_1 \cdot (||\mathbf{r}||^2 + 2r_u^2) + 2 \cdot t_2 r_u r_v \\ t_2 \cdot (||\mathbf{r}||^2 + 2r_v^2) + 2 \cdot t_1 r_u r_v \end{pmatrix} \cdot (1 + t_3 \cdot ||\mathbf{r}||^2 + t_4 \cdot ||\mathbf{r}||^4 + \ldots)$$

The right image of Figure 2.5 shows a vector field for tangential distortion where vectors are majorly oriented along the x-axis of the image. If the parameters of the distortion model are known we can compute the distorted pixel position $(u_d, v_d)^T$, i.e. the position on the image plane, where a real Photon would end up, as

$$\mathbf{d}(u,v) = \begin{pmatrix} u_d \\ v_d \end{pmatrix} = \begin{pmatrix} u \\ v \end{pmatrix} + \mathbf{d}_k(u,v) + \mathbf{d}_t(u,v) \tag{2.26}$$

Many computer vision applications need to look up image intensities at projected point positions. Incorporating an distortion model in such applications makes the computation of pixel positions computationally expensive and complicated, since we need to evaluate higher order polynomials. This is why most applications first convert the input images to undistorted images, where the correct image intensities can be found at pixel positions computed with the standard pinhole camera model. The algorithm to compute the undistorted image iterates over all integer pixel positions $(u, v)^T$, computes the real valued distorted pixel position $(u_d, v_d)^T$ and copies the interpolated image intensity from that position to the pixel (u, v).

2.4.3. Camera Calibration

The process of calibrating a single camera or even a camera rig, which consist of multiple cameras mounted on a stiff construction, always starts with a measurement procedure where several images are taken from a specific calibration pattern. Such a calibration pattern has to fulfill two basic requirements: First of all, it must contain some extraordinary points, whose exact 3D location on that pattern are known. Second of all it should be easy to recognize these points in an image, which defines 3D to 2D point correspondences for each taken image.

Once the correspondences have been collected, the task is to compute intrinsic and extrinsic camera parameters which describe the observed 2D point locations, i.e. which minimize the distances of the projected 3D points and their corresponding 2D points. The parameters to compute such projections usually contain the intrinsic parameters of the lens distortion, the parameters of the pinhole camera model and in case the task is to calibrate a whole camera rig, the extrinsic camera parameters containing the relative translation and

rotation between the cameras of the rig. In this thesis we adopted the method of Zhang [Zha00] and implemented a fully automatic calibration method which finds 3D to 2D correspondences in a fast and robust way, which will be described next.

Estimating a homography from 3D to 2D correspondences. A homography is a projective mapping between two projective spaces. In three dimensional space it can be seen as the projective mapping between to planes, e.g. from one plane in 3D space to the image plane. Using homogenous coordinates we can represent it as a linear mapping

$$\begin{pmatrix} \lambda \cdot u \\ \lambda \cdot v \\ \lambda \end{pmatrix} = \begin{bmatrix} \mathbf{h}_1^r \\ \mathbf{h}_2^r \\ \mathbf{h}_3^r \end{bmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = H \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

where H is a 3×3 matrix with rows $\mathbf{h}_1^r, \mathbf{h}_2^r, \mathbf{h}_3^r$ that maps points $(x, y)^T$ from one plane to de-homogenized points $(u, v)^T$ of another plane. If we define $\mathbf{x} = (x, y, 1)^T$ and include this de-homogenization we can directly compute the coordinates $(u, v)^T$ as

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} \frac{\mathbf{h}_1^r \cdot \mathbf{x}}{\mathbf{h}_3^r \cdot \mathbf{x}} \\ \frac{\mathbf{h}_2^r \cdot \mathbf{x}}{\mathbf{h}_3^r \cdot \mathbf{x}} \end{pmatrix} \tag{2.27}$$

Since the estimation of a homography plays a central role in several steps of the calibration process, we will shortly repeat how this can be done from a given set of 3D to 2D correspondences. Different to computing the point positions from a given homography, for this the parameters of an unknown homography explaining the observed projections need to be computed. Therefore we reformulate Equation 2.27 such that we get an expression which is linear in the parameters of the homography

$$\begin{bmatrix} \mathbf{x}^T & \mathbf{0}^T & -u \cdot \mathbf{x}^T \\ \mathbf{0}^T & \mathbf{x}^T & -u \cdot \mathbf{x}^T \end{bmatrix} \cdot \mathbf{h} = L_i \cdot \mathbf{h} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

where $\mathbf{h} = [\mathbf{h}_1^r, \mathbf{h}_2^r, \mathbf{h}_3^r]^T$ is the vector holding the parameters of the homography. For each corresponding point pair i we get a constraint matrix $L_i \in \mathbb{R}^{2 \times 9}$. If n corresponding point pairs have been measured, the concatenation of all

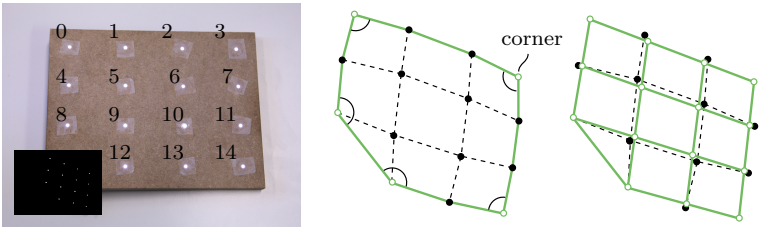


Figure 2.6.: Planar calibration patterns with 15 LED lamps. Left: under the right illumination conditions, the lamps of the pattern are easily detectable in an image. Middle: The convex hull of the detected points (green line) is a polygon with at most 11 points. Among those points the corner points (green circles) are found by identifying their characteristic large inner angles. Right: To identify the remaining points we use the homography induced by the detected corner points to predict the positions (green circles) of the remaining LED lamps. A detected (black) point is then associated to the closest predicted position of a lamp.

constraints leads to a constraint matrix L with dimension $2n \times 9$, from which we can compute the optimal homography explaining the observed projections by solving the least squares problem $L^T L \cdot \mathbf{h} \rightarrow \min$. To avoid the trivial solution, $L^T L \cdot \mathbf{h}$ is minimized subject to the constraint $\sum_i h_i^2 = 1$. Then the optimal solution for $\mathbf{h}$ is the Eigenvector of $L^T L$ corresponding to the smallest Eigenvalue.

2.4.3.1. Automatic Identification of Correspondences

Instead of using a checker board pattern, as suggested in [Zha00], we constructed a planar pattern with 15 LED lamps as depicted in the left image of Figure 2.6. The pattern was precisely fabricated such that the exact location of each LED on that planar surface is known. Since images of this pattern, taken in a dark environment, only show 15 glowing dots representing the LED lamps, the detection of those extraordinary points of the pattern is much easier than the detection of crossing sections of a checker board. This often requires manual interaction as it is e.g. needed in the Matlab Calibration Toolbox [Bou08].

Detecting the LED lamps. For all dots seen in an image, we observe that the image intensity is high at their midpoints and that the intensity drops at the border of each dot (cf. left image of Figure 2.6). This is why we model the intensity distribution as a Gaussian function. In order to detect the midpoints of the dots representing the LED lamps, we compute the convolution of the captured image with that Gaussian function, which can efficiently be done using the FFTW library [FJ05]. At the midpoints of these glowing dots the convolution with a Gaussian function reaches a local maximum. We identify a set of 15 pixels, representing these midpoints, by successively identifying pixels from a set of candidates having high intensity values in the convoluted image. After each identification step we apply a non maximum suppression and remove all surrounding pixels from the set of candidate pixels within a user given radius and proceed with the identification of the next LED until we found 15 LED or the next image intensity is below a preset threshold, in which case we declare the detection as invalid.

Computing a one-to-one correspondence is necessary to correctly calculate the re-projection errors in later steps of the calibration pipeline. Once the center of each glowing dot has been identified, the task is to assign a specific LED lamp index to it. Then the 2D position of the midpoint and the 3D position of the LED lamp on the calibration pattern define a corresponding point pair. In order to break the symmetry of the calibration pattern and thereby make this assignment for all LEDs unique, we omit one LED in the grid of LEDs (cf. lower left corner of the calibration pattern in Figure 2.6).

The algorithm used to compute the correspondences of 2D midpoints and 3D positions of the LED lamps starts with the extraction of the convex hull of the detected midpoints. The result is a two dimensional polygon with at most 11 vertices as shown as the green border in the middle image of Figure 2.6. Among these vertices we identify the corner points of our pattern (LEDs 0,3,8,12 and 14) by successively deleting that vertex of the 2D polygon which incident edges enclose the largest angle. This decimation is repeated until only five vertices remain, which are assumed to be the corners of the LED pattern. Since the counter-clockwise ordering of the polygon equates the counter-clockwise ordering of the LEDs in our pattern, the possible mappings between the midpoints

and the LED lamps at the corners reduce to five possible configurations which can be generated by a cyclic permutation. Each of this permutations gives a unique set of five 2D to 3D point correspondences from which we compute a homography as described in Section 2.4.3. This homography is used to project all 15 LED lamps into the image and for each LED lamp we define its corresponding 2D point as that midpoint of a glowing dot having the minimal distance to the projection of the LED. As the total error we accumulate the individual re-projection errors and accept the set of corresponding 2D to 3D point pairs with minimal total error as the set holding the correct correspondences.

2.4.3.2. Closed form Intrinsic Calibration

With the algorithm described in the previous section it is possible to detect LED lamps in an image and relate the detected 2D points (u, v) to 3D point positions $(x, y, 0)$ located on the planar calibration pattern. We could use the algorithm described in Section 2.4.3 to compute a special projective mapping, i.e. a homography H, which only characterizes the relation between points on one particular plane (the calibration pattern) and the image plane. In order to be able to predict the 2D positions of other 3D points not lying on the calibration pattern, we would prefer a more general model, instead of one specific homography. For a pinhole camera this general model is the intrinsic calibration, which is mathematically described by the shearless 3×3 matrix K mapping points to squared pixels, i.e. where $\alpha = 1$ (cf. Equation 2.23). Zhang [Zha00] describes a closed form solution, how the intrinsic matrix can be computed from a set of homographies, which are obtained from images showing the calibration pattern at different positions and orientations w.r.t. to the camera.

The first step to compute the intrinsic matrix K is to assume that the homographies computed for all images are composed of the constant intrinsic matrix K and a rigid transformation, which is variable for different images. Starting with the general projection function as described in Equation 2.24, we

see that 3D points $(x, y, 0)$ stored in coordinates of the calibration pattern, are projected to the image plane as

$$\begin{pmatrix} \lambda \cdot u \\ \lambda \cdot v \\ \lambda \end{pmatrix} = K \cdot \begin{bmatrix} \mathbf{r}_1 & \mathbf{r}_2 & \mathbf{r}_3 & \mathbf{t} \end{bmatrix} \begin{pmatrix} x \\ y \\ 0 \\ 1 \end{pmatrix} = \underbrace{K \cdot \begin{bmatrix} \mathbf{r}_1 & \mathbf{r}_2 & \mathbf{t} \end{bmatrix}}_{=H} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

where the perpendicular vectors $\mathbf{r}_1$ and $\mathbf{r}_2$ lie in the plane of the calibration pattern and represent its orientation with respect to the camera, while $\mathbf{t}$ is related to the position of the pattern. If $[\mathbf{h}_1\ \mathbf{h}_2\ \mathbf{h}_3]$ are the columns of one homography H, the two axes $\mathbf{r}_1$ and $\mathbf{r}_2$ of the pattern are related to these columns by $\mathbf{h}_i = K\mathbf{r}_i$ for $i \in \{1, 2\}$. The idea of finding the best suitable matrix K explaining the observed homographies in all images, is to constrain the two axes of the pattern to be orthonormal, i.e. $\mathbf{r}_1^T \cdot \mathbf{r}_1 = 1 = \mathbf{r}_2^T \cdot \mathbf{r}_2$ and $\mathbf{r}_1^T \cdot \mathbf{r}_2 = 0$, which leads to a pair of equations for each homography of the form

$$\begin{aligned} \mathbf{h}_1^T \cdot B \cdot \mathbf{h}_2 &= 0 \\ \mathbf{h}_1^T \cdot B \cdot \mathbf{h}_1 - \mathbf{h}_2^T \cdot B \cdot \mathbf{h}_2 &= 0 \end{aligned} \tag{2.28}$$

where $B = K^{-T}K^{-1}$ is a symmetric matrix which can more compactly be written as a six dimensional vector $\mathbf{b}' = [B_{11}, B_{22}, B_{33}, B_{12}, B_{13}, B_{23}]^T$ only using the three diagonal and three off-diagonal elements. When plugging in our definition of the intrinsic matrix, B expands to

$$B = \frac{1}{f^2} \cdot \begin{bmatrix} 1 & 0 & -p_x \\ 0 & 1 & -p_y \\ -p_x & -p_y & (p_x^2 + p_y^2 + f^2) \end{bmatrix}$$

and we observe that it is possible to represent it in a more compact form using the four dimensional vector $\mathbf{b} = [B_{11}, B_{33}, B_{13}, B_{23}]^T$. Once $\mathbf{b}$ is known, we can directly deduce the intrinsic parameters f, p_x, p_y, as e.g. $f = 1/\sqrt{B_{11}}$.

The task then is to determine the vector $\mathbf{b}$ from the set of Equations 2.28 which are available for each image showing the calibration pattern. Defining the vector

$$\mathbf{v}_{ij} = \begin{pmatrix} h_{1i} \cdot h_{1j} + h_{2i} \cdot h_{2j} \\ h_{3i} \cdot h_{3j} \\ h_{1i} \cdot h_{3j} + h_{3i} \cdot h_{1j} \\ h_{2i} \cdot h_{3j} + h_{3i} \cdot h_{2j} \end{pmatrix}$$

we can rewrite the Equations 2.28 in terms of the unknown vector $\mathbf{b}$ as

$$\underbrace{\begin{bmatrix} v_{12}^T \\ v_{11}^T - v_{22}^T \end{bmatrix}}_{:=V} \cdot \mathbf{b} = \mathbf{0}$$

Again, the optimal solution in the least squares sense is to set $\mathbf{b}$ to the Eigenvector of the matrix $V^T V$ corresponding to the smallest Eigenvalue, from which we can compute the closed form solution for the intrinsic camera parameters.

2.4.3.3. Closed form Extrinsic Calibration

So far we only considered the calibration of a single camera, where we computed the intrinsic matrix K from a set of homographies deduced from images which show the calibration pattern from different positions. The intrinsic matrix, together with the observed homographies, directly give us the position and orientation of the calibration pattern by the relation $\begin{bmatrix} \mathbf{r}_1 & \mathbf{r}_2 & \mathbf{t} \end{bmatrix} = K^{-1}H$, such that we can represent the positions of the LEDs in the coordinate system of the camera as

$$\mathbf{x}_r = \begin{bmatrix} \mathbf{r}_1 & \mathbf{r}_2 & \mathbf{t} \end{bmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \underbrace{\begin{bmatrix} \mathbf{r}_1 & \mathbf{r}_2 & \mathbf{r}_3 & \mathbf{t} \end{bmatrix}}_{:=T_r} \cdot \begin{pmatrix} x \\ y \\ 0 \\ 1 \end{pmatrix} \tag{2.29}$$

where $\mathbf{r}_3$ can be computed by the cross product $\mathbf{r}_1 \times \mathbf{r}_2$. Note that the matrix T_r depends on the orientation of the pattern as observed in a specific image i and should correctly noted as T_r^i.

Now we have all ingredients to extend our setup to a stereo system, where we add another camera which is rigidly linked to the first camera we considered so far. In this context *rigidly linked* means, that there is a fixed transformation that maps 3D points from the coordinate system of one camera to the other. Without loss of generality we can define the coordinate system of one camera r of your rig to be the identity, and we call this camera the *reference camera*. With respect to the coordinate system of one camera, the 3D positions of each LED lamp observed in each image can be computed using Equation 2.29, which

gives us a list of points $\mathbf{x}_r^j$. If the second camera is synchronized with the reference camera and observes the same pattern at the same time, we can compute an additional list of points $\mathbf{x}_c^j$ representing the same LED lamps in the coordinate system of the second camera. Observe, that when capturing n images of a calibration pattern with 15 lamps, this would imply that both lists contain a point cloud of $15 \cdot n$ points and that both lists are in full correspondence. By minimizing

$$E = \sum_j ||R_{r,c}\mathbf{x}_r^j + \mathbf{t}_{r,c} - \mathbf{x}_c^j||^2 \tag{2.30}$$

as described in [Hor87], we can compute the rigid transformation $T_{r,c} = [R_{r,c}|\mathbf{t}_{r,c}]$ that models the rigid connection between both cameras and that maps points from the coordinate system of the reference camera to the coordinate system of c.

To stabilize this initial estimate for a multi-camera rig, we first set the reference camera to that camera which most often sees the calibration pattern simultaneously with any other camera. Then we successively select a pair of cameras c and c', with largest corresponding point clouds and where the transformation $T_{r,c}$ to the camera c was already computed (note that the transformation to the reference camera is the identity $T_{r,r} = Id$). Using Equation 2.30 we can compute the rigid transformation $T_{c,c'}$ from c to c' and thereby the transformation between the reference camera and the new camera c' to be registered as $T_{r,c'} = T_{c,c'} \cdot T_{r,c}$.

After the initial intrinsic and extrinsic calibration, we got a first estimate for the intrinsic parameters of all cameras (without distortion), the 3D positions of the LED lamps w.r.t. each individual camera and transformations, describing the rigid connections between the cameras of our rig.

2.4.3.4. Refining the Calibration

The closed form calibrations approximate the whole projection process and especially omit an distortion model in order to derive an estimate for the intrinsic and extrinsic camera parameters from linear systems of equations. But the correct computation of the projection $(u_d, v_d)^T$ of a point $\mathbf{x} = (x, y, 0, 1)^T$ into an image i of a camera c is computationally more involved. First we apply

a set of linear transformations to compute the homogenous representation of the undistorted pixel coordinates w.r.t. to camera c as

$$\begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{bmatrix} f & 0 & p_x & 0 \\ 0 & f & p_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \cdot T_{r,c} \cdot T_r^i \begin{pmatrix} x \\ y \\ 0 \\ 1 \end{pmatrix}$$

then the distortion model described in Equation 2.26 is used to compute the distorted pixel position as $\mathbf{d}^j(a/c, b/c)$. After detecting the positions $\mathbf{x}^j$ of the LEDs (cf. Section 2.4.3.1) we know in which image i of which camera c its projected midpoint (u^j, v^j) is located. This leads to a set of residuals

$$\mathbf{f}_j = \mathbf{d}^j - \begin{pmatrix} u^j \\ v^j \end{pmatrix} \in \mathbb{R}^2$$

modeling the difference between the measured point positions and the projections predicted by our mathematically model. Under the assumption, that a number of C cameras are mounted on the rig and that each camera sees the pattern in n images, the concatenation of all differences leads to a residual vector $\mathbf{f} \in \mathbb{R}^{2 \cdot 15 \cdot C \cdot n}$. The aim is to refine the rigid transformation of the pattern, the mounting of the cameras, the intrinsic parameters and the parameters of the distortion model such that they minimize the energy $E = \mathbf{f}^T\mathbf{f}$.

This mathematical formulation has a nice geometric interpretation. Considering the images simultaneously taken at one specific time point, we can compute 3D viewing rays originating from the camera centers through the detected midpoints, by applying the reverse projection pipeline, which is determined by its parameters. These parameters minimizing E will produce viewing rays which exactly intersect at the 3D position of LED lamps. This is why the application of the Levenberg-Marquard algorithm to this least squares problem is called Bundle Adjustment [ESN06].

Although most parameters (throughout this thesis, we use two radial and two tangential distortion parameters) are directly named in the residual function, it is not clear how to parameterize the rigid transformations T_r^i and $T_{r,c}$. One

way to do this is to represent $T = [R|\mathbf{t}]$ by its displacement vector $\mathbf{t}$ and three rotation matrices

$$R_\alpha = \begin{bmatrix} \cos\alpha & \sin\alpha & 0 \\ -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad R_\beta = \begin{bmatrix} 1 & 0 & 0 \\ \cos\beta & \sin\beta & 0 \\ -\sin\beta & \cos\beta & 0 \end{bmatrix} \quad R_\gamma = \begin{bmatrix} \cos\gamma & \sin\gamma & 0 \\ -\sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

defined by the Euler angles (α, β, γ). To avoid a Gimbal lock at large angles, we consider the Euler matrices as an incremental rotation added to the initial rotation R and parameterize a rigid transformation as

$$T = [R \cdot R_\alpha \cdot R_\beta \cdot R_\gamma | \mathbf{t}]$$

by using the six parameters $(\alpha, \beta, \gamma, t_x, t_y, t_z)$.

Calibration Accuracy We evaluated this approach on four different image sets captured with four synchronized cameras. Later we will use the same camera rig to capture facial movements. For each calibration set we took 1500 to 2000 shots resulting in approximately 6000 images potentially seeing the calibration pattern. All images have a resolution of 780×580 pixels. The calibration result is summarized in Table 2.1. Whenever the calibration pattern could be detected in an image, it is possible to measure the projection error (measured in pixels) for each of the 15 LEDs, resulting in a number of 50.000 - 70.000 projections in total (cf. second column of Table 2.1). The last two columns of the Table show a stable average distance of about 0.35 pixels between the detected LED points and its projections, which are computed using the intrinsic (with distortion) and extrinsic camera parameters.

2.5. Stereo Reconstruction

The aim of Stereo Reconstruction is to compute 3D shapes purely from two images taken at the same time. In contrast to Section 2.4.3 where we optimized camera parameters such that the projection of an object with known 3D shape fits to the observed images, we now assume that the intrinsic and extrinsic parameters of the camera are given and that we optimize for the 3D shape of the object itself. In the end, Stereo Reconstruction approaches are based

	# Projections	Ø Error	σ Error
Calibration 1	60465	0.36	0.30
Calibration 2	57345	0.35	0.3
Calibration 3	69000	0.38	0.32
Calibration 4	51870	0.38	0.33

Table 2.1.: Calibration accuracy for four different calibration sets. The second column states the number of total point projections. The Ø Error is the average distance (in pixels) between the detected point and its computed projection (third column). The fourth column shows the standard deviation for these errors.

on the principal of triangulation. The idea is to identify for each pixel in the left image an corresponding pixel in the right image. Under the assumption that both corresponding points see the same patch of a 3D surface, we can use Equation 2.25 to compute rays through the corresponding pixels that intersect at a 3D point lying on that surface. In what follows we will describe how to find the corresponding pixels and explain the principal of surfel reconstruction which extends computation of the pure position of the patch by simultaneously computing a surface normal to get a correctly oriented surface patch in 3D space.

2.5.1. Epipolar Geometry

Consider a calibrated camera rig with two cameras where the left and the right camera centers are located at the endpoints $\mathbf{c}_l$ and $\mathbf{c}_r$ of the so called *baseline*. For any point on the left image plane Equation 2.25 defines a viewing ray, which, together with the baseline, spans the so called *epipolar plane*. Projecting the viewing ray onto the right image plane results in a 2D line, the *epipolar line*. For a given point this line can be obtained by constructing the epipolar plane using back-projection and intersecting it with the right image plane. Figure 2.7 illustrates this geometric construction and it becomes clear that all possible epipolar lines, constructed from any point on the left image plane, intersect in only one point. This point is called the *epipolar point* and it can

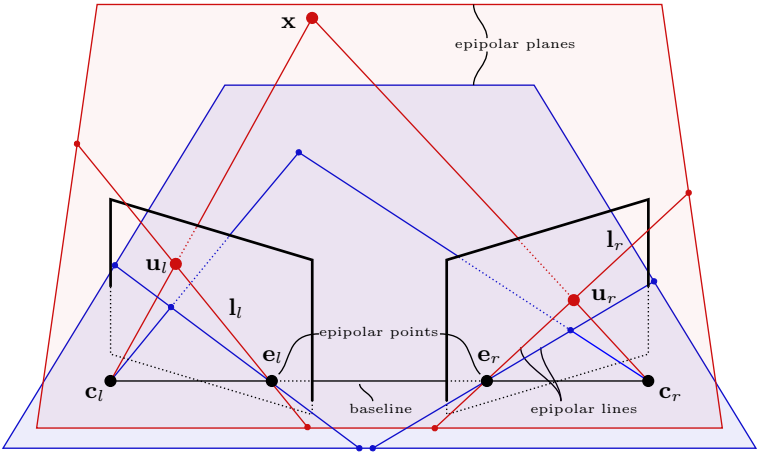


Figure 2.7.: Geometric construction of epipolar lines and the two unique epipolar points.

be computed by projecting the camera center $\mathbf{c}_l$ onto the right image plane as $\mathbf{e}_r = P_r \mathbf{c}_l$. Mathematically the epipolar line can elegantly be obtained using the so called *fundamental matrix* $F \in \mathbb{R}^{3\times3}$. Representing points in homogenous coordinates, the fundamental matrix maps points from one image plane to epipolar lines in the other image plane:

$$\mathbf{l}_r = F\mathbf{u}_l \qquad\qquad \mathbf{l}_l = F^T\mathbf{u}_r$$

and is obtained from the projection matrices P_l, P_r as $F = [\mathbf{e}_r]_x P_r P_l^+$, where

$$[\mathbf{e}_r]_x = \begin{bmatrix} 0 & -e_{z,r} & e_{y,r} \\ e_{z,r} & 0 & -e_{x,r} \\ -e_{y,r} & e_{x,r} & 0 \end{bmatrix}$$

is a skew symmetric matrix representing the cross product and $P_l^+ = P_l^T(P_l P_l^T)^{-1}$ is the pseudo inverse of the left projection matrix.

Stereo reconstruction approaches find for each point $\mathbf{u}_l$ of the left image plane a corresponding point $\mathbf{u}_r$ on the right image plane to compute the 3D

position $\mathbf{x}$ of a surface point. Under the consideration of the epipolar geometry this two dimensional search problem can be restricted to a one dimensional line search, since the corresponding point $\mathbf{u}_r$ can, by construction, only lie on the epipolar line $\mathbf{l}_r$, i.e. $\mathbf{u}_r^T \mathbf{l}_r = 0$. This is called the epipolar constraint. For corresponding points it can be conveniently formulated using the fundamental matrix as $\mathbf{u}_r^T F \mathbf{u}_l = 0$.

2.5.2. Epipolar Rectification

Although the epipolar constraint simplifies the search for the corresponding pixel considerably, there is an even more convenient setting where both cameras have the same local coordinate system R (cf. Equation 2.24). Then both image planes are coplanar resulting in parallel epipolar lines, i.e. they intersect at infinity (cf. Figure 2.7). When further adjusting the cameras rotation around the viewing direction, we can even make sure that the epipolar lines are all parallel to the x-axis of the image plane and that corresponding epipolar lines have the same y coordinate. Such a setting would strongly simplify the implementation of any algorithm, which identifies corresponding image points to reconstruct an observed 3D point: for any point (u_l, v) on the left image plane, we only need to find a corresponding point $(u_r, v) = (u_l + d, v)$ on the right image plane by adjusting a single offset parameter $d = u_r - u_l$, called the *disparity*. Note, that for simplicity we express pixel coordinates relative to the principal point, i.e. the origin of the image coordiante system is assumed to lie on the prinicpal point (cf. Figure 2.8). In this rectified setting there is a simple linear relation between the disparity d and the depth value z of the point to be reconstructed. From the theorem of intersecting lines (cf. Figure 2.8) we know that

$$\frac{x_l}{z} = \frac{u_l}{f} \qquad \text{and} \qquad \frac{x_r}{z} = \frac{u_r}{f}$$

where f is the common focal length of both cameras and x_l (x_r) is the x-coordinate of the 3D point w.r.t. to the left (right) camera. Subtracting both equations and substituting $x_l = b + x_r$ leads to the linear relation

$$\frac{x_r - x_l}{z} = \frac{u_r - u_l}{f} \qquad \Leftrightarrow \qquad z = \frac{-b \cdot f}{d} \tag{2.31}$$

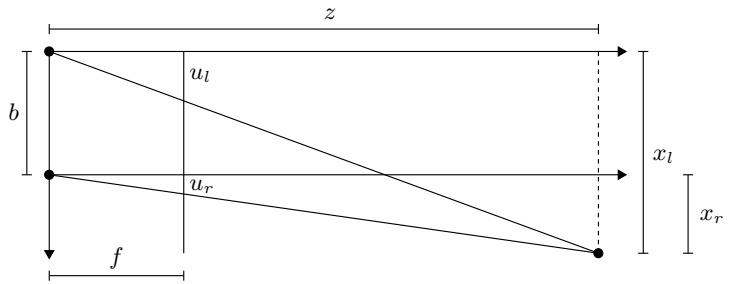


Figure 2.8.: The linear relation between disparity and depth follows from the theorem of intersecting lines.

from which we can directly compute the depth value of a point from the known disparity.

From a practical point of view it is hard to precisely construct a camera rig with this convenient configuration. But it is possible to arbitrarily define new projection matrices P_l' and P_r' for both cameras which project 3D points to coplanar image planes [Sze10]. To complete the construction of the rectified configuration, the associated images need to be distorted to match the projective behavior of these new matrices. Let $P_i = [Q_i|\mathbf{q}_i] = K_i[R_i|\mathbf{t}_i]$ with $i \in \{r, l\}$ be the original projection matrices of the left and the right camera and let $P_i' = [Q_i'|\mathbf{q}_i'] = K_i'[R|\mathbf{t}_i']$ with $i \in \{r, l\}$ be the new projection matrices, where both cameras share a common coordinate frame represented by $R = [\mathbf{x}, \mathbf{y}, \mathbf{z}]^T$. To align the $\mathbf{x}$ and $\mathbf{y}$-axis of this coordinate frame to the horizontal and vertical axis of the image plane we define

$$\mathbf{x} = \frac{\mathbf{c}_r - \mathbf{c}_l}{||\mathbf{c}_r - \mathbf{c}_l||} \qquad \text{and} \qquad \mathbf{y} = \frac{\mathbf{z}_l \times \mathbf{x}}{||\mathbf{z}_l \times \mathbf{x}||}$$

such that $\mathbf{x}$ is the normalized vector connecting both camera centers $c_i = -Q_i^{-1}\mathbf{q}_i$, with $i \in \{l, r\}$ and $\mathbf{y}$ is a vector perpendicular to $\mathbf{x}$ and the old viewing direction $\mathbf{z}_l$ extracted from the third row of R_l. To complete R, the missing axis is set to $\mathbf{z} = \mathbf{x} \times \mathbf{y}$. With R and the positions of the camera centers, the translational part of the new rigid transformation becomes $\mathbf{t}_i' = -R\mathbf{c}_i$ where $i \in \{l, r\}$.

Since both image planes should have the same distance to the camera center and corresponding epipolar lines should lie at the same y coordinate, we set $f = (f_l + f_r)/2$ and $p_y = (p_{y,l} + p_{y,r})/2$ to construct the new left and right intrinsic matrices as

$$K'_l = \begin{bmatrix} f & 0 & p_{x,l} \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix} \qquad \text{and} \qquad K'_r = \begin{bmatrix} f & 0 & p_{x,r} \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix}$$

Observe, that the x coordinate of the principal points ($p_{x,l}$ and $p_{x,r}$) are still unknown. When these parameters are adjusted such that the principal point of the original image will later be located at the same x coordinate in the distorted image, we avoid the image to be cropped due to the rectification. To achieve this we compute a mapping from the original image with projection matrix $P = [Q|\mathbf{q}]$ to the new image with new projection matrix $P' = [Q'|\mathbf{q}] = K'[R|t']$. Using Equation 2.24 and 2.25 this mapping is described by the homography

$$\begin{aligned} \begin{pmatrix} h_x(u,v) \\ h_y(u,v) \\ h_z(u,v) \end{pmatrix} &= Q' \left(\mathbf{c} + \lambda Q^{-1} \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} \right) + \mathbf{q}' = \lambda Q' \cdot Q^{-1} \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} \\ &= K' \cdot \lambda \cdot R \cdot Q^{-1} \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} =: K' \cdot \begin{pmatrix} a(u,v) \\ b(u,v) \\ c(u,v) \end{pmatrix} \end{aligned}$$

where the center of the camera $\mathbf{c}$ is the same for both projections P and P'. Requiring that the mapped principal point of the original image keeps its x coordinate in the distorted image means that the following equality must hold

$$\frac{h_x(p_x, p_y)}{h_z(p_x, p_y)} = p_x$$

from which the missing values of K' are computed for the left and right image analogously as $p'_x = p_x - f \cdot \frac{a(p_x,p_y)}{c(p_x,p_y)}$.

In order to actually rectify the image by distorting the original image, we use an algorithm similar to undistort an image (cf. Section 2.4.2): We iterate of all integer pixel positions $(u', v')^T$ and compute the real valued pixel position $(u, v)^T$ in the original image using the inverse homography $Q \cdot Q'^{-1}$. Then we copy the intensity value, interpolated at $(u, v)^T$ in the original image, to the pixel position $(u', v')^T$ of the distorted image.

2.5.3. Surfel Reconstruction

One of your 3D multi-view stereo reconstruction methods is a modified version of the *surfel fitting* approach introduced by Habbecke et al. [HK06]. The input for this algorithm is a set of images obtained from calibrated cameras and an initial estimate of a surface element (surfel) defined by a point and a normal. To calibrate the cameras we employed the calibration method outlined in Section 2.4.3.

The basic idea is to use the given projection matrices and the plane associated to define a warping function, that depends on the parameters of the plane and that maps pixels from the reference image to the comparison image. In the optimization we can then measure the differences in pixel intensities to drive the update of the parameters. The warping function is designed as a homography, that maps pixels perspectively correct. Thereby, it is superior to ordinary 3D stereo reconstruction which often compare correlation windows of a fixed size in image space, which does not consider any perspective distortions.

From an initial point position $\mathbf{p} \in \mathbb{R}^3$ and a normal $\mathbf{n} \in \mathbb{R}^3$ we infer the parameters of the plane equation $\mathbf{N} = (\mathbf{n}^T, \delta)^T$, where $\delta = -\mathbf{n}^T\mathbf{p}$. We assume that the plane is associated to a reference image T and a set of comparison images

$$\begin{aligned} \mathbf{ref}(\mathbf{p}) &= T \\ \mathbf{comp}(\mathbf{p}) &= \{I_{c_1}, \ldots, I_{c_k}\} \quad c_1, \ldots c_k \in \{1, \ldots, C\} \end{aligned}$$

where C is the total number of views. The reference image can, for example, be chosen as the image where viewing direction and the vertex normal are closest to parallel, while comparison images have to fulfill some visibility criterion. For simplicity, from now on we consider only one comparison image I and assume the projection matrices for the reference and the comparison image to be

$$\mathbf{P}_r = [\mathbf{Q}_r|\mathbf{q}_r] \quad \text{and} \quad \mathbf{P}_c = [\mathbf{Q}_c|\mathbf{q}_c]$$

According to [HZ03], the homography that maps pixels from the reference image over the plane to the comparison image is defined by the 3×3 matrix

$$H(\mathbf{N}) = \left(\delta\mathbf{Q}_c - \mathbf{q}_c\mathbf{n}^T\right)\left(\delta\mathbf{Q}_r - \mathbf{q}_r\mathbf{n}^T\right)^{-1} \tag{2.32}$$

Using this Equation in an optimization process is not easy, since it involves the inversion of a 3×3 matrix, which depends on the parameters to be optimized. To simplify this plane induced homography we transform the whole scene, such that the projection matrix of the reference camera equals the identity. Therefore, we first define the matrix

$$\mathbf{B} = \begin{bmatrix} Q_r^{-1} & -Q_r^{-1} \cdot \mathbf{q}_r \\ \mathbf{0}^T & 1 \end{bmatrix} \in \mathbb{R}^{4\times4}$$

When augmenting a projection $P\mathbf{x}$ with the identity $B \cdot B^{-1}$ we obtain a new projection matrix $P' = P \cdot B$ and a transformed point $\mathbf{x}' = B^{-1}\mathbf{x}$, where $P'\mathbf{x}' = P \cdot B \cdot B^{-1}\mathbf{x}$ leads to the same result. For the reference projection matrix we obtain the wanted effect, since $P'_r = P_r \cdot B = [I_3|\mathbf{0}]$ equals the identity matrix simplifying Equation 2.32. Besides transforming projection matrices and scene points, we can also transform a plane equation $\mathbf{N}$ using B. A point $\mathbf{x} = B\mathbf{x}'$ lying on the plane fulfills

$$\mathbf{x}^T\mathbf{N} = \mathbf{x}'^T \cdot B^T\mathbf{N} = 0$$

From this equation we can directly infer the new plane equation which needs to be transformed by the transpose of B, such that $\mathbf{N}' = B^T\mathbf{N}$.

If the plane is not passing through the center of the reference camera, we can divide all components of $\mathbf{N}$ by its fourth component δ and use only three parameters to describe the plane by its normalized plane equation where $\delta' = 1$. In what follows we express the transformed and normalized plane equation by the three dimensional vector $\mathbf{n} = (n_x, n_y, n_z)^T$, which leads to a minimal parametrization of the plane equation. Using this transformed and normalized plane equation, we can drastically simplify Equation 2.32 and express the plane induced homography as

$$H(\mathbf{n}) = \mathbf{Q}'_c - \mathbf{q}'_c\mathbf{n}^T$$

where we lost the inversion of a matrix. This homography further defines the warping function $\mathbf{W}(\mathbf{x}, \mathbf{n})$ which maps pixels $\mathbf{x} = (u, v)^T$ from the reference

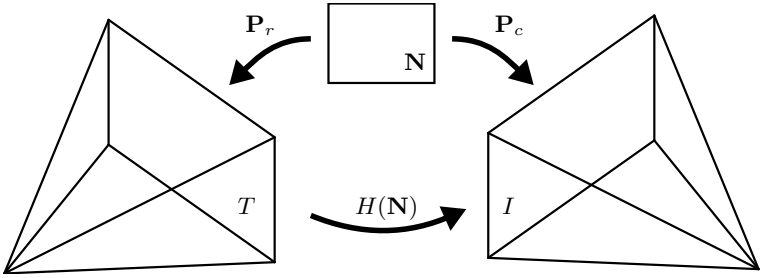


Figure 2.9.: The 3D plane together with the image projection matrices define a homography H which maps image points from T to points in the image I.

image to the comparison image (cf. Figure 2.9) and which depends on the plane parameters

$$\mathbf{W}(\mathbf{x}, \mathbf{n}) = \begin{pmatrix} \frac{a}{c} \\ \frac{b}{c} \end{pmatrix} \qquad \text{with} \qquad \begin{pmatrix} a \\ b \\ c \end{pmatrix} = H(\mathbf{n}) \cdot \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} \tag{2.33}$$

Basically, we formulate the optimization process as an additive image alignment problem (cf. Section 2.3.1) and define a residual function that measures intensity differences

$$\mathbf{f} = \mathbf{I}(\mathbf{W}(\mathbf{n})) - \mathbf{T}$$

where we again used an abbreviate notation and concatenated image intensities to get the vectors $\mathbf{I}$ and $\mathbf{T}$ (cf. Section 2.3.1). For the reference image T these pixels are chosen from a small region (typically a window of 15×15 pixels) around the projection of the input point $\mathbf{p}$. Notice that when we include multiple comparison cameras, we can simply extend the vector $\mathbf{f}$ by additional measurements.

To optimize the associated energy function $E(\mathbf{n}) = \mathbf{f}^T\mathbf{f}$, we employ the Levenberg-Marquard algorithm (cf. Section 2.1), which, in general, shows a more stable convergence behavior [NW06]. For this we need to evaluate the Jacobian, whose rows are computed from the derivatives of the components of $\mathbf{f}$ as

$$J_{i,\cdot} = \frac{\partial f_i}{\partial \mathbf{n}} = \nabla I(\mathbf{W}_i(\mathbf{n})) \frac{\partial \mathbf{W}_i(\mathbf{n})}{\partial \mathbf{n}}$$

where we made use of the short notation $\mathbf{W}_i$ that represents the warped position of the i^{th} reference pixel. The first term in this equation is simply the image gradient at the warped pixel position. The derivative of the second term involves the quotient rule and leads to a matrix

$$\frac{\partial \mathbf{W}_i(\mathbf{n})}{\partial \mathbf{n}} = \begin{bmatrix} \frac{\frac{\partial a}{\partial n_x} \cdot c - a \cdot \frac{\partial c}{\partial n_x}}{c^2} & \frac{\frac{\partial a}{\partial n_y} \cdot c - a \cdot \frac{\partial c}{\partial n_y}}{c^2} & \frac{\frac{\partial a}{\partial n_z} \cdot c - a \cdot \frac{\partial c}{\partial n_z}}{c^2} \\ \frac{\frac{\partial b}{\partial n_x} \cdot c - b \cdot \frac{\partial c}{\partial n_x}}{c^2} & \frac{\frac{\partial b}{\partial n_y} \cdot c - b \cdot \frac{\partial c}{\partial n_y}}{c^2} & \frac{\frac{\partial b}{\partial n_z} \cdot c - b \cdot \frac{\partial c}{\partial n_z}}{c^2} \end{bmatrix} \in \mathbb{R}^{2\times 3}$$

where $\frac{\partial (a,b,c)^T}{\partial n_x} = -\mathbf{q}_c' u$, $\frac{\partial (a,b,c)^T}{\partial n_y} = -\mathbf{q}_c' v$ and $\frac{\partial (a,b,c)^T}{\partial n_z} = -\mathbf{q}_c'$.

From there we can apply the Levenberg-Marquard algorithm described in Section 2.1 and iteratively compute updates of the plane parameters by solving the linear system

$$(J^T J + \lambda I_3)\Delta \mathbf{n} = -J^T \mathbf{f} \tag{2.34}$$

where again λ is a damping factor controlling the influence of the regularization matrix. The update $\Delta \mathbf{n}$ is accepted and the damping factor is reduced if they lead to a smaller energy value, i.e. if $E(\mathbf{n} + \Delta \mathbf{n}) < E(\mathbf{n})$. For stability reasons we normalize the pixel intensities within the small image regions by subtracting the average intensity of that region.

After the optimization is converged, the 3D position of the reconstructed surfel is obtained by back projecting (cf. Equation 2.25) the center of the 15×15 pixel region to obtain a ray in 3D space. The intersection of the ray with the Surfel leads to a reconstructed point position.

2.6. Geometry Processing

In this section we describe some basic approaches to deform triangular meshes. Throughout this thesis, we use *Laplace Mesh Editing* [SCOL*04] and *As-rigid-as-possible Mesh Editing* [SA07], whenever we need to non-rigidly register one shape to another. We use these techniques especially when we need to deform a generic *face template* to fit a dense or a sparse point cloud. By carefully defining the necessary modeling constraints, we ensure that the resulting meshes are in full correspondence to each other. The third technique we describe at this point is the *Deformation Transfer* [SP04, BSPG06] for triangle meshes, which

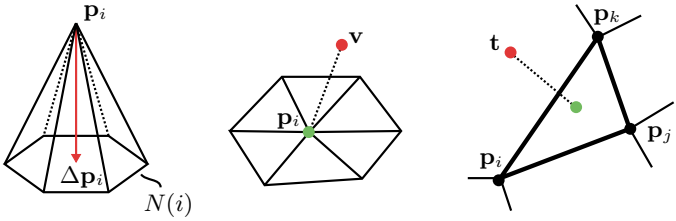


Figure 2.10.: Left: One-ring neighborhood to compute the Laplace operator. Middle: A vertex constraint minimizes the distance between a mesh vertex (green) and an attraction point (red). Right: A face constraint minimizes the distance between a point lying on a triangle (green) and an attraction point (red).

is originally designed to map local triangle deformations from one mesh to another. But in this thesis it is also the key technique to build our database of dynamic facial expressions.

2.6.1. Laplace Mesh Editing

The input for this approach is a triangle mesh $M = (V, T)$ represented by a set of vertices $V = (\mathbf{q}_1, \ldots, \mathbf{q}_n)$ and a set of triangles T. The aim is to find new vertex positions $V' = (\mathbf{p}_1, \ldots, \mathbf{p}_n)$ such that on the one hand some positional constraints for the vertices $\mathbf{p}_i$ are fulfilled and on the other hand that the geometric details of the mesh are as good as possible preserved. These geometric features are represented by Laplace vectors (cf. Figure 2.10), which can be computed from the one-ring neighborhood of a vertex as

$$\Delta\mathbf{p}_i = w_i \cdot \mathbf{p}_i - \sum_{j \in N(i)} w_{i,j} \cdot \mathbf{p}_j$$

where we defined $N(i)$ to be the set of vertices in the one-ring neighborhood of the vertex i. This discrete *Laplace Operator* [DC76] approximates the second derivative of a scalar function, stored at each vertex of the triangular mesh, w.r.t to the spatial coordinates x, y, z. While in general the scalar function can be an arbitrary function, in this case we consider this function to be the

vertex coordinates themselves, leading to 3 scalar functions, one for each spacial dimension. When rearranging the components of the vertex positions into a matrix $[\mathbf{p}_x, \mathbf{p}_y, \mathbf{p}_z] = [\mathbf{p}_1, \ldots, \mathbf{p}_n]^T$, the Laplace operator can be stated for the whole triangle mesh. This leads to a matrix $L \in \mathbb{R}^{n \times n}$ where each row stores the sum $w_i = \sum_{j \in N(i)} w_{i,j}$ on the diagonal element and the negations of the weights $w_{i,j}$ at the columns corresponding to the respective neighbor vertices. As elaborated in [WMKG07] there is a large theory dealing with this operator and how to define the weights $w_{i,j}$. In our cases we use the well known cotangent weights

$$w_{i,j} = \frac{\cot \alpha_{i,j} + \cot \beta_{i,j}}{2}$$

to define the Laplace operator, where $\alpha_{i,j}$ and $\beta_{i,j}$ are the angles opposite to the edge connecting the vertices i and j.

With the definition of the Laplace operator, the objective to preserve geometric details means that the new vertex positions should have the same Laplace vectors as the original vertex positions, which translates to the residual function

$$\mathbf{f}_x = L \cdot \mathbf{p}_x - L \cdot \mathbf{q}_x \tag{2.35}$$

Observe that in this case we defined the residual function only for the x-component of the vertex positions, for the other components this and the following equations are analogous.

As described in Section 2.1, the minimization of the energy function of such residuals leads to a linear system of equations, since the residuals are linear in its parameters. But the resulting linear system is under-determined and does not have an unique solution, since each mesh whose new vertices have the same Laplace vertices as the input vertices, can arbitrarily translated in 3D space.

Therefore positional constraints need to be introduced. In our reconstruction pipeline we deal with two kind of constraints: Vertex constraints minimize the distance between a vertex $\mathbf{p}_i$ and an arbitrary point in 3D space (cf. middle of Figure 2.10). Assuming we have v vertex constraints we can write these constraints as a new residual function of the form

$$\mathbf{g}_x = C \cdot \mathbf{p}_x - \mathbf{v}_x \tag{2.36}$$

where $\mathbf{v}_x \in \mathbb{R}^v$ encodes all x-components of the points the vertices are attracted to and $C \in \mathbb{R}^{v \times n}$ is a matrix with $C_{i,j} = 1$ if the vertex j is attracted by the i^{th} 3D point (equations for the y and z components can be defined analogously).

The second type of constraints are face constraints. They minimize the distance between an arbitrary point in 3D space and a point located on a triangle of the mesh. As depicted in the right image of Figure 2.10, this point has barycentric coordinates (α, β, γ). Having t face constraints, we can define a third residual function as

$$\mathbf{h}_x = F \cdot \mathbf{p}_x - \mathbf{t}_x \tag{2.37}$$

where again $\mathbf{t}_x \in \mathbb{R}^t$ encodes all x-components of the points the faces are attracted to and $F \in \mathbb{R}^{t \times n}$ stores the barycentric coordinates of the surface point at those columns corresponding to the vertices adjacent to the triangle.

With both types of constraints and the residuals measuring the change of the local detail vectors, we can setup an energy function

$$E = w_v \cdot \mathbf{g}_x^T \mathbf{g}_x + w_t \cdot \mathbf{h}_x^T \mathbf{h}_x + w_L \cdot \mathbf{f}_x^T \mathbf{f}_x \tag{2.38}$$

from which we can compute the least squares solution by solving the linear system

$$(w_v \cdot C^T C + w_t \cdot F^T F + w_L \cdot L^T L) \cdot \mathbf{p}_x = w_v \cdot C^T \mathbf{v}_x + w_t \cdot F^T \mathbf{t}_x + w_L \cdot L^T L \cdot \mathbf{q}_x \tag{2.39}$$

where we introduced weights w_v, w_t and w_L to control the influence of the attractions and the flexibility of the surface.

2.6.2. As-rigid-as-possible Mesh Editing

In this thesis we use Laplace Editing when only small deformation need to be applied in order to fit a mesh to a target surface. In those scenarios we usually have a lot of soft constraints, e.g. a dense point cloud obtained from stereo reconstruction, and we need to smoothly interpolate areas where no attraction points are available. As we will see later, with a good initialization, Laplace Mesh Editing can produce a nicely interpolated surface without holes and little noise, that fits well to the surface to be reconstructed.

But in some other scenarios we only have given a few hard constraints, which need to be fulfilled exactly. Additionally the surface to be constructed is far from the input surface, such that large deformations need to be applied. When large deformations are involved, we need to take special care of local rotations

of the Laplace vectors, which makes Laplace Mesh Editing hard to use. So, for such scenarios with large deformations and only a few constraints we rather use *As-rigid-as-possible Mesh Editing* [SA07].

As before the input for this approach is a triangular mesh $M = (V, T)$, where the original vertex locations are denoted as $V = (\mathbf{q}_1, \ldots, \mathbf{q}_n)$. The aim is to compute new vertex locations $V' = (\mathbf{p}_1, \ldots, \mathbf{p}_n)$, such that on the one hand some vertices lie at specific locations, i.e. $\mathbf{p}_j = \mathbf{c}_k$ for some vertices j. On the other hand the new vertices should be distributed in 3D space such that local rotations R_i, associated to each vertex i are as small as possible. To model the later requirement, Sorkine et al. [SA07] defined an energy function $E(R_i, \mathbf{p}_i)$, that measures how the transformation of the one-ring-neighborhood deviates from a pure rotation:

$$E(R_i, \mathbf{p}_i) = \sum_{j \in N(i)} w_{i,j} ||(\mathbf{p}_i - \mathbf{p}_j) - R_i(\mathbf{q}_i - \mathbf{q}_j)||^2$$

where again $N(i)$ denotes the one-ring-neighborhood of vertex i and $w_{i,j}$ are defined as the cotangent weights [SA07] in order to make this energy formulation independent from the resolution or anisotropic element distribution of the mesh. The integration over all one-ring-neighborhoods then leads to the surface energy:

$$E(V, V') = \sum_i E(R_i, \mathbf{p}_i)$$

This energy function is highly non-linear, since the parameters to be optimized are the new vertex locations $\mathbf{p}_i$ and the rotations R_i of the one-rings. With a proper parametrization of the rotational matrices we could apply one of the optimization techniques presented in Section 2.1, but in [SA07] the authors suggest a simpler approach, which alternates the computation of new vertex positions given fixed rotations with updating the rotations given new vertex locations. The key observation to make this work, is that optimal rotations can analytically be computed independent for each one-ring from the singular value decomposition [QSS07] of the covariance matrix

$$S_i = \sum_{j \in N(i)} w_{i,j}(\mathbf{q}_i - \mathbf{q}_j)(\mathbf{p}_i - \mathbf{p}_j)^T = U_i \Sigma_i V_i^T \tag{2.40}$$

where Σ_i contains the singular values and U_i and V_i contain the left and right-singular vectors. Then the optimal rotation is simply computed as $R_i = V_i U_i^T$.

In the second step of this alternating approach the rotations are seen as constants. In what follows we restrict our considerations on the x-components of the new vertex positions, but the y and z-components can be computed analogously by solving the Laplace System

$$L\mathbf{p}_x = \mathbf{b}_x \tag{2.41}$$

which is deduced from the partial derivatives of the energy function w.r.t. the new vertex locations. Again, $L \in \mathbb{R}^{n \times n}$ is the Laplace matrix containing the cotangent weights, $\mathbf{p}_x$ is the n-dimensional vector holding the x-coordinates of the new vertex positions and $\mathbf{b}_x$ is the concatenation of the x-components of the vectors

$$\mathbf{b}_i = \sum_{j \in N(i)} \frac{w_{i,j}}{2}(R_i + R_j)(\mathbf{q}_i - \mathbf{q}_j)$$

In order to constrain a vertex to a fixed position, e.g. $\mathbf{p}_j = \mathbf{v}_k$, we remove the corresponding column and row from the Laplace matrix L and if $\mathbf{p}_j$ occurs in another row, we insert the x-component of $\mathbf{v}_k$ and bring it on the right side of the equation.

Both steps, the computation of new rotations of the one-rings (cf. Equation 2.40) and the computation of new vertex positions $\mathbf{p}_i$ (cf. Equation 2.41) are alternated in each iterations. As common for most non-linear optimization problems, this approach converges quickly when starting with a good initial solution. To obtain this, we compute a global transformation that minimizes the distance between the constraints $\mathbf{v}_k$ and the transformed corresponding vertices $\mathbf{p}_j$. As proposed by Horn [Hor87] this transformation is the composition of a rotational part $R \in \mathbb{R}^{3\times3}$, a scaling $S \in \mathbb{R}^{3\times3}$ and a translational part $\mathbf{t} \in \mathbb{R}^3$ that minimizes the energy function

$$E(R, S, \mathbf{t}) = \sum_{(j,k)} ||R \cdot S \cdot \mathbf{q}_j + \mathbf{t} - \mathbf{v}_k||^2$$

While the translational part is simply the vector between the center of gravity of both point clouds ($\mathbf{q}_j$ and $\mathbf{v}_k$) and the scaling can be obtained from the diameters of both point clouds, the rotational part is a bit more involved and requires to compute the eigenvectors of a four dimensional matrix [Hor87]. Then the rotational matrix R can be obtained from the larges Eigenvector,

witch represents the optimal rotation between both point clouds as a unit quaternion [Vic01].

The initial solution for the described iterative procedure is obtained by applying this global transformation to all vertices of the mesh M, such that $\mathbf{p}_i = R \cdot S \cdot \mathbf{q}_j + \mathbf{t}$. In practice we only use a small number of four iterations to find the final vertex positions of the deformed mesh M'.

2.6.3. Deformation Transfer

Deformation transfer for triangle meshes [SP04, BSPG06] is the key technique to map the dynamic movements from one avatar face to another. The technique represents these dynamic movements as local deformations per triangle, called the *deformation gradients*. Under the assumption, that all facial models are in full correspondence, meaning their polygonal surface meshes have the same topology and vertices are uniquely associated to specific facial regions like the nose or the mouth, we use deformation gradients to decouple the dynamic movements from the individual shape of the face. Our dynamic model (cf. Chapter 7), learned from a database of facial movements, is able to generate new deformation gradients by adjusting a few parameters, which can in turn be mapped to any new facial shape to animate it.

In this section we shortly recall the principals of deformation transfer for triangle meshes and how deformations can be mapped to a new shape. Assume a source triangle mesh $\mathcal{S} = (V, T)$ is given, with $V = \{\mathbf{q}_1, \ldots, \mathbf{q}_n\}$ being the set of vertices and $T = \{t_1, \ldots, t_m\}$ the set of triangles. Further assume this mesh was deformed into another mesh $\mathcal{S} \rightarrow \mathcal{S}'$ where new point positions in $\mathcal{S}'$ are denoted by $\{\mathbf{q}'_1, \ldots, \mathbf{q}'_n\}$. As described in [BSPG06] we can compute a deformation gradient $\mathbf{S}_t \in \mathbb{R}^{3\times3}$ for each triangle t that maps a point from the undeformed to the deformed state. This deformation gradient is given by

$$\mathbf{S}_t = (\mathbf{q}'_1 - \mathbf{q}'_3, \mathbf{q}'_2 - \mathbf{q}'_3, \mathbf{n}') \cdot (\mathbf{q}_1 - \mathbf{q}_3, \mathbf{q}_2 - \mathbf{q}_3, \mathbf{n})^{-1}$$

where $\mathbf{n}$, $\mathbf{n}'$ and $\mathbf{q}_i, \mathbf{q}'_i$ are the normals and vertex positions of an undeformed and a deformed triangle t.

The idea is to find a deformation for a target mesh $\mathcal{T} \to \mathcal{T}'$ such that the deformation gradients $\mathbf{T}_t \in \mathbb{R}^{3\times 3}$ of the target mesh match those of the source mesh:

$$\mathbf{T}_t = (\mathbf{p}'_1 - \mathbf{p}'_3, \mathbf{p}'_2 - \mathbf{p}'_3, \mathbf{n}') \cdot (\mathbf{p}_1 - \mathbf{p}_3, \mathbf{p}_2 - \mathbf{p}_3, \mathbf{n})^{-1} = \mathbf{S}_t$$

Here we denote the vertices of $\mathcal{T}$ and $\mathcal{T}'$ as $\{\mathbf{p}_1, \ldots, \mathbf{p}_n\}$ and $\{\mathbf{p}'_1, \ldots, \mathbf{p}'_n\}$. Note that the indices of the triangles do not change since both meshes are in full correspondence. Equivalent to this we can also require that the transposed of these matrices should be the same:

$$\mathbf{T}_t^T = \mathbf{S}_t^T$$

This has the advantage that we can find an expression for $\mathbf{T}_t$ which is linear in the new point positions $\mathbf{p}'_i$:

$$\tilde{\mathbf{T}}_t^T = \underbrace{\left((\mathbf{p}_1 - \mathbf{p}_3, \mathbf{p}_2 - \mathbf{p}_3, \mathbf{n})^{-1}\right)^T \cdot \begin{pmatrix} 1 & 0 & -1 \\ 0 & 1 & -1 \\ 0 & 0 & 0 \end{pmatrix}}_{=:\mathbf{G}_t} \cdot \begin{pmatrix} \mathbf{p'_1}^T \\ \mathbf{p'_2}^T \\ \mathbf{p'_3}^T \end{pmatrix}$$

where $\mathbf{G}_t$ is the constant gradient matrix of the coordinate function of a triangle t of the target mesh $\mathcal{T}$.

In order to compute the deformation transfer for all triangles we want to find new vertex positions $\mathbf{p}'_i$ such the all deformation gradients of the target mesh are equal to the deformation gradients of the source mesh. This is expressed by the global system

$$\begin{pmatrix} \mathbf{G}_1 \\ \vdots \\ \mathbf{G}_m \end{pmatrix} \cdot \begin{pmatrix} \mathbf{p'_1}^T \\ \vdots \\ \mathbf{p'_n}^T \end{pmatrix} = \mathbf{G} \begin{pmatrix} \mathbf{p'_1}^T \\ \vdots \\ \mathbf{p'_n}^T \end{pmatrix} = \mathbf{S} = \begin{pmatrix} \mathbf{S}_1{}^T \\ \vdots \\ \mathbf{S}_m{}^T \end{pmatrix}$$

Where $\mathbf{G} \in \mathbb{R}^{3m\times n}$ is the global gradient matrix computed from $\mathcal{T}$ and $\mathbf{S} \in \mathbb{R}^{3m\times 3}$ is a global matrix computed from the deformation $\mathcal{S} \to S'$. Since this system is over determined we solve it in the least squares sense

$$\mathbf{G}^T\mathbf{G} \begin{pmatrix} \mathbf{p'_1}^T \\ \vdots \\ \mathbf{p'_n}^T \end{pmatrix} = \mathbf{G}^T\mathbf{S}$$

This system weights the deformations per triangle in a uniform way which may lead to unwanted distortions. To avoid this, the deformation gradients are weighted by the area of the triangle which finally yields the Poisson equation describing the deformation transfer:

$$\mathbf{G}^T\mathbf{D}\mathbf{G}\begin{pmatrix} \mathbf{p}_1'^T \\ \vdots \\ \mathbf{p}_n'^T \end{pmatrix} = \mathbf{G}^T\mathbf{D}\mathbf{S} \tag{2.42}$$

where $\mathbf{D} \in \mathbb{R}^{3m\times 3m}$ is the diagonal area matrix computed from all triangles in $\mathcal{T}$. Observe that $\mathbf{G}^T\mathbf{D}\mathbf{G}$ is equal to the standard Laplace matrix of the mesh $\mathcal{T}$ using the cotangent weights and $\mathbf{G}^T\mathbf{D}$ represents the divergence operator [BSPG06].

3. Related work in Facial Performance Capture

Our systems to track facial movements from stereo images or single images is related to previous work in the areas of facial modeling, face reconstruction, tracking and animation.

Artistic animation design. A common technique to generate facial movements for movies and computer games is Free Form Deformation (FFD). FFD provides a framework which allows artists to drag vertices of a cage to intuitively deform the space inside the cage and thus the underlying geometry. Sophisticated methods, e.g., introduced in [JMD*07, LLCO08, BCWG09], compute mappings in 3D space which do not induce large distortions (such as volumetric shrinking). Using such tools, artists can be very creative when producing animation sequences. Such hand-made animations are usually applied for cartoon like movies and often appear awkward if they are meant to be realistic, since the degree of realism is not sufficient to leave the problematic region of the uncanny valley [Mor70].

Physically based facial animation. One way to simulate more realistic movements of a human face is to use physically based methods, which usually rebuild some anatomical features of the human head with the aim of mimicking natural movements. Waters [Wat87] simulates facial muscle contractions by abstracting the facial action units originally introduced by Ekman and Friesen [EF78]. However, this work only uses a few muscles to reproduce basic human emotions. Similarly, Lee et al. [LTW93] build an anatomically accurate physically based head model with tissue, skull and synthetic muscles, which are used to deform the tissue to produce facial expressions. Kähler et al. [KHYS02] use a similar model to perform real-time deformations based on anthropometrically meaningful landmarks. Their method is also capable of simulating aging. Sifakis et

al. [SNF05] uses finite element methods to deform the synthetic tissues around a skull model.

The major problem with all biomechanical models is that it is quite difficult to build them correctly, because our anatomical knowledge about human skin, muscles, and bone structures is still limited. Thus, models sometimes require extensive tuning to produce a realistic output. Our animation technique is purely data driven and does not require this special parameter tuning.

Marker-based methods. Many motion capture systems [CBMK*06, SNF05, BBA*07, WL90] use special cameras to track the movements of 3D marker points placed on the actors face. Sifakis et al. deduce muscle activities from such sparse motion data obtained with such systems and feed the resulting forces in their tissue model to produce anatomically plausible muscle movements [SNF05]. Curio et al. computed blendshape weights from the motion capture data to drive a morphable shape model [CBK*06]. Using such sparse sets of markers, wrinkles and fine details within the facial movements disappear. Bickel et at. suggest a thin shell based mesh deformation approach to simulate the appearance of wrinkles in a physically plausible manner [BBA*07]. All systems use special camera systems to track the 3D motion of marker points, while our system only needs a simple video as input.

Besides the sparsity of the produced data, in many applications as, e.g., psychological studies, the original videos should appear as natural as possible, such that markers are forbidden to use. With increased computational power, dense reconstruction and tracking methods have moved into focus and can roughly be divided into active sensing and passive sensing systems.

Active sensing An early active system was proposed by Zhang et al. [ZSCS04]. They reformulated space-time stereo as a global optimization problem to compute a time varying disparity map between two image sequences obtained by a structured light system. Then they fit a template mesh to the time varying depth maps and use optical flow conditions to maintain temporal correspondences. Since we use pure video data instead of structured light scanners we deal with a different setup, which also works at high frame rates, usually not

achievable with active systems. Ma et al. use polarized light emitted from multiple directions (lightstage) to separately measure a diffuse and a specular normal map [MHP*07]. In [ARL*09] this technique was further improved to compute high quality renderings of the facial performance of an actor in an offline process. In [MJC*08] Ma et al. combine active sensing with sparse motion capture data. In their setup they use structured light scanning and photometric stereo to capture wrinkles and fine facial features, while motion capture markers, which make this also a marker based method, are used to track large scale deformations and to establish correspondences between faces. In a training phase polynomial displacement maps are computed to represent medium-frequency facial deformations and high-frequency facial details and are used in a synthesis phase to produce new animation sequences. Since our marker-less capture systems (cf. Chapter 5 and 6) maintains vertex correspondences between faces and through time we can use a rather simple approach, like the one proposed by Botsch et al. [BSPG06], to transfer (blended) deformations to other faces.

Other active sensing methods were proposed by Hernández et al. [HVB*07] which use multispectral photometric stereo to compute a dense normal field from untextured surfaces. Weise et al. [WLG07] use active illumination based on phase-shift to reconstruct surfaces at high frame rates. A drawback of both approaches is that they are unable to maintain correspondences between vertices in time. In [WLVP09] Weise et al. improved their system to be able to track a generic face template at a rather high temporal resolution of 30 frames per second. The data produced by this system were used to create an actor's face model in an offline process which enables to track and transfer facial movements of this actor in real time to other faces. Similar to their approach, we consider the deformation transfer [BSPG06] as a key technique and introduce a statistical expression model by performing a principal component analysis on the time varying deformations of the template's triangles.

In [WBLP11] Weise et al. proposed a real-time facial performance capture system, which requires as input depth maps obtained from Microsofts Kinect sensor [Zha12]. This is probably the most related approach to the system presented in Chapter 7. The main difference is, that our system is purely image based, such that it works at higher frame rates. Additionally we propose a new,

general motion model for human faces which decouples the facial movements from the individual facial shapes. The most recent active sensing system to capture facial performance was proposed by Li et al. [LYYB13]. Before the tracking phase, the system first reconstructs the neutral face by fitting a mesh to aggregated depth maps taken from a Kinect sensor. To obtain a generic blendshape model [WBLP11], the neutral shape is then generically deformed using deformation transfer (cf. Section 2.6.3) to mimic facial expressions inspired by the facial action coding system [EF78]. Just fitting this blendshape model to depth maps and detected facial feature points, would not lead to an accurate tracking of the facial performance. Therefore, the authors suggest to perform a second fitting stage where they find shape parameters of an adaptive PCA model to reproduce the observed input data. Their main contribution is an EM-based [Bil98] learning approach, that incrementally incorporates individual facial details in this PCA model during the tracking phase. In their results they demonstrate the effectiveness of this one-the-fly adaption of the PCA model to decrease the fitting error after time.

Since we do not use active sensing methods [ZSCS04, HVB*07, WLG07, MJC*08, WLVP09, WBLP11, LYYB13], which in general need special hardware to project light patterns or colors onto an object, we do not distract the actor. We also get along with a rather inexpensive, simple camera setup working at high framerates.

Passive sensing In comparison to active sensing methods passive methods usually reconstruct the facial movements from pure video data. The main advantage might be, that the actor is not distracted by flickering lights produced by an active system and that videos can be taken at higher framerates. Since facial movements, such as micro-expressions [RBQ14], often are very quick, this is clearly worth to consider. A famous passive data driven approach was suggested by Blanz and Vetter [BV99]. They learn shape and texture parameters for a morphable model by performing a principal component analysis (PCA) on a set of laser scanned human heads. In a pure image based approach, they optimize these parameters to extract the static geometry and the texture of a human head from a photo. In order to decouple identities, expressions and

visemes Vlasic et al. [VBPP05] use multilinear models. They perform a statistical analysis on the captured data to obtain a multilinear face model. With this model they were able to track facial movements from a monocular video and transfer moods and articulation to another video. Dellepiane et al. [DPT*08] deform a dummy head to reconstruct the shapes of human heads from images and use them for binaural rendering. Active Appearance Models (AAMs), as in [OOB03, KBM*05], are used to track motion through (multiview) image sequences. As with all morphable models, though, the reconstructions are always restricted to the low-dimensional space spanned by the parametric model, while our reconstruction methods (cf. Chapter 5 and 6) produce results not restricted to such parametric spaces.

Vedula et al. introduced the term *dense scene flow* in [VBR*99], which was further improved by Li and Sclaroff [LS08]. In their pure video based approach they reformulate the optical flow problem, find corresponding pixels in time, and use disparity to find correspondences between different views. The extraction of geometric information which could be used for simple visibility tests was not considered. In [FP09] Furukawa extended a previous approach to track vertices of a mesh reconstructed in the first image of a video stream. In order to ease the 3D stereo reconstruction and to produce compelling results they put additional paint on the faces. We designed our system such that we always have good initial solutions for the 3D reconstruction, thus we do not need to artificially texture human faces. Another difference is that we use a predefined template with fixed topology, which gives us inter-subject correspondences and simplifies deformation transfer and blending. Borshukov et al. [BPL*05] propose a system that is similar to ours. A scanned model of a neutral expression of an actor is tracked in time. They use optical flow to ensure temporal correspondences and use 3D stereo to triangulate 3D positions of the models vertices. Since they are using a very expensive camera setup ($>$ 100.000 \$) and correct tracking errors in 2D and 3D manually the results look very impressive. Furukawa and Ponce use a multiview stereo system and define filters for the local neighborhood of a vertex to smoothly track the vertices of a triangular mesh [FP09].

There are other expensive commercial solutions to capture facial movements from pure video data like DI3D$^{\text{TM}}$ from Digital Imaging. Recently Bradley et

al. [BHPS10] designed a passive, video based capture system which does not need special markers. They use 7 stereo pairs of high resolution cameras to reconstruct single patches, which are merged into a 3D point cloud. An optical flow based tracking which involves minimal user interaction is used to establish temporal correspondences. With this quite expensive system they are able to create textured animated faces in a high resolution. Our method is low-cost and the tracking part involves no manual interaction. In addition we decided to use a mesh based 2D tracking, which allows to track, e.g., upper and lower lips independently. In general a simple implementation of optical flow could smooth out those contradictory up and down motions.

Among passive sensing there is a special field in computer vision, where 3D facial movements are infered from a single video. Such systems usually depend on a priori knowledge about the facial shape. Marker based methods [WL90, LO05] produce quite robust motion parameters used to deform a morphable model. In [XBMK04] Xiao et al. use a non-rigid-structure-from-motion approach to augment a 2D Active Appearance Model (AAM) [CET01, MB04] with additional 3D shape modes in order to track 3D facial features. Similar to [BV99, LRF93] and [DM96] Pighin et al. build a 3D morphable model and used it to track facial expressions in an analysis by synthesis approach [PSS99]. Chuang and Bregler present a framework to manually model facial animations assisted by the shape and motion information of a large database [CB02]. Their system involves an expensive search in the database and does not run in real-time. Chai et al. track facial features to extract 2D motion control signals, which are mapped to high quality 3D facial expressions using a preprocessed database [CXH03].

Most recently Cao et al. [CWLZ13, CHZ14] proposed systems to capture facial movements from videos taken by a single camera. The system presented in [CWLZ13] required an individual training phase in which the user performs a set of facial expressions in different poses. From automatically detected 2D landmarks the system then automatically reconstructs user specific blendshapes and computes a linear combination of the blendshapes to reproduce the 3D facial shape for each image obtained during this training phase. Since the reconstruction of 3D shapes from single images is in general an ill-conditioned problem, they employ a huge database of facial expressions [CWZ*14] to recover

the user specific blendshapes. Then, image data and reconstructed facial shapes are converted into a training set from which they learn a regression model that is able to recover the 3D facial shape from image intensities sampled around facial features. Similar to our approach in Chapter 7 they employ an animation prior to stabilize the reconstruction result. Since this system had the drawback that an individual model had to be trained, Cao et al. [CHZ14] proposed a similar system, which learned a similar regression model from an existing database of annotated facial images.

The biggest difference to our approach presented in Chapter 7 is that all systems model facial expressions as some sort of shape combination. Instead of this we use a deformation model extracted from the motion data itself (cf. Chapter 7) to decouple the shape of a face from its dynamic motion and thereby reduce the number of (user independent) parameters to be optimized. For a more complete overview about facial performance capture see [PL06]. Specially for the data-driven facial animation we recommend the book [DN].

Part I.

Static Faces

4. Facial Feature Localization

In this Section we propose a new approach to detect facial features in images showing a static facial expression. This detector can provide useful clues about the facial pose, the shown facial expression or the identity of the captured person. Besides these applications, in the context of this thesis the detector is of interest when roughly reconstructing a neutral face: after detecting facial features in images, simultaneously captured by a multi-camera rig, we can triangulate these features in 3D an fit a *face template* to these feature points (cf. Chapter 5). In what follows we will first present related approaches to detect facial features in Section 4.1. Since our approach is based on a spacific facial feature detector [KSYY10], we will recap this approach in Section 4.2 before we present the overview and contributions of our system. Section 4.3 describes the generation of a compact codebook, which is used in Section 4.4 to infer the positions of facial features. We evaluate the performance of the presented system in Section 4.5. Parts of this chapter previously appeared in [CSK15].

4.1. Related Work

Beyond detection and tracking of facial regions, several methods for finding facial feature points (center of the eyes, tip of the nose or corners of the mouth) have been proposed, since more and more accuracy is required to recognize, e.g., the user's expressions. Such algorithms can be categorized into two different methodologies, namely iterative parameter optimization and non-parametric localization.

Iterative parameter optimization methods [CTCG95, MN08, MB04, LWC*09, CO10, BV03] define the facial feature localization as a parameter optimization problem and solve it by iteratively finding least squares solution for a predefined error metric. Active shape models (ASM) [CTCG95, MN08], active ap-

pearance models (AAM) [MB04, LWC*09, CO10], and 3D morphable models (3DMM) [BV03] are statistical models, which are widely used for facial feature localization. Such approaches find optimal parameters representing shape and texture. Although they show good results with some constraints on parameters, they need to set an initial position near the optimum and require several iterations to minimize an energy function. Especially, the calculation of a large hessian matrix is required to fit a 3DMM and this causes high computational complexity.

As a non-parametric localization method, Chen et al. [CZLZ04] calculate the positions of facial feature points using pixel likelihood maps. They divide the facial region into several segments corresponding to each feature point. Then, likelihood maps for each feature point related to the image segments are calculated using the classifier trained by a boosting strategy. This algorithm shows very accurate localization results and fast performance with illumination and scale invariance. However, the method needs a large bundle of non-face training images and cannot deal with occlusion. Wiskott et al. [WFKvdM97] define an elastic bunch graph to recognize human faces. In a refinement procedure the nodes of an image graph are shifted across a query image such that wavelet responses at these nodes match examples from the bunch graph. While this approach is able to handle pose variations, it needs some manual input and is rather time consuming. Bourdevet al. [BMBM10] use a parts model containing poslets to detect object parts and infer relative positions of these parts. While in general this works on a broad variety of objects, it is rather difficult to precisely locate facial features. Leibe et al. [LLS04] propose to learn a codebook for a specific object class, which contains descriptors based on Harris interest points and a shape model represented as a spatial probability distribution for the positions of the object parts. Besides using a complicated voting procedure this method is more suited for a categorization task rather than precise feature localization. Kozaya et al. [KSTN08, KSYY10] propose a codebook-based approach and experimentally showed that their algorithm is more accurate than the state-of-the-art approach ASM (STASM) [MN08]. For this algorithm an exhaustive search on a large database as well as the extraction of many signatures at prospective feature positions yields to problems regarding time and memory consumption.

4.2. Fundamental Approach to Detect Facial Features

Our technique for facial feature localization is based on the approach described in [KSYY10]. In this approach, features are not detected directly but rather by a voting procedure: First, a simple face detector is used to roughly identify the location of a face in an image. Then, the face region (typically a square) is split into a set of overlapping query segments. For each query segment a signature (histogram of oriented gradients (HOG) [DT05]) is computed and matched with a database of segments. This database contains a large number of segments H_s associated with ground truth offset vectors $\mathbf{v}_{s,k}$ which result from a manually labeled training data set of faces. For each segment H_s in the database, the corresponding offset vector $\mathbf{v}_{s,k}$ is the vector from the center of the segment to the k^{th} feature point.

Let $G_1, \ldots, G_n$ be the set of overlapping query segments covering the face region in the query image and let $\mathbf{c}_1, \ldots, \mathbf{c}_n$ be the corresponding fragment centers. Moreover let $H_1, \ldots, H_n$ be the respective best matches from the database and $\mathbf{v}_{1,k}, \ldots, \mathbf{v}_{n,k}$ the associated offset vectors indicating the relative position of the k^{th} feature. Then each segment G_i predicts the expected feature location to lie on the line $\mathbf{c}_i + \lambda \cdot \mathbf{v}_{i,k}$. By considering all votes, we can estimate the feature position to be the point with minimal distance to all lines. To increase accuracy, this procedure weights each line according to the qualtiy of the match. The computation of these weights is rather expensive since it involves the extraction of signatures at the numerous prospective feature positions $\mathbf{c}_i + \mathbf{v}_{i,k}$. When it comes to occlusion this quality measurement further has the drawback that the signature extracted at the prospective feature location is very different from the one in the database, although the offset vector might be very precise.

While in general this facial feature localization approach is conceptually simple and very effective in practice, it has a number of additional drawbacks, e.g., the memory requirements for the database of signatures are quite strong due to the HOG-descriptors that are typically used. Moreover, empirically it has been observed that covering a face region with 81 overlapping query fragments leads to good results in practice which implies considerable computation costs

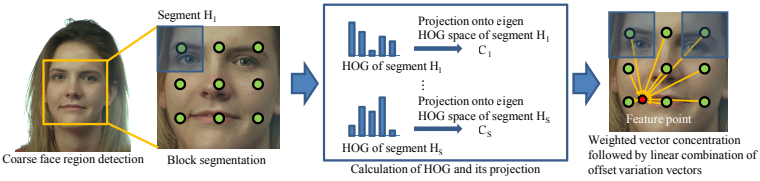


Figure 4.1.: The proposed facial feature localization method. The yellow rectangle on the first image represents the coarse face region extracted by the Viola-Jones face detector. The green spots near eyebrows are centers of segments. The red spot on the last image is a feature point, left corner of lip. The orange arrows are the offset vectors from the segment centers to the feature point. Images taken from [PMRR00].

for the signature matches even if hierarchical data structures [AMN*94] are employed.

Since our eventual goal is to run the facial feature localization on mobile phones in realtime, we have to be more restrictive with the memory and compute resources. This is why we are applying a statistical data reduction scheme to the database of fragments. Through a principal component analysis, we can extract the major variation modes within the set of fragments leading to a significantly smaller set of *eigen-descriptors* faithfully representing the entire set. Accordingly we derive an *offset variation vector* for each eigen-descriptor. For the feature localization the descriptor matching steps are then replaced by a simple projection of the query descriptor into the basis of eigen-descriptors which yield the feature location prediction vote. Since our method employs statistical characteristics of the training set, we do not need to extract signatures at the prospective feature positions to compute the quality of the match, which saves a lot of computation time. In experiments we show that we are still able to handle occlusions efficiently and robustly.

Overview and Contributions Figure 4.1 shows the proposed framework to localize facial features. First, the face region within the input image is roughly detected by the Viola-Jones face detector [VJ01]. Then, the image inside the detected region is re-sized to a prescribed size and segmented into a number

of regularly positioned blocks. For each block a HOG descriptor is computed and projected into the eigen HOG (EHOG) space of that segment, which is precalculated in the training phase. The projection coefficients are then used to linearly combine offset variation vectors (OVV), which are also precalculated in the training phase. This generates vectors pointing from the segments center to the prospective feature point positions. We can robustly decide whether a segment is occluded by simply checking if the projection coefficients are similar to those observed in the training data. Finally, the position of each feature point is calculated by WVC [KSYY10].

The contributions of this work are:

1. Low memory consumption: We use principal component analysis (PCA) as a data reduction technique to compute a compact codebook. This allows us to use the energy preserving level of PCA to control the tradeoff between memory consumption and accuracy.

2. Fast localization: Offset vectors are simple linear combinations where coefficients are determined by projection instead an involved search in a large database.

3. Fast validity checking: Occlusion detection and confidence checking for offset vectors is done in EHOG Space.

4. High accuracy: Experiments show that we achieve better accuracy than state-of-the-art approaches.

5. Occlusion handling: Our new method to handle occlusions exploits the distribution of the input HOGs. In experiments we demonstrate the robustness.

4.3. A compact codebook

In the conceptional description of this approach (cf. Section 4.2) we stated that an image segment is characterized by a descriptor, the HOG descriptor. In what follows we describe in detail how this descriptor is assembled and how we generate a compact codebook containing information about the descriptors and the locations of facial features.

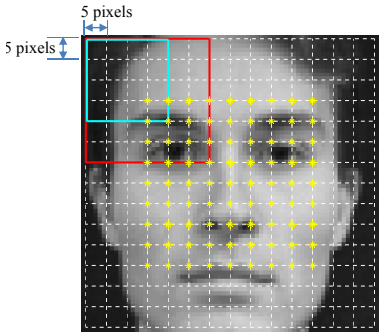


Figure 4.2.: Canonical segmentation of a face region. The yellow asterisks are the centers of the overlapping segments (red), white dashed rectangles are cells (5x5 pixel), cyan rectangles are blocks (4x4 cells) and red rectangles are the segments (3x3 overlapping blocks).

Histograms of oriented gradients HOG is a well-known feature for its illumination invariance and high distinctiveness [DT05]. Similar to [KSYY10] we detect the facial region in each training and input image using the Viola-Jones face detector [VJ01]. We resize the detected facial region to 72x72 pixel and place the centers of partial overlapping image segments on a regular 9x9 grid, which defines a set of 81 image segments. For each image segment s we compute a HOG descriptor.

A HOG descriptor can be computed in different ways, we use the procedure experimentally validated in [KSYY10, DT05]. First we compute gradients g_p for each pixel p in the inner 70x70 image region. Note that therefore the additional pixel rows and columns at the borders are needed. The gradients can be represented as $g_p = (\alpha_p, r_p)$, where $\alpha_p \in [0, \ldots, 180]$ encodes the unoriented direction and r_p the magnitude of the gradient. As in [DT05] we discretize the orientations into 3 bins:

$$bin(\alpha_p) = \begin{cases} 1, & 0 < \alpha_p \leq 60 \\ 2, & 60 < \alpha_p \leq 120 \\ 3, & 120 < \alpha_p \leq 180 \end{cases}$$

Then the inner region is divided into *cells* of 5x5 pixel (see Figure 4.2) and for each cell c containing a set P_{cell_c} of 25 pixel, we compute a histogram of oriented gradients C-HOG$_c$ by adding the magnitude of each pixels gradient to the bin corresponding to its orientation:

$$\text{C-HOG}_c = \left[\sum_{p \in P_{c,1}} r_p, \sum_{p \in P_{c,2}} r_p, \sum_{p \in P_{c,3}} r_p \right] \in \mathbb{R}^3$$

Here the set $P_{c,i} = \{p \in P_{cell_c} | bin(\alpha_p) = i\}$ is the set of pixel from cell c whose gradient orientation falls into bin i. A *block* is defined to be a set of 4x4 cells with cell indices $\{c_1, \ldots, c_{16}\}$ (see Figure 4.2). The histogram B-HOG$_b$ of block b is constructed by concatenating the histograms of its cells:

$$\text{B-HOG}_b = [\text{C-HOG}_{c_1}, \ldots, \text{C-HOG}_{c_{16}}] \in \mathbb{R}^{48}$$

Then this histogram is normalized. For convenience we refer to the normalized block histogram as B-HOG$_b$. An image *segment* is defined to be a set of 3x3 blocks with block indices $\{b_1, \ldots, b_9\}$. One segment covers an image region of 30x30 pixels such that we have one cell overlap between neighboring blocks (see Figure 4.2). The final HOG descriptor for the segment s is then computed by concatenating the histograms of the overlapping blocks

$$\text{HOG}_s = [\text{B-HOG}_{b_1}, \ldots, \text{B-HOG}_{b_9}] \in \mathbb{R}^{432}$$

which is again normalized.

Using HOG patterns themselves requires a huge amount of memory because we need to store all the high dimensional patterns, i.e., HOG patterns of training segments, HOG patterns of feature points, and their corresponding offset vectors. Since such extensive memory consumption is far too much for a facial feature localization application, we propose to use PCA (cf. Section 2.2.1) as a data compression technique to reduce the codebook size.

Generating the compact codebook. In order to compute the compact codebook, we collect a set of histograms of oriented gradients from n training images. For each training image I_i and segment H_s, we extract a HOG descriptor $\hat{\mathbf{h}}_s^i$. We assume that facial features have been assigned such that we can store for

each image I_i a set of offset vectors $\hat{\mathbf{v}}_{s,k}^i$ which point from the center of segment s to the k^{th} feature point. Here k is the index of one of the K manually annotated feature points. From the histogram data, we construct for each segment H_s a matrix $\hat{M}_s = \left[\hat{\mathbf{h}}_s^1, \ldots, \hat{\mathbf{h}}_s^n\right]$ where the columns contain the histograms $\hat{\mathbf{h}}_s^i$. Running PCA on such a matrix extracts a mean histogram $\mathbf{h}_s^{\text{avg}}$ and eigen histograms $\mathbf{h}_s^1, \ldots, \mathbf{h}_s^D$, with $D = n - 1$. We assume that the eigen histograms are sorted according to their significance, i.e. $\lambda_s^1 > \cdots > \lambda_s^D$, where λ_s^i is the eigenvalue of the eigenvector $\mathbf{h}_s^i$. To reduce dimensionality only the P_s most significant eigenvectors are stored in the column matrix $M_s = \left[\mathbf{h}_s^{\text{avg}}, \mathbf{h}_s^1, \ldots, \mathbf{h}_s^{P_s}\right]$. Here P_s is chosen such that at least a fixed fraction f (e.g. 0.99) of variance (energy) is preserved:

$$f \leq \frac{\sum_{d=1}^{P_s} \lambda_s^d}{\sum_{d=1}^{D} \lambda_s^d} \tag{4.1}$$

After PCA (Figure 4.3), EHOGs can be represented as a linear combination of HOGs

$$\hat{M}_s \cdot C_s = M_s, \tag{4.2}$$

The matrix $C_s \in \mathbb{R}^{n \times (P_s+1)}$ is a coefficient matrix. Since the dimension of the histograms is usually much smaller than the number n of training images, C_s is computed as the least norm solution:

$$C_s = \hat{M}_s^T \cdot \left(\hat{M}_s \cdot \hat{M}_s^T\right)^{-1} \cdot M_s. \tag{4.3}$$

The coefficient matrix linearly relates the histogram data to the eigen histograms.

For each segment s we have a set of offset vectors pointing from the segment center to a specific feature point k. Since we have n training images we can build a data matrix for each feature point k and segment s:

$$\hat{V}_{s,k} = \left[\hat{\mathbf{v}}_{s,k}^1, \ldots, \hat{\mathbf{v}}_{s,k}^n\right] \in \mathbb{R}^{2 \times n}$$

In order to compute offset variation vectors (OVVs) from these data matrices we do not perform a PCA. Instead we deduce the OVVs from $\hat{V}_{s,k}$ by using the same linear mapping that relates $\hat{M}_s$ and M_s:

$$V_{s,k} = \hat{V}_{s,k} \cdot C_s, \tag{4.4}$$

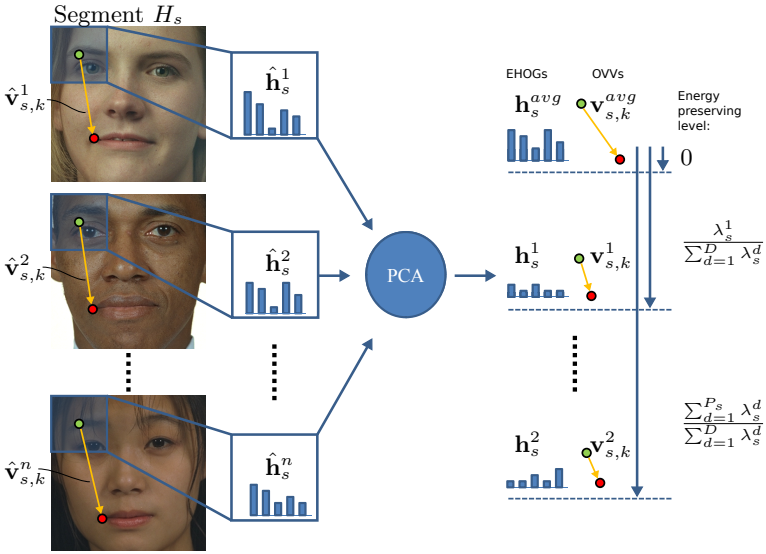


Figure 4.3.: EHOGs and OVVs: PCA is used to extract EHOGs. For each EHOG we can compute an OVV. The Energy preserving level is increased when adding more and more EHOGs and OVVs to the codebook. Images taken from [PMRR00].

where the offset variation vectors are stored in the matrix

$$V_{s,k} = \left[\mathbf{v}^{\text{avg}}_{s,k}, \mathbf{v}^1_{s,k}, \ldots, \mathbf{v}^{P_s}_{s,k}\right] \in \mathbb{R}^{2\times(P_s+1)}.$$

This allows us to reproduce a pair of a histogram $\hat{h}_s$ and a variation vector $\hat{v}_{s,k}$, which can be observed in the training data, by linearly combine EHOGs and OVVs using the *same* coefficients.

4.4. Localizing facial features

In the following we describe how we compute the position of facial features in a query image. As during the training, we detect the facial region using

the Viola-Jones face detector [VJ01] and resize this region to 72x72 pixel. As pointed out in Section 4.3 we regularly position the centers of 9x9 segments in the re-sampled face region and compute histograms of oriented gradients $\mathbf{g}_s$ for each query segment G_s. Each HOG can be approximated as a linear combination of EHOGs where the coefficients are obtained by a simple and fast projection step. We use these coefficients to compute offset vectors to all facial features as a linear combination of OVVs and use the weighted vector concentration approach (WVC) from [KSYY10] to infer the feature positions. Note that one can also use a simple voting procedure, which computes the feature position as a weighted sum of offset vectors, but we choose to use WVC to produce comparable results.

4.4.1. Estimating offset vectors

In our method we do not rely on any time consuming search algorithm like ANNS [AMN*94] to compute the best matching HOG from the codebook. Assume we want to compute the k^{th} offset vector originating from segment G_s with histogram $\mathbf{g}_s$. We can compute coefficients $c_s^1, \ldots, c_s^{P_s}$ such that

$$\mathbf{g}_s \approx \mathbf{h}_s^{\text{avg}} + \sum_{i=1}^{P_s} c_s^i \mathbf{h}_s^i. \tag{4.5}$$

The coefficients are computed by projection as $c_s^i = (\mathbf{g}_s - \mathbf{h}_s^{\text{avg}})^T \cdot \mathbf{h}_s^i$.

In order to compute the offset vector $\mathbf{v}_{s,k}$ pointing to the k^{th} feature we linearly combine OVVs:

$$\mathbf{v}_{s,k} = \mathbf{v}_{s,k}^{\text{avg}} + \sum_{i=1}^{P_s} c_s^i \mathbf{v}_{s,k}^i. \tag{4.6}$$

This process is demonstrated in Figure 4.4.

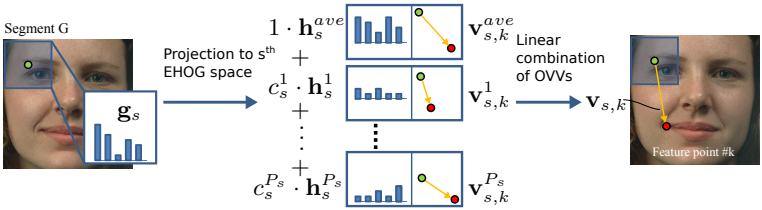


Figure 4.4.: Calculating the offset vector from one segment to a specific facial feature point. Images taken from [PMRR00].

4.4.2. Computing the position of a feature

After computing the offset vectors, WVC [KSYY10] is used to calculate the positions of feature points such that they have the least sum of weighted squared distances from the lines:

$$\arg\min_{x,y} \sum_{s=1}^{S} w_s^2 \|a_s x + b_s y + c_s\|^2, \tag{4.7}$$

where (x, y) is the coordinate of the k^{th} facial feature point, w_s is the weight of segment G_s, S is the total number of facial segments and (a_s, b_s, c_s) are the coefficient of the linear equation $a_s x + b_s y + c_s = 0$ with $a_s^2 + b_s^2 = 1$ representing the line with directional vector $\frac{v_{s,k}}{||v_{s,k}||}$ and going through the center of G_s. We define the weight for the offset vector of segment G_s as the dot product of $\mathbf{g}_s$ and its approximation in the s^{th} EHOG space:

$$w_s = \mathbf{g}_s^T \cdot \left(\mathbf{h}_s^{\text{avg}} + \sum_{i=1}^{P_s} c_s^i \mathbf{h}_s^i \right). \tag{4.8}$$

The dot product of the HOG extracted at the prospective feature position with the HOG at the feature position observed in the training data is a good indicator for the correctness of the offset vector, if the face is not occluded [KSYY10]. However, if the input face image is partially occluded, the HOG extracted at the occluded prospective feature position is very different from the trained one even thought the HOG patterns of the corresponding image segments are very

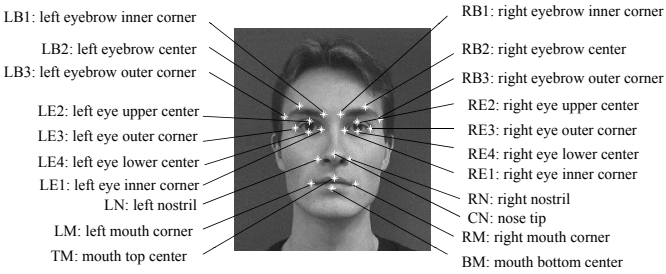


Figure 4.5.: Definition of the 21 facial feature points.

similar to each other. Note that we do not extract a HOG at positions predicted by each offset vector to compute w_s, which is computationally quite expensive since we need to extract KxS HOGs. We rather base our quality measurement on the Mahalanobis distance which measures the similarity of unknown samples to known ones. We compare the projection coefficients to the eigenvalues $\lambda_s^1, \ldots, \lambda_s^{P_s}$. If one of the coefficients exceeds the eigenvalue by a factor of 2.5, we infer that we are far away from the average HOG observed for the corresponding segment. In such case we explicitly set $w_s = 0$:

$$w_s = \begin{cases} w_s & \text{if } |c_s^i| < 2.5\lambda_s^i \\ 0 & \text{otherwise} \end{cases} \tag{4.9}$$

As in [RL87] we use least median of squares (LMedS) to compute the preliminary feature position and refine this position using adapted weights w_s. Therefore we re-weight each line according to its distance d to the preliminary feature position:

$$w_s \leftarrow w_s \cdot e^{-\frac{d^2}{L^2}}, \tag{4.10}$$

where L is the length of the diagonal of the face region.

4.5. Evaluation

In this section we compare our method using EHOGs and OVVs to the original approach [KSYY10], which uses HOGs and a nearest neighbor search. We

further compare our results to those obtained using the extended active shape model (STASM) suggested in [MN08]. In what follows we refer to the original method as HOGs+NNS, and to our method as EHOGs+OVVs. When comparing errors we use an approximation error of $\epsilon = 0$ for HOGs+NNS, to ensure best accuracy of the competitive method. When we compare the timings we select $\epsilon = 10$, as suggested in [KSYY10], to ensure low computational costs for HOGs+NNS. For all experiments we used a Pentium 4 PC with a 2.6GHz Quad core CPU and 2Mbyte memory.

4.5.1. Training

We gathered 969 upright frontal-view face images from various sources by using the Viola-Jones face detector [VJ01]. We manually marked 21 facial feature points like eyes, eyebrows, nose and mouth (see an example in Figure 4.5). Using the 9×9 block design (Section 4.3) we extract 78,489 HOG patterns (969 images, 81 segments per image) and 1,648,269 offset vectors (969 images, 81 segments per image, 21 feature points per segment) to compute our compact codebook containing EHOGs and OVVs (Section 4.3).

In order to be able to compare our method with the original approach we additionally extracted 20,349 local likelihood HOG patterns (969 images, 21 feature points per image) at the positions of the facial features. The set of the local HOG patterns, the offset vectors, and the local likelihood HOG patterns were used to generate the codebook for HOGs+NNS.

Data preparation We use the FERET duplicate I dataset [PMRR00] to collect a set of 722 probe images and manually selected the 21 facial features to define the ground truth for the evaluation. Note that the training set in Section 4.5.1 is independent of this probe set. The first step in each localization is to detect [VJ01] and to subsample the face region.

4.5.2. Storage Size

As shown in Figure 4.6 the energy preserving level during PCA can be used to control the storage size of the compact codebook. While the original codebook

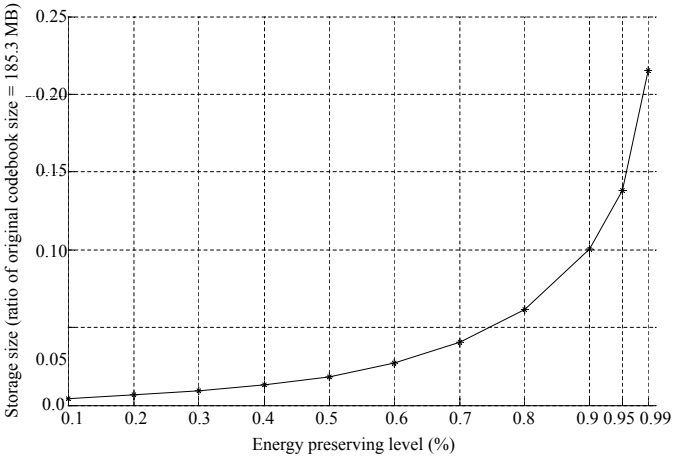


Figure 4.6.: Storage size vs. energy level.

(energy level at 100%) needs about 185 MB [1], the storage size of EHOGs+OVVs is drastically reduced as the energy preserving level decreases. At an energy level of 70%, the codebook has only 5% of the size of the original codebook. Even when the level is at 99%, the storage size is less than a quarter of the original codebook size. This means that the space of HOG patterns has a large amount of redundancy and that applying PCA effectively reduces the storage size.

4.5.3. Performance without occlusion

In Figure 4.7 we show how the localization error and the computational time changes according to the energy preserving level. The computational time represents the time to extract the HOGs, to deduce offset vectors and to compute the final feature positions. The detection error was measured as the pixel distance between the detected feature points and the manually marked ground

[1] We use approximately only half the number of training images as Kozakaya et al. [KSYY10] does.

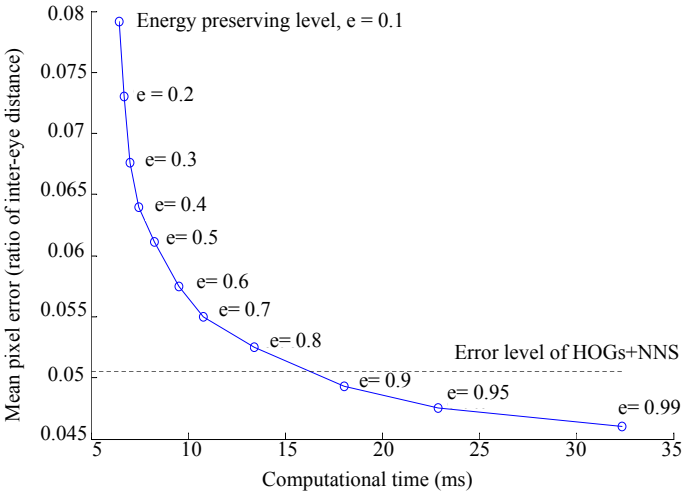


Figure 4.7.: Localization error vs. computational time. The error level of HOGs+NNS is computed using an approximation error of $\epsilon = 0$.

truth. The mean pixel error is calculated as the average of the pixel distances divided by the distance between the centers of the eyes. When decreasing the energy level the computational time drops since the codebook contains less EHOGs and OVVs which saves time in the projection step and during the combination of OVVs (see Section 4.4.1). In order to compare the errors we added the best error level to Figure 4.7, which was achieved by HOGs+NNS (0.0507). Compared to this error level, it is possible to decrease the energy preserving level by nearly 80% until the error is larger than that of HOGs+NNS. Note that the localization error of the energy preserving level above 90% is smaller than the error of HOGs+NNS. Since we combine offset vectors from different HOGs instead of selecting the best match, we tend to smooth the final positions of the feature points resulting in a more accurate localization.

In Figure 4.8 we show the localization result without occlusion for each individual feature point. This indicates that, compared to HOGs+NNS, the

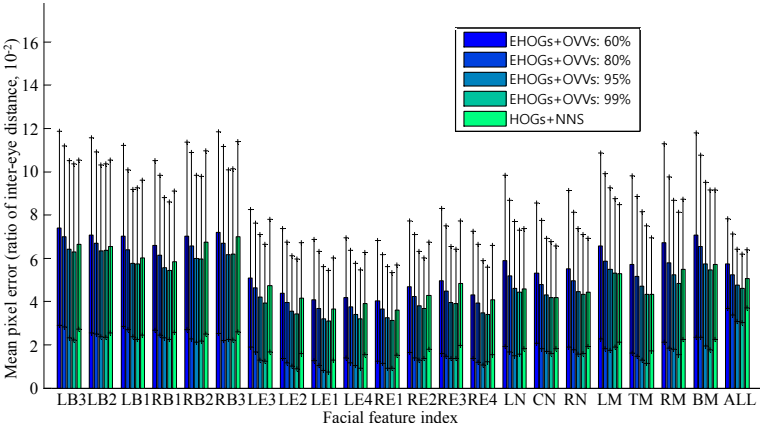


Figure 4.8.: Localization errors and standard deviations for individual feature points without occlusion.

proposed method which utilizes the statistical characteristics of segments from the training set, performs better in terms of accuracy and storage size.

In Figure 4.9 the localization error compared to STASM [MN08], another state-of-the-art facial feature localization method, is presented. The error values of STASM are adopted from [KSYY10] because they used the same test database. With an energy preserving level over 80%, our proposed method produces much better localization results than STASM for most of the feature points except the nose tip and corners where we observe a slightly larger error using our method.

4.5.4. Performance with partial occlusion

In order to evaluate the robustness of our method against partial occlusion, we place a white block on the test images. Its size is 10% of the area of the facial region while its position is randomly selected. The results for HOGs+NNS and EHOGs+OVVs are shown in Figure 4.10. The tendency of the average accuracy is similar to the non-occluded case (Figure 4.8). However, the difference between the errors produces by HOGs+NNS and EHOGs+OVVs is often

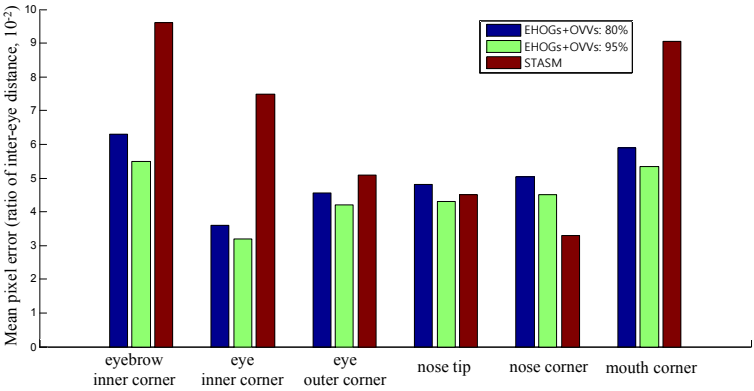


Figure 4.9.: Localization error comparison of our method and STASM.

much higher when parts are occluded. This holds especially, for the right eye brow (RB3), right eye (RE1-4), and mouth (LM, RM, BM). As described in Section 4.4.2 we use a completely different approach to compute the certainty of a offset vector, which exploits eigenvalues obtained during PCA. This allows EHOGs+OVVs to handle occlusions much more robustly.

Figure 4.11 shows several results (with and without occlusion) of the proposed facial feature localization with 90% energy preserving level. The images at the top row are results without occlusion and those on the bottom row show results with 10% occlusion. The blue rectangle represents the coarse face region detected by the Viola-Jones face detector, green dots are the segment centers and red dots are the detected facial feature points. The proposed method localizes the feature points correctly on non-occluded faces. Even on occluded faces, the localization of occluded feature points, inferred from offset vectors originating at not occluded segments, works quite well. The sixth column of Figure 4.11 shows a face rotated by 20°. Since our training database does not contain these images, this causes an incorrect localization of the feature points.

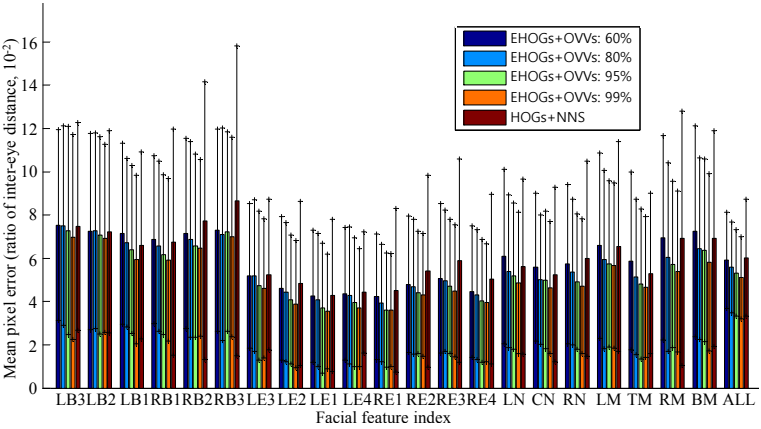


Figure 4.10.: Localization errors and standard deviations of facial feature points with 10% occlusion.

4.5.5. Summary of the evaluation

For easy comparison we summarize time consumption, required storage size, and accuracy of EHOGs+OVVs and HOGs+NNS in Table 4.1. Our method is very fast, accurate, and consumes low memory compared to HOGs+NNS. In order to get optimal timings for HOGs+NNS we set the approximation error to $\epsilon = 10$ as suggested in [KSYY10].

The difference in timing is due to the fact that HOGs+NNS needs to calculate more HOG patterns at the prospective facial feature points (81 segments x 21 feature points = 1701 HOGs) which requires several hundred milliseconds computational time. According to Table 4.1 EHOGs+OVVs with an energy preserving level of 80% performs best regarding the balance between storage size (94% memory is saved compared the the original codebook) and accuracy (slightly worse than HOGs+NNS).

Figure 4.11.: Some exemplary results for the localization using an energy preserving level of 90%. Top row: no occlusion. Bottom row: 10% occlusion. All images are from the FERET face database.

Method	EHOGs+OVVs 60%	EHOGs+OVVs 80%	EHOGs+OVVs 95%	EHOGs+OVVs 99%	HOGs+NNs ($\epsilon = 10$)	HOGs+NNs ($\epsilon = 0$)
HOG	3ms				483ms	
OV	3 ms	7 ms	16 ms	25 ms	16 ms	1490 ms
WVC	5ms					
Total	11 ms	15 ms	24 ms	33 ms	499 ms	1978 ms
Storage size	5.0 MB (2.7%)	11.4 MB (6.2%)	25.5 MB (13.8%)	39.8 MB (21.5%)	185.3 MB (100.0%)	
∅ error σ error (no occlusion)	0.0572 0.0193	0.0526 0.0189	0.0475 0.0167	0.0457 0.0156	0.0527 0.0151	0.0507 0.0136
∅ error σ error (10% occlusion)	0.0593 0.0213	0.0571 0.0204	0.0515 0.0192	0.0506 0.0189	0.0631 0.0185	0.0602 0.0174

Table 4.1.: Performance comparison. For different methods the table shows the total time consumption (Total) to extract facial features. The total time is split into times for extracting HOGs (HOG), deriving offset vectors (OV), and computing the feature points positions using weighted vector concentration (WVC). The last rows contain average errors (∅) and standard deviations (σ) for the localization with and without occlusion.

5. Reconstructing Static Faces

Although the final goal of our reconstruction pipeline is to reconstruct dynamic facial expressions from images captured with a calibrated and synchronized camera rig, in this chapter we first deal with the reconstruction of a static human face showing the neutral pose. Such a neutral pose is usually shown in the first frame of the synchronized videos. Its reconstruction will later be used to initialize the automatic tracking and reconstruction process, which ensures the reconstructed faces to be in full correspondence.

We basically describe two reconstruction systems. The first system, presented in [SHK09, SHK11], was a prototypical system, which was designed to reconstruct only the inner part of the human face. It omits peripheral regions near the cheek, the ears and the forehead. In this work we did not focus on the performance of the system, so reconstructing a large database of facial expressions, as it was needed in the approach presented in Chapter 7, was hardly possible using this system. While targeting at better performance and robustness, we reimplemented the system to be able to efficiently reconstruct a large database of dynamic faces.

In order to reconstruct static faces from multiple synchronized images both systems were designed to solve the following three problems:

1. Initialization: Adaption of a generic face template to a sparse set of 3D facial features (Section 5.3)
2. Stereo Reconstruction: Computation of a dense geometric representation of the surface (Section 5.4)
3. Fitting: Refinement of the face template to approximate the dense surface reconstruction (Section 5.5)

In what follows we first describe the generic face templates used for all reconstruction and tracking approaches. Having a face template with fixed topology enables the generation of facial reconstructions which are in full correspon-

dence. Section 5.2 describes a morphable face model which we will later use to stabilize the reconstruction process. In the Sections 5.3 - 5.5 we present the afore mentioned two approaches which differently solve the three sub-problems to reconstruct human faces showing a neutral facial expression.

5.1. Generic Face Templates

The ability to produce reconstructions that are in full correspondence is based on a generic face template, whose topology is constant during the whole reconstruction process. This face template is a high resolution, triangular mesh with some extraordinary feature points, like the corners of the eyes, the tip of the nose, some points around the mouth area and the eyebrows, which can uniquely be identify as specific landmark points as depicted in Figure 5.1. The aim in all the presented static and dynamic reconstruction methods is to deform this face template, such that (a) it geometrically fits the surface to be reconstructed and (b) the uniquely identifiably features always lie at corresponding points on the surface to be reconstructed. Then the geometrically adapted face template represents an individual face at a specific time, while correspondences to other facial reconstructions can be established simply by the ordered vertices of the generic face template.

During this thesis we employed to versions of this face template. The first version was used in the prototypical approach presented in [SHK09, SHK11] and covers only a relatively small area of the face (cf. left image in Figure 5.1). We used a small area, since the implementation in [SHK09, SHK11] could not deal with completely unseen areas like the neck and the left and right side of the head. The improved system we used in Chapter 7, can better deal with those hidden configuration, so we enlarged the face template to cover the complete area of the human face (cf. right image in Figure 5.1)). Notice that both face templates have disk topology with additional holes for the eyes and the mouth.

5.2. A Static Morphable Face Model

In general the shape of an individual face at a certain time is represented by the adapted vertex locations of our generic face template. Since this involves the

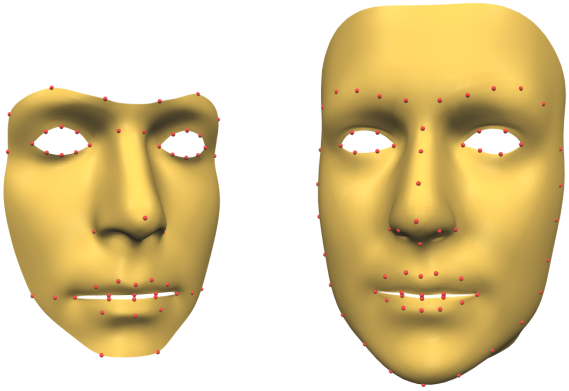


Figure 5.1.: Two versions of a generic face template we used in this thesis to reconstruct and synthesize facial expressions. The left image shows the face template we used in a prototypical reconstruction system. In Chapter 7 we used the model depicted in the right image to better cover the facial region. For both models we marked some extraordinary vertices (red points) as facial feature points.

concatenation of all vertex positions, this is a high dimensional representation, which probably contains much redundancy, since all facial shapes show some common features like eyes or noses.

Blanz and Vetter [BV99] reduced the redundancy of the shapes by introducing a static morphable face model. Analogous to the description in Section 2.3.2, where we used the Principal Component Analysis (cf. Section 2.2.1) to derive a shape model from a training set of 2D meshes, Blanz and Vetter built a shape model from a training set of high resolution 3D meshes. When the 3D meshes of the training set have been registered to a common coordinate system and are in full correspondence (cf. Section 5.2.1), all meshes have the same topology. In particular, the amount of vertices is the same for all meshes in the training set and we can represent each facial shape as a vector

$$\mathbf{s} = (x_1, y_1, z_1, \ldots, x_n, y_n, z_n)^T \in \mathbb{R}^{3n}$$

representing n vertex positions. Analyzing this high dimensional data using a Principal Component Analysis, reveals the average shape $\bar{\mathbf{s}}$ of the training set and leads to an orthonormal matrix $S \in \mathbb{R}^{3n \times k}$ with a small parameter k, such that each facial shape in the database can be approximated by

$$\mathbf{s} = \bar{\mathbf{s}} + S\mathbf{p} \tag{5.1}$$

where $\mathbf{p} \in \mathbb{R}^k$ are called shape parameters, analogous to the shape parameter of an Active Appearance Model. The matrix S encodes deviations from the average facial shape and its columns are called *Eigenfaces* (cf. Figure 5.2). In order to exactly reconstruct a facial shape $\mathbf{s}$, one needs to use as many shape parameters as examples in the database. In practice it is sufficient to only use a subset of the k largest Eigenfaces to cover a large variety facial shapes.

With such a model Blanz and Vetter [BV99] were able to parametrize human faces using only a few values instead of a large vector of vertex positions. While they used this model to approximate the shape of human faces seen in single images, we employ it to initialize and thereby stabilize the stereo reconstruction. The last missing ingredient is the construction the training data containing a registered set of corresponding 3D meshes, which is addressed in the following Section.

5.2.1. Construction of the Training Set

In the first step to construct the training set, we laser scanned about 50 faces showing a neutral expression. These non corresponding scans contain holes and noise and are depicted in the top row of Figure 5.2. In those scans we manually marked the landmark points around the eyes, the nose, mouth and eyebrows (cf. Figure 5.2) to obtain a sparse set of 3D facial features $\mathbf{v}_k$ for each scan. To fit the generic face template to an individual scan we employ both mesh editing techniques presented in Section 2.6.2 and 2.6.1. Therefore we first define the generic face template to be the reference mesh in the as-rigid-as-possible mesh editing approach and constrain the extraordinary vertices to corresponding feature positions $\mathbf{v}_k$. Solving for new vertex positions of the face template using the iterative procedure presented in Section 2.6.2, leads to a mesh, whose extraordinary landmark vertices exactly lie at the user defined

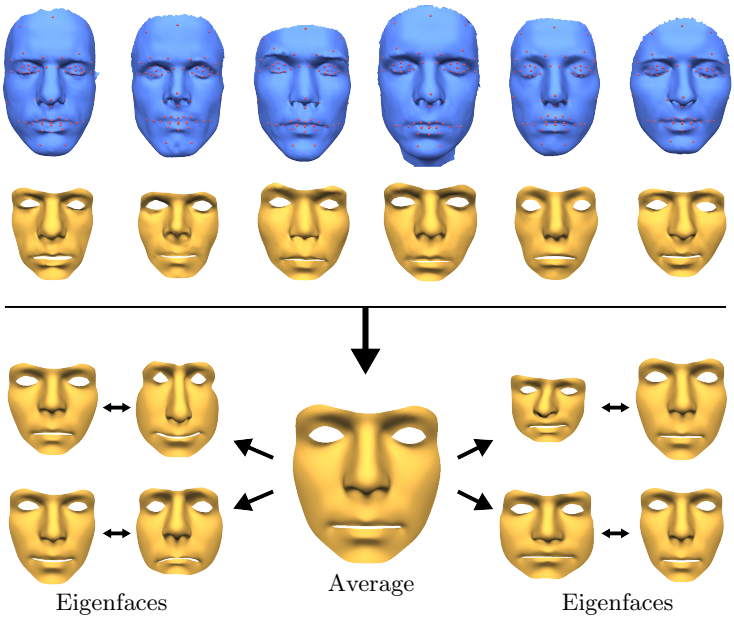


Figure 5.2.: Constructing a morphable face model. First row: input laserscans with marked facial features. Second row: Adapted face template that approximates the laserscanned surfaces. Form the training data we extract an average face and the so called *Eigenfaces*.

feature points of the laser scan and where local rigid rotations are smoothly distributed over the resulting surface.

This already leads to a close approximation of the the scan, which needs to be further refined in order to fit well to the laser scanned surface. Let the result of the first editing approach be a mesh $M = (V, T)$. For each vertex $\mathbf{p}_i \in V$ we find the closest point $\mathbf{v}$ on the laser scanned surface. If $\mathbf{v}$ is not a point on the boundary of the scan we consider it as a reliable point and use it as a soft constraint in the Laplace mesh editing approach (cf. Section 2.6.1). Doing this for all vertices leads to residual functions

$$\mathbf{g}_x = C \cdot \mathbf{p}_x - \mathbf{v}_x \tag{5.2}$$

where we again restrict the considerations on the x components of the points and where C is the matrix selecting the constraint vertices (cf. Equation 2.36). We slowly deform the mesh to fit the laser scanned surface by iterating the following steps

1. Find a global rigid transformation R that minimizes point correspondences
2. Transform the Laplace vectors according to R
3. Compute new vertex locations by solving

$$(w_v \cdot C^T C + L^T L) \cdot \mathbf{p}_x = w_v \cdot C^T \mathbf{v}_x + L^T L \mathbf{q}_x$$

4. Increase w_v
5. Select new vertex correspondences, i.e. update Equation 5.2

To find the rigid transformation in Step 1, we use the approach presented by [Hor87]. Then, to transform the Laplace vectors according to R, in Step 2 we only need to apply the rotational part to the original Laplace vectors $L\mathbf{q}$ of the reference mesh. In Step 3 the mesh is deformed to better fit the laser scanned surface. Since we increase the weight w_v in Step 4 the attraction towards the scan is increased, resulting in a closer approximation of the scanned surface in the next iterations.

The result of this process is a set of meshes with same topology, which are geometrically well aligned to their corresponding scans (cf. second row in Figure 5.2). From this training set we build the morphable face model for static faces.

5.3. Initial Reconstruction

Just using common 3D stereo approaches [SCD*06, HZ03, HK06, ES04] to reconstruct a human face would produce an unstructured point cloud with holes and outliers. We deal with this problem by properly initialize our face template to approximate the 3D surface, which can then be used as an accurate initialization to stabilize the stereo reconstruction. As input for this we assume a set of calibrated images, taken at the same time, where the user marked

some feature points. The feature points can also be found by a generic Active Appearance Model (cf. Section 2.3.2) for static human faces or by using the method presented in Chapter 4.

Feature Point Triangulation. If a 3D feature point is seen in more than two images, we can use Equation 2.25 to compute a 3D line with origin $\mathbf{o}_l$ and directional vector $\mathbf{d}_l$. The 3D position of this feature is then found by triangulation and can be estimated as the 3D point with minimal distance to all 3D lines, obtained from the 2D image positions. Under the consideration of Figure 5.3, the point of the line $\mathbf{c}_l(\mathbf{p})$ lying closest to $\mathbf{p}$ can be expressed as the linear combination

$$\mathbf{c}_l(\mathbf{p}) = \mathbf{o}_l - (\mathbf{d}_l^T \mathbf{o}_l)\mathbf{d}_l + (\mathbf{d}_l \mathbf{d}_l^T)\mathbf{p}$$

To find the optimal 3D feature location we search for a point $\mathbf{p}$ minimizing the distances to all lines, which leads to residuals of the form

$$\mathbf{f}_l = \underbrace{\mathbf{o}_l - (\mathbf{d}_l^T \mathbf{o}_l)\mathbf{d}_l}_{\mathbf{b}_l} + \underbrace{(\mathbf{d}_l \mathbf{d}_l^T - I_3)}_{A_l}\mathbf{p}$$

where I_3 is the 3×3 identity matrix. When concatenating the introduced vectors $\mathbf{b}_l \in \mathbb{R}^3$ and matrices $A_l \in \mathbb{R}^{3\times 3}$ to a large vector $\mathbf{b}$ and a matrix $\mathbf{A}$, the 3D feature position can be computed in the least squares sense by solving the linear system $\mathbf{p} = -(A^T A)^{-1} A^T \mathbf{b}$.

Doing this for a set of m feature points leads to a point cloud $\mathbf{v} = (\mathbf{v}_1^T, \ldots, \mathbf{v}_m^T)^T$ which represents the facial shape at a very coarse geometric level. These points have known corresponding vertices among the vertices of our face template. When using a consistent ordering of the feature points we can define a matrix $C \in \mathbb{R}^{3m\times 3n}$ that selects these vertices from the face template: Setting the entries $C_{3i,3j}$, $C_{3i+1,3j+1}$ and $C_{3i+2,3j+2}$ to 1 if vertex j of the face template corresponds to the i^{th} triangulated feature point, the vector $C \cdot \mathbf{s} \in \mathbb{R}^{3m}$ contains only the vertices of the face template that correspond to the feature points. We will use this notation in the next Section to fit the face template to the triangulated feature points.

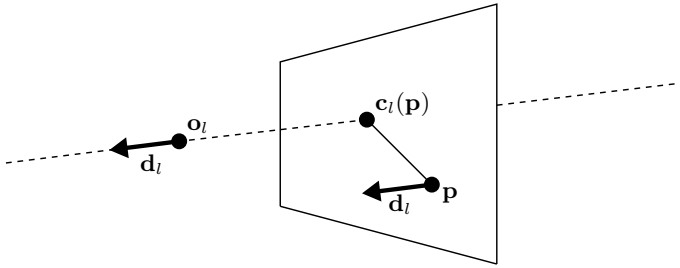


Figure 5.3.: Computing the closest point on a line w.r.t. to a 3D point $\mathbf{p}$ can be done analytically. To get the 3D location of a feature point can be formulated as an optimization problem, that finds a point $\mathbf{p}$ minimizing the distances to multiple 3D lines.

5.3.1. Approximation using a morphable model

The technique we present in this section was applied in [SHK09, SHK11]. It uses a sparse set of 3D facial feature points to compute an approximation of the observed surface which is later used to initialize the stereo reconstruction. The inputs are the triangulated feature points $\mathbf{v}$ which have known corresponding vertices

$$C \cdot \mathbf{s} = C \cdot (\bar{\mathbf{s}} + S \cdot \mathbf{p}) \tag{5.3}$$

among the vertices of our morphable model (cf. Equation 5.1). The shape parameters $\mathbf{p}$ only deform the surface and do not incorporate any rigid transformations. In order to take this into account, we allow the morphable model to be transformed according to three rotational and tree translational parameters. To simplify the notation we concatenate multiple copies of the global rotation and the global translation in a $3m \times 3m$ block diagonal matrix R respectively in a $3m$-dimensional vector $\mathbf{t}$. Then the differences between the triangulated feature points and its corresponding vertices of the morphable model can be measured by the residual functions

$$\mathbf{f}(R, \mathbf{t}, \mathbf{p}) = R \cdot C \cdot (\bar{\mathbf{s}} + S \cdot \mathbf{p}) + \mathbf{t} - \mathbf{v} \tag{5.4}$$

Since the parameters of the rigid transformation appear in addition to the shape parameters in this residual function, it is highly non-linear. Although we could

employ the Levenberg-Marquard algorithm [NW06] to optimize for the parameters simultaneously, we propose an interative scheme similar to As-Rigid-As-Possible mesh editing. Therefore we first compute an optimal alignment using the approach presented in [Hor87]. Then we consider the rigid transformation to be fixed and compute new shape parameter $\mathbf{p}$ encoding the deformation of the surface.

Directly forcing the vertices lying at the corresponding triangulated features, might produce large deformations. To drive the development of the deformation towards regular shapes, i.e. in the sense of plausible facial shapes, Blanz and Vetter [BV99] introduced additional residual functions

$$\mathbf{g} = D \cdot \mathbf{p} \tag{5.5}$$

that measure the deviation from the average shape $\mathbf{p} = \mathbf{0}$. By setting the elements $D_{i,i} = \frac{1}{\lambda_i}$ of the diagonal matrix $D \in \mathbb{R}^{k \times k}$ to the inverse of the eigenvalues associated to the shape parameters (cf. Section 2.2.1), they rather penalize deviations of those shape parameters, which have only a small standard deviation.

Using both residual functions we define the energy function

$$E(\mathbf{p}) = \mathbf{f}^T\mathbf{f} + w_s \cdot \mathbf{g}^T\mathbf{g} \tag{5.6}$$

and minimize it in the least squares sense (cf. Section 2.1). Before deriving Equation 5.6 w.r.t. the parameters $\mathbf{p}$ we introduce the abbreviatory notations $S_C = C \cdot S$ and $\mathbf{v}' = R^T \cdot (\mathbf{t} - \mathbf{v})$, such that the linear system of equations minimizing the energy function can be stated as

$$(S_C^T \cdot S_C + w_s \cdot D^T \cdot D) \cdot \mathbf{p} = -S_C^T \cdot (C \cdot \bar{\mathbf{s}} + \mathbf{v}') \tag{5.7}$$

Note that we omitted the term $R^T R$ on the left hand side, since this equals the identity matrix. In each iteration, where we compute a new rigid transformation and new shape parameters, we lower the regularization value w_s to allow for a more flexible surface deformation.

5.3.2. Approximation using ARAP-Modelling

The system presented in [SHK09, SHK11] uses *Surfel Fitting* (cf. Section 2.5.3) to obtain a dense reconstruction of the surface to be reconstructed. Since it

associates to each point a normal, it is able to accurately model perspective distortions between image pairs. Nevertheless, it needs a good initial solution, where the image intensities around the projected point in the reference image roughly matches the image intensities around the projected point in the comparison image. To allow for a more flexible and thereby more accurate surface deformation, we included the additional degrees of freedoms of a morphable model.

In the more advanced system, we used to compute our large database of facial expressions (cf. Chapter 7), we were able to simplify the initialization procedure drastically, since the stereo reconstruction is based on a searching approach. For a reference image point we basically search for a corresponding point in the comparison image by traversing the epipoloar line. This allows to find corresponding image points, even if their respective regions do not overlap at the beginning, as it was needed for *Surfel Fitting*.

Clearly it is not wanted to search along the whole epipolar line, visible in a comparison image. To increase accuracy and computation time we still compute a proper, more rough initialization of the surface to be reconstructed. For this we use the ARAP-Modelling framework (cf. Section 2.6.2), where we require each extraordinary vertex of the face template to lie exactly on the positions of its corresponding triangulated feature point. This results in a set of equations

$$C \cdot \mathbf{p} = \mathbf{v}$$

which serve as constraints for the ARAP-Mesh editing.

5.3.3. Results

To analyze the two initialization methods either using a morphable model (MM) or as-rigid-as-possible mesh editing (ARAP), we compare the result of the initialization with the reconstruction we obtain in the last step of our pipeline (cf. Section 5.5). Thereby we substitute the missing ground truth data with the final reconstruction, which gives us a hint how much the surface still needs to be deformed until it approximates the observed facial surface. Figure 5.4 shows a few examples of this initialization process. Each example shows (from left to right) an original photo, the face template deformed by MM and the face template deformed by ARAP. In each example we color code the distance

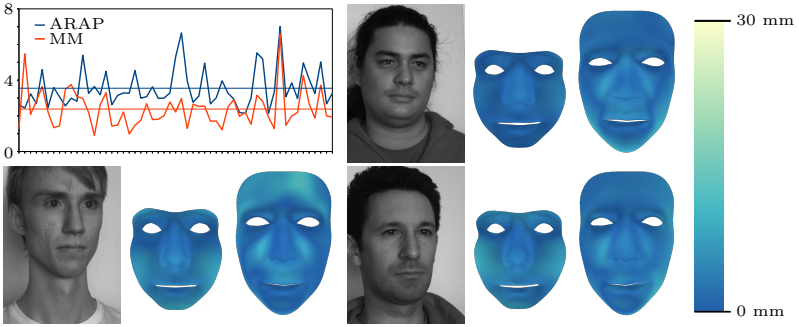


Figure 5.4.: Accuracy of both initialization methods (MM and ARAP). For each individual face (x-axis), the plot shows the average vertex distance (in mm) between initialization and final reconstruction. Examples visually indicate MM leads to a better initialization than ARAP, but it also comes with a higher algorithmic complexity.

between a vertex of the initial face template and the corresponding vertex of the final reconstruction. The color coding indicates that the deformation of the ARAP-initialization is in some regions much higher than the deformation of the MM-initialization. We quantified this visual impression by computing an average vertex displacement for overall 55 different individual faces. The plot in Figure 5.4 shows the distribution of these average displacements. In nearly all initializations we obtain lower errors when using a morphable model and the mean of the error is with 2.5mm for the MM-initialization also lower than 3.5mm for the ARAP-initialization. Although it produces a lower error it is noticeable that the ARAP-initialization is much simpler, since it is only based on one face template instead of a full database of facial shapes.

5.4. Stereo Reconstruction

In this section we compare two different stereo reconstruction approaches designed to compute a dense reconstruction of the facial surface. While the first approach optimizes small planar surface patches, the second method is based on the search for a corresponding image point lying on the epipolar line. Both

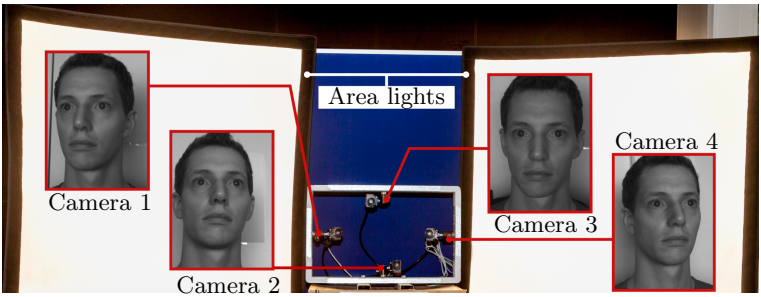


Figure 5.5.: The used camera rig with four cameras and two area light sources.

methods start with an approximation of the facial surface (cf. Section 5.3) in order to properly initialize the optimization or to limit the search intervals.

Having such good initializations is not typical for stereo reconstruction methods. In some applications the user interactively specifies some initial depth values [HK09] or the depth of the surface is estimated in some reliable regions and propagated to neighboring regions. When reconstructing human faces, which usually show some common geometric characteristics, we have the advantage that it is possible to compute an initial surface from which the stereo reconstruction can be started. On the other hand, multiview stereo is especially unstable when using it to reconstruct human faces. This is because those approaches perform best when the surface is well textured, which is usually not the case for human skin, which appears as rather homogenous region in an image when captured at a macroscopic level (cf. [BHPS10]).

Since our system only uses four cameras to capture the whole face at once, we need to apply filtering and smoothing operations in order to get a reliable surface reconstruction. The four cameras of the rig are able to capture images at a framerate of 40 fps with a resolution of 580×780 pixels. The set-up containing the four camera rig with two area light sources and examples of four captured images are shown in Figure 5.5. In what follows we detail both reconstruction methods and show some comparative results.

5.4.1. Surfel based reconstruction

Our surfel based reconstruction approach optimizes small surface elements tangential to all vertices of the face template using surfel fitting, which is described in detail in Section 2.5.3. This approach emerges from the Lucas-Kanade image alignment, which adjusts a warping function such that pixel intensities from a reference image region correlate with the pixel intensities from a region of a comparison image. The optimization only converges if the image regions initially show parts of the same content, meaning if the position and orientation of the initial surfel is already close to the optimal solution. As mentioned in Section 5.3.1, this is the reason why we use here the more accurate initialization obtained from fitting a morphable model to the feature points.

Given the result from the fitting procedure (cf. Section 5.3.1) we initialize a plane for every vertex i of the face template using its vertex position $\mathbf{q}_i$ and normal $\mathbf{n}_i$. This potentially leads to the same number of surfels as vertices of the face template. As described in Section 2.5.3, for each surfel we need a reference image and a set of comparison images. Therefore we create one depth image of the initial face template as seen from each camera center using the OpenGL z-Buffer [AMHH08]. This allows us to check if a specific vertex is visible form a certain camera position by comparing the distance between vertex position and camera center to the distance stored in the depth image. If the distance between vertex and camera is significantly larger than the distance stored in the depth image, the vertex is not visible w.r.t. that camera. For each vertex, this leads to a set of images, where the vertex is visible. Among those images we select the reference image as that image whose viewing direction is most parallel to the vertex normal and define the remaining images of the set to be the comparison images.

Given the initial plane parameters, a reference and a set of comparison images, we start the optimization approach described in Section 2.5.3. The result of this optimization does not always lead to a usable result. Occasionally, due to noise in the images or badly textured parts, this process does not succeed for every plane. If the plane equation is numerically ill-conditioned, the Levenberg-Marquard algorithm did not converge or if the vertex is not visible in two or more images, we discard the result.

5.4.2. Point based reconstruction

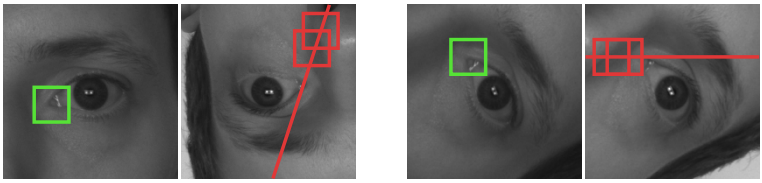
Stereo reconstruction is often based on a simple search procedure [HZ03] to identify corresponding pixels in a left (reference) and a right (comparison) image. Under the consideration of the epipolar geometry (cf. Section 2.5.1) the two dimensional search can be restricted to one dimension: For a specific pixel in the left image, one searches for a corresponding pixel in the right image lying on the epipolar line associated to the left pixel.

Normalized Cross Correlation. For any pair of pixels $(\mathbf{x}_l, \mathbf{x}_r)$ from the left and the right image, it is essential to measure the quality of the correspondence. One of the most commonly used quality criterion is the *Normalized Cross Correlation* (NCC) [ES04, HZ03, HLL11, SIA*13], which compares the color or intensity distribution in a small image region, called *correlation window*, around both pixels (cf. Figure 5.6). The size of the correlation window in image space should be selected such that it covers a reasonably sized 3D surface patch. In our settings, where we deal with images of 580×780 pixels and where we assume the subject to sit at a certain distance w.r.t. to the camera rig, we set the correlation windows to 15×15 pixels. The intensity values interpolated at the pixel positions within a correlation window can be concatenated to form a $15 \cdot 15 = 225$ dimensional vector. For the pixel $\mathbf{x}_l$ and the pixel $\mathbf{x}_r$, this leads to two intensity vectors $\mathbf{I}(\mathbf{x}_l)$ and $\mathbf{I}(\mathbf{x}_r)$ representing the small sub-images from the left and right images around both pixels. The normalized cross correlation now measures the similarity of those sub-images and compensates for differences in the illumination conditions. It is defined as

$$NCC(\mathbf{x}_l, \mathbf{x}_r) = \frac{\mathbf{I}^T(\mathbf{x}_l) - \bar{\mathbf{I}}^T(\mathbf{x}_l)}{||\mathbf{I}(\mathbf{x}_l) - \bar{\mathbf{I}}(\mathbf{x}_l)||} \cdot \frac{\mathbf{I}(\mathbf{x}_r) - \bar{\mathbf{I}}(\mathbf{x}_r)}{||\mathbf{I}(\mathbf{x}_r) - \bar{\mathbf{I}}(\mathbf{x}_r)||} \in [-1, \ldots, 1]$$

where $\bar{\mathbf{I}}$ is a vector with same dimensionality as $\mathbf{I}$, where each entry holds the average intensity value of $\mathbf{I}$. When the left and right correlation window show the same surface patch, ideally the NCC value reaches its maximum value of 1.

Search for corresponding pixels. The biggest problem with this quality measurement is the shape of the correlation window: in the simplest implementa-



Input pair Rectified pair

Figure 5.6.: Close-ups of correlation windows around pixels of the input image pairs and image pairs, which have been rectified. For a correlation window in the left image (green) we search for a matching window (red) in the right image by traversing the epipolar (red) line. In the rectified setup the correlation windows are automatically aligned correctly.

tion of the search along the epipolar line it is assumed that the left and right correlation windows have the same shape and that pixels are e.g. ordered from the upper left to the lower right corner of the window. As seen for not rectified images (cf. left of Figure 5.6) this is especially problematic if not all cameras of the rig face upwards, as it is the case for our camera rig. In such a case one should at least rotate the correlation window of the right camera to be aligned with the correlation window of the left camera. It it also problematic to assume a constant shape for correlation windows, since different orientations of the surface patches would induce perspective distortions which strictly spoken would alter the shape of the correlation window.

An elegant way to handle rotations is to search corresponding pixels in a rectified camera setup. For the used rig, we therefore define pairs of cameras (cf. Figure 5.7) and perform an epipolar rectification for each pair. As outlined in Section 2.5.2, after the rectification, epipolar lines are parallel to the x-axis of the image coordinate system and corresponding epipolar lines are located at the same y coorindate. In this setting the correlation windows are properly aligned (cf. Figure 5.6), and we do not need to consider any rotations. In order to compensate for perspective distortions we follow the approach presented by Bradley [BBH08]. Their idea is to extend the search along the epipolar line by using three differently scaled versions of the correlation window in the right image. Since the left and the right principal point is located at the same

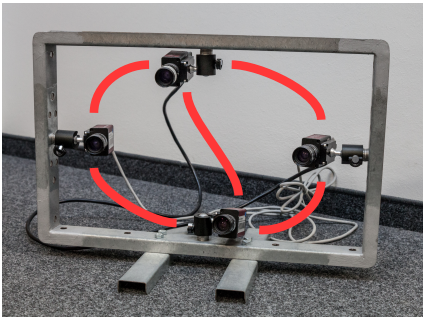


Figure 5.7.: Stereo pairs we use for the point based stereo reconstruction. Since the baseline between the left and right camera is to large, we simply omit this pair.

y-coordinate, the perspective distortion in y direction is the same for the left and right correlation window. Hence, Bradley only considers the scaling of the correlation window in x direction by constant factors $\sqrt{2}, 1$ and $1/\sqrt{2}$ to approximate three different surface orientations. Instead of scaling the correlation window to compensate for perspective distortions, it is more efficient to scale the right (comparison) image with the inverse factors and use uniformly shaped correlation windows with a fixed size of 15×15 pixel.

Potentially for each left pixel, one would need to perform the search along the whole epipolar line of the right image. Besides a decreased performance this is also problematic since the more pixel pairs are compared, the higher is the probability to assign the wrong correspondence. We overcome both problems by a proper initialization of the depth values using the face template fitted to the triangulated feature points as described in Section 5.3.2. To efficiently obtain the initial depth values we use the OpenGL rendering pipeline [AMHH08] to simulate the projective behavior of the left rectified camera. During the rasterization step [AMHH08], a depth value is computed for each pixel and quantized in the interval between the near and the far plane. The definition of the projection matrix containing the values for the near and the far plane is crucial for some functions of the OpenGL rendering pipeline. *Culling*, e.g., removes (among others) primitives that lie in front of the near or behind the

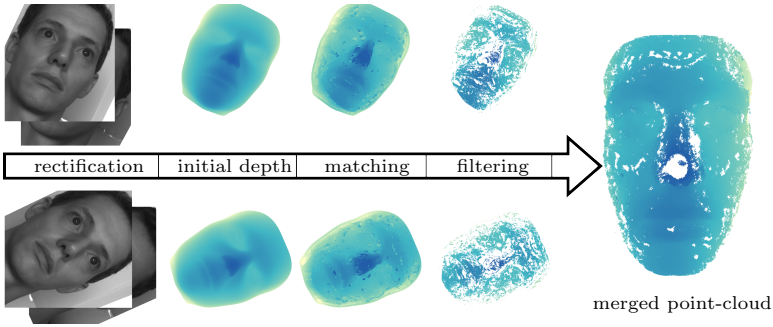


Figure 5.8.: Point based reconstruction pipeline. For each stereo pair (here: two pairs), we perform an initialization step to narrow down the search interval. In the matching step we use the GPU to efficiently find correspondences between the left and the right image. The filtering step removes outliers and smoothes the depth map. In the end the points produced by each stereo pair contribute to a dense point cloud (right image).

far plane. To avoid quantization artifacts for the depth values, we compute a near and far plane, that encloses the initial face template as tight as possible.

Initial depth values for each pixel of the left image are then obtained from on single draw call. Figure 5.8 shows an example of such an initial depth map, which additionally gives us a rough background segmentation from which we identify pixels whose depth values need to be optimized. In order to shrink the search interval on the epipolar line, we define a conservatively estimated depth interval of 5 cm we expect the surface to lie in. To simplify the computation of disparities, we again express pixels relative to the principal point of the rectified image (cf. Section 2.5.2). Assuming the initial depth value at a pixel $\mathbf{x}_l = (u_l, v)$ is z, we define the depth interval, the surface is expected to lie in, as $[z^-, z^+]$. Using Equation 2.31 this interval can be mapped to a pixel interval $[u_r^-, u_r^+]$ in the right image, where

$$u_r^- = u_l - \frac{b \cdot f}{z^-} \quad \text{and} \quad u_r^+ = u_l - \frac{b \cdot f}{z^+} \tag{5.8}$$

If the observed surface patch lies in the assumed depth interval, the pixel in the right image, which corresponds to the reference pixel from the left image,

must lie in the pixel interval $[u_r^-, u_r^+]$, i.e. we can restrict our search to this interval. Of course when scaling the right image to compensate for perspective distortion we also need to scale the search intervals.

Finding the right pixel corresponding to a left pixel $\mathbf{x}_l$ is then done in two steps. First we visit each integer pixel position from the interval $([u_r^-, u_r^+], v)^T$, and search for that pixel $\mathbf{x}_r$, which maximizes the normalized cross correlation

$$\mathbf{x}_r = \arg \max_{u \in [u_r^-, u_r^+]} NCC(\mathbf{x}_l, (u, v)^T)$$

to get an integer disparity value. After this we refine this value to sub-pixel precision.

Refining the disparities If we would only allow for integer valued disparities, this approach would lead to small discretization artifacts in the resulting depth values. The lenses we used in our camera rig have a focal length of 2000 mm, while the base line is about 300 mm. Hence a disparity difference of one pixel (for images of a resolution of 780×580 can lead to a variation in depth of over 1 mm. Since this is an unwanted source of error, we follow the approach of Bradley [BHPS10] and compute the optimal correlating pixel position with sub-pixel accuracy. For this Bradley suggest to frequently sample the space between two neighboring pixels and search for the optimal correlation value. Although this leads to a high precision estimate of the location of the corresponding pixel (they use 10 samples between two pixels) the result is not exact. Moreover it involves many interpolation steps and evaluations of the normalized cross correlation.

We prefer to compute the optimal position of the corresponding pixel analytically, since this involves fewer pixel look-ups and leads directly to the optimal result. When the best integer pixel position computed so far is $\mathbf{x}_r = (u, v)^T$, the sub-pixel interval can be located left from that pixel $(u-1, v)$ or right from that pixel $(u+1, v)$. In what follows we only consider the second case, since both cases are analogous. As depicted in Figure 5.9 we can extract two intensity vectors $\mathbf{L} = \mathbf{I}((u, v)^T)$ and $\mathbf{R} = \mathbf{I}((u+1, v)^T)$, which are obtained from correlation windows located at the two neighboring pixels in the right image. By introducing an interpolation value $\alpha \in [0, 1]$ we can express any intensity vector

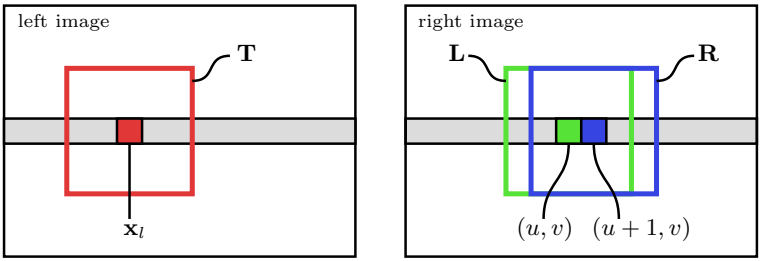


Figure 5.9.: To refine the pixel corresponding to a position $\mathbf{x}_l$ in the left image, two correlation windows are extracted at neighboring pixels in the right image. At a position between both pixels, a correlation window can be obtained by interpolating the windows $\mathbf{L}$ and $\mathbf{R}$. Our algorithm finds the optimal interplation weight reproducing a correlation window that best fits to the (normalized) reference window $\mathbf{T}$.

$\mathbf{C}(\alpha)$ between both pixels as the linear combination $\mathbf{C}(\alpha) = \mathbf{L} + \alpha(\mathbf{R} - \mathbf{L})$. Subtracting the average intensity value of $\mathbf{C}(\alpha)$ leads to a vector

$$\mathbf{I}(\alpha) = \mathbf{C}(\alpha) - \bar{\mathbf{C}}(\alpha) = \mathbf{L} - \bar{\mathbf{L}} + \alpha \cdot (\mathbf{R} - \bar{\mathbf{R}} - \mathbf{L} + \bar{\mathbf{L}}) = \mathbf{A} + \alpha \cdot \mathbf{B}$$

where we introduced the abbreviatory vectors $\mathbf{A}$ and $\mathbf{B}$. Together with the normalized intensity vector $\mathbf{T}$, extracted at the reference position $\mathbf{x}_l$ (left image), this allows us to define the normalized cross correlation in terms of α such that

$$NCC(\alpha) = \mathbf{T}^T \cdot \frac{\mathbf{I}(\alpha)}{||\mathbf{I}(\alpha)||}$$

The optimal interpolation value maximizes this function and can be computed by setting the derivative to zero, which leads to the equation

$$\frac{\partial NCC(\alpha)}{\partial \alpha} = \mathbf{T}^T \cdot \frac{\mathbf{B} \cdot ||\mathbf{I}(\alpha)|| - \mathbf{I}(\alpha) \cdot ||\mathbf{I}(\alpha)||^{-1} \cdot (\mathbf{A}^T\mathbf{B} + \alpha\mathbf{B}^T\mathbf{B})}{||\mathbf{I}(\alpha)||^2} = 0$$

When this equation is multiplied with the factor $||\mathbf{I}(\alpha)||^3$, the norms vanish and the equation simplifies to

$$\mathbf{T}^T \cdot \mathbf{B} \cdot \mathbf{I}^T(\alpha) \cdot \mathbf{I}(\alpha) - \mathbf{I}(\alpha) \cdot (\mathbf{A}^T\mathbf{B} + \alpha\mathbf{B}^T\mathbf{B}) = 0$$

When we further insert the definition $\mathbf{I}(\alpha) = \mathbf{A} + \alpha \cdot \mathbf{B}$, expand the equation and rearrange terms w.r.t. to α we get the equation

$$\alpha \cdot (\mathbf{T}^T\mathbf{B}\mathbf{A}^T\mathbf{B} - \mathbf{T}^T\mathbf{A}\mathbf{B}^T\mathbf{B}) = \mathbf{T}^T\mathbf{A}\mathbf{A}^T\mathbf{B} - \mathbf{T}^T\mathbf{B}\mathbf{A}^T\mathbf{A} \tag{5.9}$$

which is linear in α, since the coefficients of α^2 cancel out each other. Solving the last equation w.r.t. α, which is easy since $\mathbf{T}$, $\mathbf{A}$ and $\mathbf{B}$ are constant, directly leads to the optimal interpolation value and thereby to the x coordinate of the sub-pixel

$$u_{\text{opt}} = (1 - \alpha) \cdot u + \alpha \cdot (u + 1) = u + \alpha$$

which correlates best to reference pixel.

After we identified the pixel with highest correlation value in a possible scaled image, its x-coordinate needs to be scaled back to the original image size from which we then compute the disparity value and thereby the new optimized depth value.

The refinement procedure is embedded in the filtering pipeline, and needs to be done only for those pixels, which survived the removal of outliers. An example of the effectiveness of the presented procedure is shown in Figure 5.10 (c). Stair-stepping artifacts, which arise from integer valued disparities, are clearly reduced after the refinement step.

Filtering the depth map. The first simple filter restricts the image region in the left image where we search for optimal depth values. From the depth map we used to initialize the search intervals, we identify the pixel which project on the initial face template. As mentioned before this gives us a rough partition of the image into fore- and background. Pixels from the background are filtered out and will not contribute to the final point cloud (cf. Image (a) of Figure 5.10).

For the next filter we switch the role of the left and the right image. For each pixel in the right image we search for a corresponding pixel in the left image. Observe that for this we need to create an initial depth map w.r.t the right image and perform the whole search procedure again. With the result of this additional reconstruction step we then can verify the correspondence $(\mathbf{x}_l, \mathbf{x}_r)$ computed so far: if we obtain the same correspondence after switching the role of the left and the right image, we consider this as a confirmation of

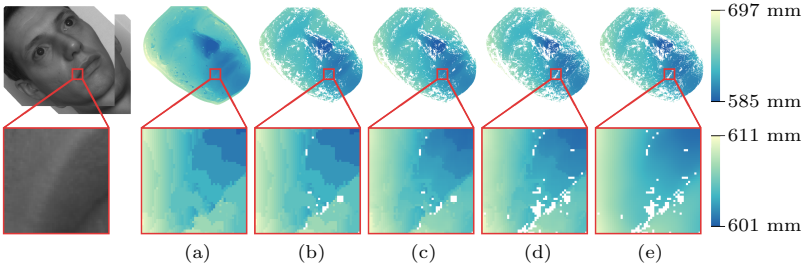


Figure 5.10.: Filtering the depth map. (a) initial depth map reconstructed form pair of images. (b) Removal of unconfirmed correspondences and bad NCC values. (c) The analytic refinement of pixel correspondences results in less stair-stepping artifacts. (d) Result of the median rejection filter and the removal of isolated pixels. (e) Smoothing result of the trilateral filter. The values of all presented depth maps are color coded according to the right chart. Close-ups are shown in the lower row.

the correspondence. Otherwise we do not use it to augment the final point cloud (cf. Image (b) of Figure 5.10).

For the other filters we follow the approach presented in [BHPS10]. These filters aim at the removal of outliers and at the reduction of noise, which is usually present in the captured images. The first obvious outlier filter is to garantee a certain reconstruction criterion. Therefore we remove all corresponding pixels $(\mathbf{x}_l, \mathbf{x}_r)$, where the normalized cross correlation $NCC(\mathbf{x}_l, \mathbf{x}_r)$ drops below a certain threshold (cf. Image (b) of Figure 5.10). Then we remove all pixels from the left image, which depth value is far from the median depth value of its surrounding neighbors. This will produce some holes in the depth map, which is further reduced by removing isolated pixel, i.e. pixels whose neighbors are not properly reconstructed (cf. Image (d) of Figure 5.10). In order to remove noise we use a bilateral filter. It is basically a Gaussian filter where the influence of a neighboring pixel from a small region Ω is not only affected by the distance to the center pixel, but also by the difference between both depth values stored at the pixels. Bradly suggests to additionally integrate the normalized cross correlation and increase the weight of a neighboring depth value [BHPS10], if the NCC value is high, meaning if the reconstruction

is very reliable. This third weighting criterion turns this filter to a trialteral filter. With the definition of a normal distribution in Equation 2.7 this can be stated as

$$z(\mathbf{x}) \leftarrow \frac{1}{W(\mathbf{x})} \sum_{\mathbf{y} \in \Omega(\mathbf{x})} \mathcal{N}(\mathbf{y}|\mathbf{x}, \Sigma_s) \cdot \mathcal{N}(z(\mathbf{y})|z(\mathbf{x}), \Sigma_z) \cdot NCC(\mathbf{y}) \cdot z(\mathbf{y})$$

where we slightly abused the notation $NCC(\mathbf{y})$ to be the resulting normalized cross correlation obtained after reconstructing the depth value of the neighboring pixel $\mathbf{y}$ and where $W(\mathbf{x}) = \sum_{\mathbf{y} \in \Omega(\mathbf{x})} \mathcal{N}(\mathbf{y}|\mathbf{x}, \Sigma_s) \cdot \mathcal{N}(z(\mathbf{y})|z(\mathbf{x}), \Sigma_z) \cdot NCC(\mathbf{y})$ is the sum of the computed weights. The weight depending on the distance of two neighboring pixels is here controlled by the covariance matrix Σ_s while Σ_z is adapted to suppress the influence of neighbors whose depth values strongly deviate from the depth value of the center pixel $\mathbf{x}$. This allows the filter to preserve depth continuities, often caused by occlusions. The result of the final depth map is visualized in Figure 5.10 (e). Stair-stepping artifacts are reduced by the analytic refinement of pixel correspondences and by the trilateral filter.

Parallel computation Since all operations, like the look up of pixel intensities and the computation of the NCC-values are rather simple and can be executed independent for each pixel, this approach is well suited to be parallelized. We use the CUDA framework [SK10] to efficiently compute the optimal pixel correspondences on a modern GPU. The proposed algorithm to optimize the initially computed depth values can be summarized by the following steps:

1. Undistort and rectify the input image pair
2. Compute a search interval (right image) for each pixel in the left image using Equation 5.8
3. Generate three rescaled version of the right image to compensate for possible perspective distortions.
4. Search for the highest NCC value within the (scaled) search intervals and refine the correspondence using Equation 5.9
5. Take the disparity with highest NCC value and update the depth using Equation 2.31
6. Filter the resulting depth map

	∅ NCC (σ NCC)	∅ points (σ points)	∅ μs (σ μs)
Surfel based	0.93 (0.01)	6270 (83)	613.697 (93.562)
Point based	0.85 (0.01)	431457 (83234)	3.399 (0.585)

Table 5.1.: Comparison of the two stereo reconstructions. While surfel fitting produces a better reconstruction quality (NCC value), the number of points (second column) and the time to reconstruct one point or surfel (third column) are better for the point based method.

For each camera pair (cf. Figure 5.7) we repeat this procedure and merge the resulting point clouds as indicated in the right image of Figure 5.8.

5.4.3. Results

We compare the quality of the surfel based (SB) reconstruction and the point based (PB) reconstruction by having a detailed look at the resulting NCC values associated to the resulting surfels and points. Since this is never explicitly computed for surfels (cf. Section 2.5.3), we additionally compute that value after the optimization. The examples in Figure 5.11 color code the NCC values for each point or surfel ranging from 0.5 to 1 (the value for a perfect match is $NCC = 1$, the minimum is $NCC = -1$). The visual inspection indicates that the quality of SB is much better than the result produced by PB. This is not surprising, since perspective distortions, induced by the surfels orientations, are inherently handled by the warping function defined in Equation 2.33. We verify this observation by using both methods to reconstruct 55 individual neutral faces. For each reconstruction we compute the average NCC value and its standard deviation, and plot the result in the diagram of Figure 5.11. As expected, the average NCC value is much higher for each face reconstructed with the surfel based approach. In the plot we scale the error bars, visualizing the standard deviation, by a factor of 0.1. This additionally shows that the NCC value is less fluctuating for the surfel based reconstruction, so it seems the the surfel based method should be preferred over the point based method. But, when looking at the number of reconstructed points and the time to reconstruct a single point or surfel (cf. Table 5.4.3), the point based method clearly shows

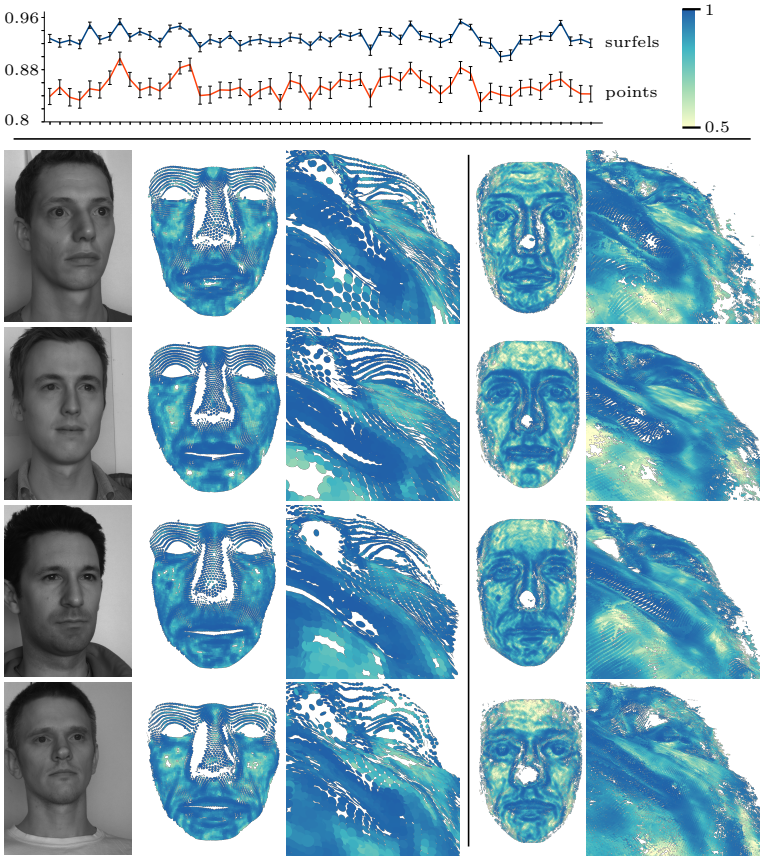


Figure 5.11.: Plot: Average NCC values and their standard deviation for each of the 55 performed reconstructions. The examples show color coded NCC values for the surfel fitting method (left) and the point based method (right).

some interesting advantages, since it can reconstruct much more points in the same amount of time. It is even fast enough to reconstruct nearly 300.000 points per second, while surfel fitting reconstructs approximately 1600 points

per second (we run all experiments on an Intel®Core™i7-4770 with an NVIDIA GeForce GTX 770). The hope is that due to the high number of reconstructions the final error averages out, such that the quality of the reconstruction is still comparable with the reconstruction produced by surfel fitting.

5.5. Fitting the Face Template

So far we deformed the face template to fit a sparse set of triangulated feature points and computed a dense point cloud representing the surface of the observed face. The last step to obtain the reconstruction of a static face is to fit the face template to that point cloud. While our prototypical application [SHK09, SHK11] uses a morphable model for this approach, the method presented in Chapter 7 is more simple and is based on Laplace Editing. This improved system also uses a more complete face template, where additional parts of the head are visible (cf. right image of Figure 5.1). Since the cameras of our rig only capture the front and slightly some peripheral parts of the human face, the resulting point clouds will never cover the full surface of that head model. In order to still be able to deform this face template in a meaningful way, we explicitly mark specific triangles of our face template as handle regions. These regions are depicted as red regions in Figure 5.5. When deforming the face template to fit the point cloud, correspondences to the face template are only established, if they fall into these handle regions. In what follows we describe both approaches to refine the face template to fit the reconstructed point cloud.

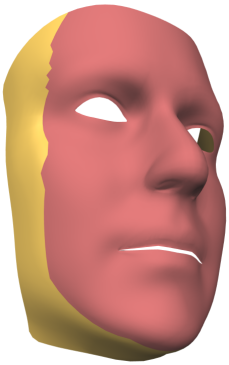


Figure 5.12.: Modifyable region

5.5.1. Refinement using a morphable model

The refinement of the face template using a morphable model follows the approach presented in Section 5.3.1. Here we alternated the computation of a

rigid transformation with the adaption of the shape parameters such that a sparse set of vertices (cf. Equation 5.3) fit well to a sparse set of triangulated (user defined) facial features. Now we can extend this fitting approach by incorporating the previously computed stereo reconstruction (cf. Section 5.4.1), which is represented as a set of point $\mathbf{q} = [\mathbf{q}_1^T, \ldots, \mathbf{q}_k^T]^T$. In contrast to Section 5.3.1, where each triangulated facial feature points is associated to only one vertex of the morphable model, we now do not know the association of the reconstructed points to points of the morphable model. Therefore we need to define new correspondences in each iteration of the optimization process. This is done by computing the closest point on the surface of the morphable model w.r.t. to the reconstructed points $\mathbf{q}_i$. In each iteration of the procedure optimizing rigid and shape parameters, this leads to a matrix $F \in \mathbb{R}^{3k \times 3n}$ which analogously to Equation 5.3 is used to map the vertices of the morphable model to a set of points which correspond to the reconstructed surface points $\mathbf{q}$

$$F \cdot \mathbf{s} = C \cdot (\bar{\mathbf{s}} + S \cdot \mathbf{p})$$

The closest point to $\mathbf{q}_l$ always falls into a specific triangle where it has barycentric coordinates w.r.t. the vertices adjacent to that triangle. To construct the matrix F the barycentric coordinates are inserted into F as 3×3 block matrices. If, e.g., j is a vertex of the closest triangle and α_j is the barycentric coordinate of the closest point w.r.t. to vertex j, we insert the entries $F_{3l,3j} = F_{3l+1,3j+1} = F_{3l+2,3j+2} = \alpha_j$ into the matrix F. This allows us to define an additional residual function that measures the distances between the reconstructed points $\mathbf{q}$ and their closest points on the surface of the morphable model as

$$\mathbf{h}(R, \mathbf{t}, \mathbf{p}) = R \cdot F \cdot (\bar{\mathbf{s}} + S \cdot \mathbf{p}) + \mathbf{t} - \mathbf{q}$$

where analogously to Equation 5.4 R and $\mathbf{t}$ encode the rigid transformation of the morphable model. We use these residuals to extend the energy function of Equation 5.6 and minimize

$$E(p) = \mathbf{f}^T\mathbf{f} + \mathbf{h}^T\mathbf{h} + w_s \cdot \mathbf{g}^T\mathbf{g} \tag{5.10}$$

where again $\mathbf{g}$ describes the deviation from the average facial shape (cf. Equation 5.5). As mentioned before we optimize this energy function by alternately fixing the shape parameters $\mathbf{p}$ and the rigid transformation. In each iteration

we lower the regularization factor w_s and update the matrix F by computing new correspondences between the reconstructed point cloud and points on the surface of the morphable model. While the computation of new rigid parameters is analytically found by the method presented in [Hor87], the optimal shape parameters minimizing Equation 5.10 are found by minimizing the linear system

$$(S_C^T \cdot S_C + S_F^T \cdot S_F + w_s \cdot D^T \cdot D) \cdot \mathbf{p} = -S_C^T \cdot (C \cdot \bar{\mathbf{s}} + \mathbf{v}') - S_F^T \cdot (F \cdot \bar{\mathbf{s}} + \mathbf{q}')$$

where similar to Equation 5.7 we introduced the abbreviatory notations $S_F = F \cdot S$ and $\mathbf{q}' = R^T \cdot (\mathbf{t} - \mathbf{q})$.

5.5.2. Refinement using Laplace Editing

The idea of this approach is related to the fitting procedure presented in Section 5.5.1, where we increased the flexibility of the face template by reducing the influence of the regularization energy in each iterations. Here we use a more simpler approach based on Laplace mesh editing (cf. Section 2.6.1), where the flexibility of the mesh is controlled by the influence of the residual function which measures how the detail vectors (cf. Equation 2.35) of the initially fitted face template chance. Refining the face template is again an iterative process, where we increase the flexibility of the surface by decreasing the weighting factor w_L defined in the energy function

$$E = \mathbf{g}_x^T \mathbf{g}_x + \mathbf{h}_x^T \mathbf{h}_x + w_L \cdot \mathbf{f}_x^T \mathbf{f}_x$$

of the Laplace Mesh Editing technique (cf. Equation 2.38). To define the constraints we first of all use the already known vertex constraints described by Equation 2.36

$$\mathbf{g}_x = C \cdot \mathbf{p}_x - \mathbf{v}_x$$

which we obtained from the correspondences between some extraordinary vertices of the face template and the triangulated feature points $\mathbf{v}$. Again we denote $\mathbf{v}_x$ as the concatenation of all x-components of $\mathbf{v}$ as introduced in Section 2.6.1. Adding these constraints keeps the model globally at the right place and avoids the face template to slide under the reconstructed point cloud. Then we additionally introduce face constraints, which are supposed to drag

a surface points (lying within triangles of the face model) towards a point of the reconstructed point cloud. Thereby we omit correspondences if the closes point w.r.t. a reconstructed point does not fall into the handle region of our face model (cf. Section 5.5). As described in Section 2.6.1 these correspondences define a residual function of the form

$$\mathbf{h}_x = F \cdot \mathbf{p}_x - \mathbf{t}_x$$

where $\mathbf{t}_x$ encodes all x-components of points from the point cloud used to establish the point to triangle correspondences. The same notation is used to define the concatenation $\mathbf{v}_x$ of the x-components of the triangulated feature points. The complete algorithm that refines the deformation of the face template with increasing flexibility and that incorporates the computation of optimal rigid parameters iterates the following steps:

1. Update the residuals $\mathbf{h}$ by establishing point to triangle correspondences between point of the reconstructed point cloud and triangles of the handle region of the face template
2. Find an optimal rigid transformation $(R, \mathbf{t})$ using the method of Horn [Hor87]
3. Rotate the original Laplace vectors of the initial face template using R and denote the concatenation of the x-components of the adapted detail vectors as $L \cdot \mathbf{q}_x$
4. Solve the linear system

$$(C^T C + F^T F + w_L \cdot L^T L) \cdot \mathbf{p}_x = C^T \mathbf{v}_x + F^T \mathbf{t}_x + w_L \cdot L^T L \cdot \mathbf{q}_x$$

   to obtain new vertex position $\mathbf{p}$ of our face template.
5. Decrease w_L by a constant offset of increase the flexibility of the face template.

Using the CHOLMOD [CDHR08] solver in Step 4, we need to compute only one Cholesky factorization of the matrix of the left hand side of the equation. Then three forward and back substitutions [CDHR08] efficiently lead to the x,y and z coordinates of the new vertex positions $\mathbf{p}$.

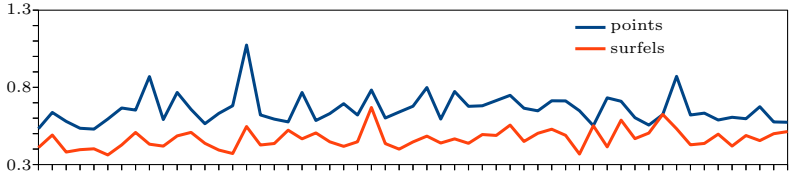


Figure 5.13.: The plot shows for each of the 55 reconstructions, the verage point (respectively splat) distance (in mm) to the deformed faces template. Both methods produce average errors below one millimeter.

5.5.3. Results

We visually compare the adaption of the face template to the point respectively surfel cloud by using the parameters computed during the calibration process (cf. Section 2.4.3) to accurately project the final face template into one image taken by our camera rig. Each pair in Figure 5.14 shows the two versions of our (deformed) face template. In the left image, the face template was refined using a morphable model (cf. Section 5.5.1) before it was deformed by one single step of the Laplace editing technique. By adding this last step we also allow to reconstruct faces not representable by the morphable model. The right image of each pair shows the result of the refinement described in Section 5.5.2. For this we use the second version of the face template which covers a larger area of the face. While the inner part is precisely reconstructed we sometimes have problems in regions where an insufficient number of points or surfels were generated. This usually happens in the region below the chin, if the person was slightly looking down. To resolve this issue, it is necessary to add more cameras to the rig.

To quantize the difference between the point (respectively surfel) cloud, we compute for each reconstructed point the point closest to the deformed face template. From this we compute the average distance for each of the 55 reconstructed faces, which is visualized in Figure 5.13. As we saw in Section 5.4.3 the point based reconstruction method generally produces lower NCC values, which should lead to additional noise in the resulting point cloud. The plot in Figure 5.13 confirms this assumption, since it shows that the average distance between the reconstructed points and the deformed face template is generally

Figure 5.14.: Visual comparison of both reconstruction methods. After performing the 3 steps to obtain the deformed face template (initialization, reconstruction and refinement) it is projected into an image, used for the reconstruction. Both methods produce accurate matches of the captured face.

higher than the average distance between reconstructed surfels and the face template. Nevertheless, for both methods the average distance between point (respectively splat) cloud is still below one millimeter.

Part II.

Dynamic Faces

6. Reconstructing Dynamic Faces

Given the reconstruction of a static face showing a neutral expression (cf. Chapter 5), the goal of this chapter is to deform the face template frame by frame to track a facial expression in 3D, which was captured by the synchronized cameras of our rig. Since the reconstructed movements are the basis of our dynamic morphable face model, it is mandatory that the tracking process maintains temporal correspondences.

Following the structure of Chapter 5, we describe and compare two multi-view systems, designed for this purpose. The prototypical system was presented in [SHK09, SHK11], while the second system was presented in [SK]. The second system was used to create the large database of facial expressions, needed to train our dynamic morphable face model (cf. Chapter 7).

To solve the 3D tracking problem both systems basically solve two sub-problems:

1. Tracking the surface in image space to maintain temporal correspondences (Section 6.1)
2. Deforming the face template such that the 3D surface is reconstructed and projected vertices of the face template move according to the tracking result of Step 1 (cf. Section 6.2)

In what follows we describe an approach, specially designed for faces, to solve the 2D tracking problem, which we call *mesh tracking*. A main contribution is to formulate the simple additive mesh tracking as inverse compositional mesh tracking, which incorporates smoothness constraints for the vertices of an *image mesh*. We show that the additive and the inverse compositional approach lead to reliable results, while the inverse compositional mesh tracking is computationally much more efficient. In Section 6.2 we will complete the description of both tracking systems by discussing two approaches to deform the generic face template according to the 2D tracking result and the 3D reconstruction

which is either based on surfel reconstruction (cf. Section 5.4.1) or on the fast point based reconstruction (cf. Section 5.4.2).

6.1. Video Tracking

Our capture system is majorly designed to build a database of dynamic faces. In order to analyze the resulting data in a statistical meaningful way we need the reconstructed faces to be in full correspondence. This includes geometric correspondences between individual faces and also in time when tracking the facial expressions of a specific actor. The later type of temporal correspondences can be established by applying an optical flow based image tracking [WBBP06] on each individual image sequence produced by the synchronized cameras of the calibrated camera rig. Unlike, e.g., [ZSCS04] we do not use a standard form of optical flow [HS80], where a displacement vector is computed individually for each pixel and smoothed across the image in order to get a coherent displacement field. Especially between lips and eyes, this approach will be problematic, since there the displacement vectors are likely to point into opposite directions, which would be suppressed by the smoothing term.

We rather use a mesh based tracking approach [SHK11, HDH*10]. Between two successive frames of a video sequence, this approach computes for every vertex of a two dimensional mesh a displacement vector such that an energy term is minimized, which is based on the formulation of the optical flow condition. We call this two dimensional mesh *image mesh* and initialize it from the surface reconstructed at the first frame of the video. Since the position of eyes and mouth are known in this reconstruction, we can explicitly model the gaps between the eye lids and the mouth, and thereby deal with the potentially discontinuous movements at the borders of these gaps.

In what follows we will describe the tracking process only for one of the four videos produced by our camera rig. In practice we use the described tracking approach to establish temporal correspondences for all four sequences.

6.1.1. Initialization of the Image Mesh

To automatically generate a traceable image mesh for an image sequence, we employ the 3D reconstruction of the neutral face, which is obtained from the

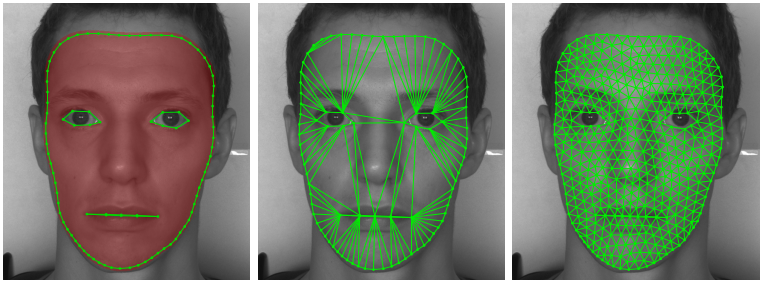


Figure 6.1.: Generation of an image mesh. Left: The projection of the initially reconstructed face template leads to the outline of the image mesh. Middle: A constraint Delaunay triangulation places triangles at the area covered by the face and leaves out holes for the eyes and the mouth. Right: Re-meshed triangulation with uniform edge lengths.

first images simultaneously taken by our camera rig (cf. Section 5.5). Using the OpenGL rendering pipeline [AMHH08] the triangles of the adapted face template are rasterized and projected into the first image of the sequence using the projective function of the respective camera. This leads to a partition of the image content into background and foreground pixels, while the later are associated to the facial surface. The red area in Figure 6.1 shows an example of the foreground obtained by this procedure.

From the foreground we identify the boundary of the image mesh we want to generate. Therefore we search for all boundary pixels, which are those red pixels having a non red neighboring pixel. Starting at an arbitrary boundary pixel, a 2D polygon is constructed by successively conquering neighboring boundary pixels until the starting point is reached again. Since the face template has holes for the eyes and the mouth, this might lead to smaller boundaries enclosing those regions. To identify the outer boundary of the facial region, we simple identify the polygon with the longest border.

To robustly compute the afore mentioned 2D vertex displacements, the image mesh has to fulfill some quality criterions. In this way triangles must not be degenerated or cover only a small image area below a few pixels. To ensure a good triangle quality, the outer boundary is resampled according to a target

edge length, we wish to reach for the image mesh. The result of the border detection and its resampling is shown in the left image of Figure 6.1. In this figure we also see the boundary of the eyes and the mouth, which can explicitly be obtained by projecting the contour of these region into the image and resample them according to the target edge length.

Once the green polygons (cf. Figure 6.1), representing the boundaries of the image mesh are constructed, the task is to distribute the inner vertices such that the triangles are uniformly shaped and the edges have approximately the desired target edge length. We achieve this by first constructing a constrained Delaunay triangulation [Che87]. Therefore we add the vertices of the boundaries to a Delaunay triangulation and declare two successive vertices of a boundary polygon to form a fixed edge in the constrained Delaunay triangulation. The result of the process is depicted in the middle image of Figure 6.1, where we already removed faces, which do not belong to the inner part of the facial region.

This gives us a triangulation for the boundary polygons, but not a triangulation with the desired quality, since triangles can have stretched shapes or cover large image areas, which makes them impossible to use for this tracking approach. We solve this, by isotropically re-mesh this initial 2D mesh [BKP*10]. We fix the vertices of the boundary polygons and iteratively split edges significantly longer and collapse edges significantly shorter than the target edge length. In each iteration we additionally smooth the inner vertices [BKP*10] using uniform edge weights. The result of this process is depicted in the right image of Figure 6.1, which shows that the resulting mesh has uniformly distributed triangles and that the initially constructed boundaries are preserved.

This resulting image mesh is formally defined by $M = (\mathbf{s}, T)$, where analogous to Section 2.3.2, T encodes the topology of the image mesh by a set of triangles connecting the n vertices with positions

$$\mathbf{s} = (x_1, y_1, \ldots, x_n, y_n)^T$$

6.1.2. Tracking the Image Mesh

As indicated above the objective is to compute for two successive frames I^f and I^{f+1} a set of displacement vectors $\mathbf{d} = (d_{x,1}, d_{y,1}, \ldots, d_{x,n}, d_{y,n})^T$ from which

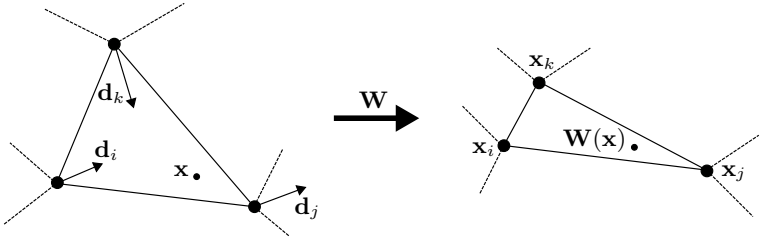


Figure 6.2.: Barycentric coordinates of a point $\mathbf{x}$ are constant in the reference (left) and the deformed triangle (right). This allows to define a warping function $\mathbf{W}(\mathbf{x})$ that maps points from the reference to the deformed triangle, given the new vertex positions $\mathbf{x}_i, \mathbf{x}_j$ and $\mathbf{x}_k$.

we can update the old vertex positions $\mathbf{s}^f$ of frame f to obtain the vertex positions in frame $f+1$:

$$\mathbf{s}^{f+1} = \mathbf{s}^f + \mathbf{d}$$

As a first step, the image mesh at frame f is rasterized according to the resolution of the underlying image I^f. For each pixel $\mathbf{x} = (x, y)$ we thereby identify the triangle $t = (i, j, k)$, where $\mathbf{x}$ lies in, and evaluate its barycentric coordinates (α, β, γ) w.r.t to that triangle. As described in Section 2.3.2 the barycentric coordinates of $\mathbf{x}$ define a matrix $B_{t,\mathbf{x}} \in \mathbb{R}^{2\times 6}$ that maps displaced vertex positions of the triangle to a warped pixel position (cf. Figure 6.2)

$$\mathbf{W}(\mathbf{x}) = \underbrace{\begin{bmatrix} \alpha_{t,\mathbf{x}} & 0 & \beta_{t,\mathbf{x}} & 0 & \gamma_{t,\mathbf{x}} & 0 \\ 0 & \alpha_{t,\mathbf{x}} & 0 & \beta_{t,\mathbf{x}} & 0 & \gamma_{t,\mathbf{x}} \end{bmatrix}}_{:=B_{t,\mathbf{x}}} \cdot \begin{pmatrix} x_i \\ y_i \\ x_j \\ y_j \\ x_k \\ y_k \end{pmatrix}$$

Introducing a selection matrix N_t (cf. Equation 2.17) we can write this warp in terms of all displaced vertex positions $\mathbf{s}^{f+1}$ or rather in terms of the displacements $\mathbf{d}$ as

$$\mathbf{W}(\mathbf{x}; \mathbf{d}) = B_{t,\mathbf{x}} \cdot N_t \cdot (\mathbf{s}^f + \mathbf{d}) \tag{6.1}$$

With this warping function we can evaluate the quality of the vertex displacements by comparing the image intensities $I^f(\mathbf{x})$ with the image intensities $I^{f+1}(\mathbf{W}(\mathbf{x};\mathbf{d}))$.

Additive Mesh Tracking In [SHK09, SHK11] we used an approach similar to the simple Additive Image Alignment (cf. Section 2.3.1). In this approach image similarities are measured by the residual functions

$$\mathbf{f}(\mathbf{d}) = \mathbf{I}^{f+1}(W(\mathbf{d})) - \mathbf{I}^f$$

where we again use the abbreviate notation for the concatenation of the image intensities evaluated at all rasterized pixel positions. To optimize this nonlinear function, we follow the approach of the Additive Image Alignment and compute the Taylor expansion of $\mathbf{f}(\mathbf{d} + \Delta\mathbf{d})$ to obtain the Jacobian J, which has rows of the form

$$J_{i,\cdot}(\mathbf{d}) = \nabla I_i^{f+1} \cdot B_{t,\mathbf{x}} \cdot N_t$$

As described in Section 2.3.1, the residual function and its derivative are used to iteratively improve the parameters. In this case these parameters are the vertex displacements $\mathbf{d}$, which deform the mesh of frame f to obtain the mesh in frame $f+1$. We start the optimization by initializing the displacements as $\mathbf{d} = \mathbf{0}$. In each iteration the displacements are improved by an update $\Delta\mathbf{d}$, which is the solution of the linear system

$$J^T J \Delta\mathbf{d} = -J^T \mathbf{f} \tag{6.2}$$

This means, in each iteration we compute new displacement vectors $\mathbf{d}' = \mathbf{d} + \Delta\mathbf{d}$ by simple additions, which are used to replace the current displacements $\mathbf{d}$.

Unfortunately, in practice this approach can lead to degenerated configuration with sliver triangles or even fold-overs. Since this makes further updates unstable to compute, we introduce a second residual function $\mathbf{g}$, which measures how the displacement of a vertex i deviates from the displacements of its neighbors:

$$\mathbf{g}_i = w_i \cdot \mathbf{d}'_i - \sum_{j \in N(i)} w_{i,j} \cdot \mathbf{d}'_j \tag{6.3}$$

As quickly noticeable, this function is based on the discrete Laplace operator for triangle meshes [DC76], which depends on the weights $w_{i,j}$ between the center

vertex i and its neighbors. Here we defined $N(i)$ to be the set of vertices in the one-ring neighborhood of the vertex i and compute w_i as $w_i = \sum_{j \in N(i)} w_{i,j}$. As elaborated in [WMKG07] there are different ways to define the weights $w_{i,j}$ all having different advantages and disadvantages. Since an image mesh is embedded into the image plane, we deal with a simple case and use the chordal weights

$$w_{i,j} = ||\mathbf{x}_i - \mathbf{x}_j||$$

to define the Laplace operator. When arranging these weights in each row of a matrix $L \in \mathbb{R}^{2n \times 2n}$, we can write the residual function in terms of the updated displacements as $\mathbf{g} = L \cdot \mathbf{d}' = L \cdot (\mathbf{d} + \Delta\mathbf{d})$. Instead of optimizing the energy function $E = \mathbf{f}^T\mathbf{f}$, we now optimize the energy function

$$E = \mathbf{f}^T\mathbf{f} + w_s \cdot \mathbf{g}^T\mathbf{g}$$

where w_s is a fixed, user defined parameter, that controls the influence of the smoothing energy. The derivative of $\mathbf{g}^T\mathbf{g}$ w.r.t. $\Delta\mathbf{d}$, leads to two additional terms which we use to expand the left and the right hand side of Equation 6.2:

$$(J^TJ + w_s \cdot L^TL)\Delta\mathbf{d} = -J^T\mathbf{f} - w_s \cdot L^TL\mathbf{d}$$

Solving this linear system of equations in each iteration and adding the update to the current solution follows the *Newton Method* presented described in Section 2.1. In order to further stabilize the optimization we perform a full *Levenberg Marquard* optimization by augmenting the linear system with a $2n \times 2n$ diagonal matrix λI. This leads to the final linear system

$$(J^TJ + w_s \cdot L^TL + \lambda I)\Delta\mathbf{d} = -J^T\mathbf{f} - w_s \cdot L^TL\mathbf{d}$$

we use to incrementally compute the update and thereby perform the mesh tracking from frame to frame.

Inverse Compositional Mesh Tracking As common for the Additive Image Alignment the main disadvantage of the Additive Mesh Tracking is, that we need to re-evaluate the Jacobian, which depends on the image gradients at the warped pixel positions, in each iteration. This makes this approach computationally quite involved and even if we precompute gradient images, finding

the vertex displacements from one frame to its successor, often takes several seconds.

In this thesis we dealt with a large database of videos showing facial expressions and so it was important to have an efficient implementation of this tracking approach. Therefore we adopted the Inverse Compositional Image Alignment for this purpose, which has the advantage that the Jacobian J can be precomputed and is constant from one frame to its successor frame.

The main Idea of the Inverse Compositional Image Alignment is to compute incremental updates $\Delta\mathbf{d}$ for the image mesh at frame f instead of the image mesh at frame $f+1$ (cf. Section 2.3.1). This means that in each iteration of the optimization we find a $\Delta\mathbf{d}$ minimizing the residual function

$$\mathbf{f}(\mathbf{d}) = \mathbf{I}^f(W(\Delta\mathbf{d})) - \mathbf{I}^{f+1}(W(\mathbf{d})) \tag{6.4}$$

Now we compute the Taylor expansion of the residual function at the point $\mathbf{d} = \mathbf{0}$ instead of an arbitrary point $\mathbf{d}$, which leads to a constant Jacobian

$$J_{i,\cdot}(\mathbf{0}) = \nabla I_i^f \cdot B_{t,\mathbf{x}} \cdot N_t$$

evaluated at the fixed reference pixel positions at frame f.

When remembering the steps of the Inverse Compositional framework:

1. Compute the error image $\mathbf{f} = \mathbf{I^f} - \mathbf{I^{f+1}}(\mathbf{W}(\mathbf{d}))$
2. Solve $J^T J \Delta\mathbf{p} = -J^T\mathbf{f}$
3. Update the warp: $\mathbf{W}(\mathbf{x};\mathbf{d}) \leftarrow \mathbf{W}(\mathbf{W}^{-1}(\mathbf{x};\Delta\mathbf{d});\mathbf{d})$

we notice (Step 3), that the computation of the updated displacements $\mathbf{d}'$ depends on the composition of the inverse warp $\mathbf{W}^{-1}(\mathbf{x};\Delta\mathbf{d})$ with the current warp $\mathbf{W}(\mathbf{x};\mathbf{d})$. Therefore we first need to apply the inverse warp to the reference vertex location $\mathbf{x}_i$ of the image mesh in frame f to obtain intermediate vertex locations

$$\mathbf{x}_i' = \mathbf{W}^{-1}(\mathbf{x}_i;\Delta\mathbf{d}) = \mathbf{x}_i - \Delta\mathbf{d}_i$$

As depicted in Figure 6.3 an incremented vertex position falls into a neighboring triangle t, where we can compute its barycentric coordinates as

$$\begin{pmatrix} \alpha \\ \beta \\ \gamma \end{pmatrix} = \begin{bmatrix} \mathbf{x}_i & \mathbf{x}_{t,1} & \mathbf{x}_{t,2} \\ 1 & 1 & 1 \end{bmatrix}^{-1} \cdot \begin{pmatrix} \mathbf{x}_i - \Delta\mathbf{d}_i \\ 1 \end{pmatrix} \tag{6.5}$$

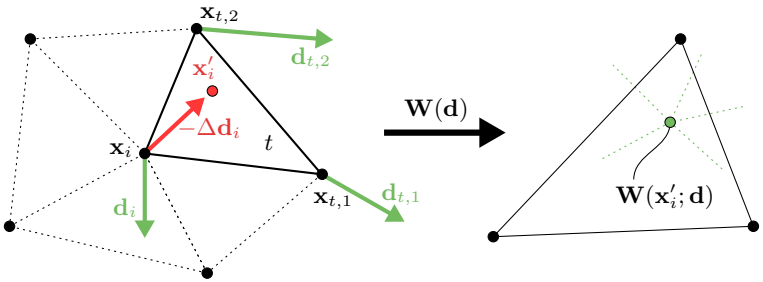


Figure 6.3.: Composition of the inverse warp with the current warp. The incrementally displaced vertex of the reference mesh falls into any of the surrounding triangles (red dot). According to the current displacement of that triangle (green arrows), which was computed in previous iterations of the optimization, the new vertex location (green dot) can be computed. In general all triangles can be used to define the warp to the new (green) vertex location. In Equation 6.7 we therefore average the predictions of all neighboring triangles.

The displaced vertices $[\mathbf{x}_i + \mathbf{d}_i, \mathbf{x}_{t,1} + \mathbf{d}_{t,1}, \mathbf{x}_{t,2} + \mathbf{d}_{t,2}]$ of the neighboring triangle t, which have been iteratively computed so far to deform the image mesh towards frame $f + 1$, are interpolated by these barycentric coordinates. This leads to the displaced vertex position $\mathbf{x}_i + \mathbf{d}'_i$ we are searching for:

$$\begin{aligned} \mathbf{x}_i + \mathbf{d}'_i &= \alpha(\mathbf{x}_i + \mathbf{d}_i) + \beta(\mathbf{x}_{t,1} + \mathbf{d}_{t,1}) + \gamma(\mathbf{x}_{t,2} + \mathbf{d}_{t,2}) \\ &= [M_t|\mathbf{m}_t] \cdot \begin{pmatrix} \mathbf{x}_i - \Delta\mathbf{d}_i \\ 1 \end{pmatrix} \end{aligned}$$

where the matrix $[M_t|\mathbf{m}_t] \in \mathbb{R}^{2\times 3}$ abbreviates the multiplication of the displaced triangle vertices with the inverse matrix occurring in Equation 6.5. Since the reference vertex locations $\mathbf{x}_i$ and the current estimate of the displacements $\mathbf{d}_i$ are known, the matrix $[M_t|\mathbf{m}_t]$ is constant for each neighboring triangle t. Now we can express the improved vertex displacement $\mathbf{d}'_i$ as a simple linear combination where

$$\mathbf{d}'_i = -M_t \cdot \Delta\mathbf{d}_i - \mathbf{x}_i + M_t \cdot \mathbf{x}_i + \mathbf{m}_t \tag{6.6}$$

Similar to the update procedure for the Active Appearance Models in Section 2.3.2, we do not know in which neighboring triangle the vertex will fall

into. This means that we do not know which barycentric warp to use in order to compute the improved displacement $\mathbf{d}'_i$ (cf. Equation 6.6). We use the same idea as in Section 2.3.2, and average the contributions of the individual warps defined by the displacements of the surrounding triangles. If $T(i)$ is the set of triangles adjacent to the vertex i, this means that we can use Equation 6.6 multiple times to compute the new (averaged) displacements $\mathbf{d}'_i$ as

$$\mathbf{d}'_i = \underbrace{\left[\frac{1}{|T(i)|}\sum_{t\in T(i)} -M_t\right]}_{A_i} \cdot \Delta\mathbf{d}_i \underbrace{-\mathbf{x}_i + \frac{1}{|T(i)|}\sum_{t\in T(i)} M_t \cdot \mathbf{x}_i + \mathbf{m}_t}_{\mathbf{b}_i}$$

where we introduce the matrix $A_i \in \mathbb{R}^{2\times 2}$ and the two dimensional vector $\mathbf{b}_i$ to write this update in a compact form of a linear equation $\mathbf{d}'_i = A_i \cdot \Delta\mathbf{d}_i + \mathbf{b}_i$. Since A_i and $\mathbf{b}_i$ only depend on the vertex location $\mathbf{x}_i$ of the image mesh at frame f and the displacement $\mathbf{d}_i$, computed so far during the optimization, they need to be evaluated in every iteration. To encode the updates of all vertices at once we define a block diagonal matrix $A \in \mathbb{R}^{2n\times 2n}$, where the i^{th} 2×2 block equals A_i, and assemble a vector $\mathbf{b} \in \mathbb{R}^{2n}$ by concatenating the vectors $\mathbf{b}_i$. Then the improved displacements (Step 3 of the Inverse Compositional framework) can be computed as

$$\mathbf{d}' = A\Delta\mathbf{d} + \mathbf{b} \tag{6.7}$$

With this simple linear equation it is easy to extend the Inverse Compositional Mesh Tracking by an additional smoothing term. To the residual functions defined in Equation 6.4, we therefore add residual functions of the form

$$\mathbf{g} = L \cdot (A\Delta\mathbf{d} + b)$$

which are similar to the ones defined in Equation 6.3. Deriving $\mathbf{g}^T\mathbf{g}$ w.r.t. the updates $\Delta\mathbf{d}$ leads to two additional terms for the left and the right hand side of the linear system, we need to solve in each iteration. With these additional terms, the update rule in Equation 6.7 and a damping matrix $Id \in \mathbb{R}^{2n\times 2n}$ needed for the more stable *Levenberg Marquard* optimization, we can summarize the iterative steps needed to perform the Mesh Tracking from frame f to frame $f+1$:

1. Compute the error image $\mathbf{r} = \mathbf{I^f} - \mathbf{I^{f+1}}(\mathbf{W}(\mathbf{d}))$
2. Solve $[J^T J + w_s A^T L^T L A + \lambda I d]\Delta\mathbf{d} = -J^T\mathbf{r} - A^T L^T L\mathbf{b}$
3. Update the warp: $\mathbf{d} \leftarrow A\Delta\mathbf{d} + \mathbf{b}$

Iterations are stopped if the change in displacements are small or if a maximum of iteration numbers have been reached.

6.1.3. Results

In Figure 6.4, we compare the results of both mesh tracking approaches by showing the deformed image mesh overlaid to four images captured at different time points. We initialized both tracking approaches with the exact same image mesh in order to visually compare the deformed meshes. The full video sequence contains 77 images and shows the transition from a neutral to a happy expression and from the happy expression back to the neutral face. The result of the additive mesh tracking is shown in the first row of Figure 6.4, while the result of the inverse compositional mesh tracking is depicted in the second row. To better rate this result we additionally show close-ups: The third row shows the close-up for the additive mesh tracking, while the forth row shows the result of the inverse compositional mesh tracking in more detail. Since both approaches visually produce quite similar results, we additionally investigated the distances between corresponding vertices of the computed image meshes. We color-coded this result in the thumbnail images between the first and second row. While in most areas the deviation is around one pixel (the images have a resolution of 780×580), the deviation sometimes becomes larger when the image mesh needs to be deformed significantly. The reason for this is, that although both approaches should be equivalent from a mathematical point of view, the practical implementation is a bit different. More concrete, the update of the displacement vectors are computed from the composition of the old warping function with an inverted incremental update. In practice we approximate this composition using Equation 6.7 and average the warping functions defined for the triangles incident to a specific vertex. This is in fact an additional smoothing operation, which might have influence on the stiffness of the image mesh. This in turn would lead to the observed, slightly different tracking behavior.

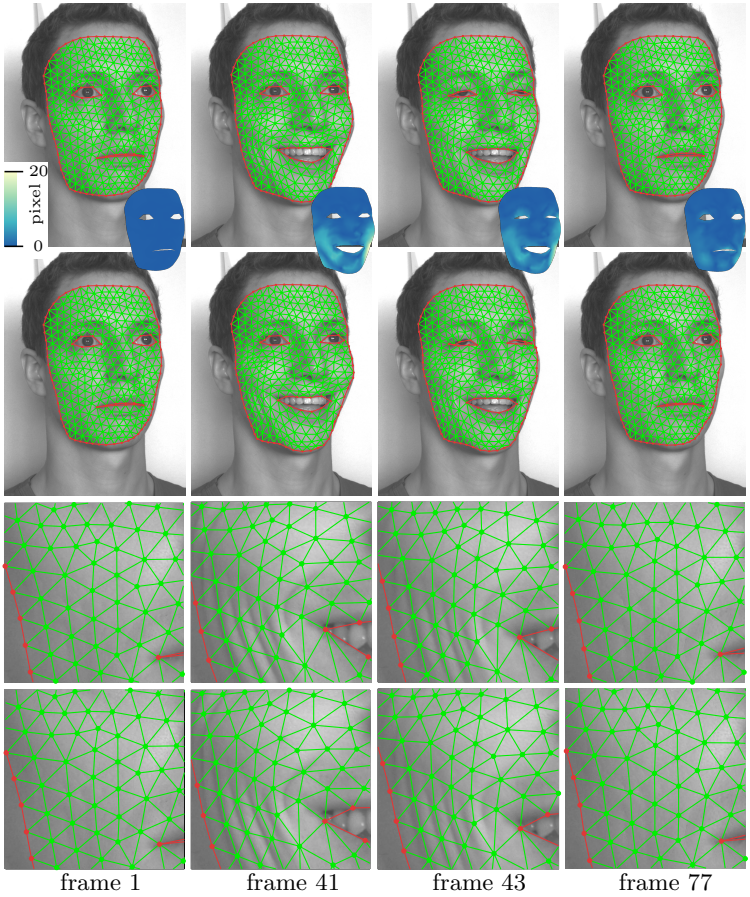


Figure 6.4.: Tracking result for a video sequence with 77 frames. First row (and third row): image meshes computed by the additive mesh tracking. Second row (and fourth row): result of the inverse compositional mesh tracking.

In general both tracking approaches work very reliable, even if large displacements are involved. The images of the second and third column of Figure 6.4 were taken within an interval of 50 ms. Although the eyes were blinking both

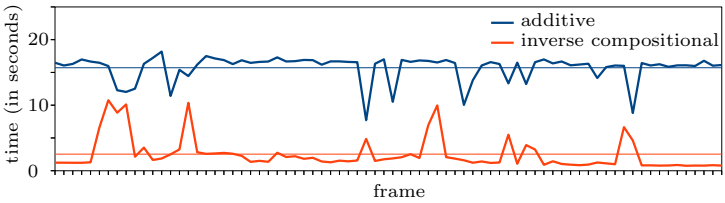


Figure 6.5.: Timings for each frame of a video sequence to compute the vertex displacements for two successive frames. The inverse compositional mesh tracking is significantly faster than the additive method.

approaches do not produce significant drift and the vertices of the image mesh at the last frame of the video (fourth column) are nearly at the same positions as at the beginning of the sequence (first column).

The main advantage of the inverse compositional mesh tracking lies in the algorithmic complexity. In Figure 6.5 we show the timings for then whole sequence containing 77 images. As we can see, for all frames the inverse compositional approach is significantly faster. It needs on average 2.5 seconds while the additive approach need on average over 15 seconds to compute the vertex displacements between two successive frames. When we neglect the outliers, this difference becomes even more significant: In nearly all frames the inverse compositional approach needs less than 2 seconds to converge, while the additive approach needs more than 15 seconds to compute the update.

6.2. Surface Tracking

In Section 5 we described two approaches to reconstruct static faces showing a neutral facial expression. In this section we follow the comparative description of the two systems and extend both approaches by a dynamic component and explain how we track dynamic facial expressions. In order to increase the robustness and the performance of the surfel based tracking approach [HK06], which will be detailed in Section 6.2.1, we propose a new tracking approach that is based on the simple and fast point based stereo reconstruction and that computes a scene flow [WBV*11] to dive the deformation of our face template. With this system, which will be detailed in Section 6.2.2, we were able to auto-

matically reconstruct a large database of dynamic facial expressions from video sequences captured by our camera rig. This database plays a central role to build our dynamic face model, which is later used to track and reconstruct facial expressions as seen in videos taken with only a single camera (cf. Chapter 7).

6.2.1. Surfel based Tracking

The surfel based tracking is initialized with the reconstruction result produced by the algorithm described in Section 5.5. Then, the objective is to find for every frame of the video sequence a deformed version of this initial mesh. Finally the sequence of the deformed face templates represent the detailed movements of the captured face as seen in the synchronized videos produced by the camera rig. In what follows we use *frame* and *view* to notate an image taken from a specific camera (view) at a specific point in time (frame).

The basic principle of this approach is to distribute 2D samples in all views at the first frame and track them through time using the 2D tracking approach presented in Section 6.1. Then *surfel fitting* (cf. Section 2.5.3) is used in every frame to reconstruct a 3D surfel for each image-sample. Finally, the reconstructed 3D surfels form trajectories in 3D space and are used to deform the face template and thereby track the observed facial movements.

Generating 2D trackable samples Applying the mesh-based tracking approach described in Section 6.1 to the videos captured from every view, produces a sequence of 2D triangle meshes $M_c^f = (\hat{\mathbf{s}}_c^f, T_c)$ for each view c and frame f, where we use the notation $\hat{}$ to indicate the two dimensional character of point positions. Supersampling the triangles of the meshes M_c^1 (first frame of each sequence), generates 2D points that have barycentric coordinates w.r.t. the triangle they are placed in. For every view c this leads to a set of points $\hat{\mathbf{p}}_{i,c}^1$ for the first frame, where a point can uniquely be identified by its index i and the view c it was placed in (cf. Figure 6.6). For simplicity from now on we use the more compact notation $\hat{\mathbf{p}}_c^f$ for an image-sample and omit its index. The number of image-samples is a user-defined parameter and we usually place 1600 samples in one view to later obtain a dense reconstruction. As stated in Equation 6.1 the barycentric coordinates of $\hat{\mathbf{p}}_c^1$ w.r.t. its associated triangle $t \in T_c$ define a matrix B_t. Together with the matrix N_t that selects the vertices

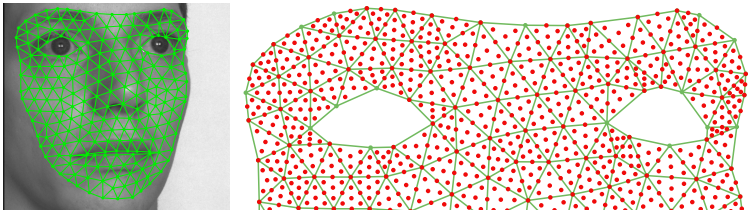


Figure 6.6.: Uniformly distributed image samples. Each sample is associated to a specific triangle of the image mesh, where it has barycentric coordinates. By this the image samples can be warped to any frame, if mesh tracking was performed over all images of a video.

adjacent to the triangle t and with the vertex positions $\hat{\mathbf{s}}_c^f$ of the image mesh at frame f, the mapping from frame 1 to frame f for this sample point is uniquely defined as

$$\hat{\mathbf{p}}_c^f = B_t \cdot N_t \cdot \hat{\mathbf{s}}_c^f \tag{6.8}$$

Employing this equation on every initially placed sample and every frame leads to 2D trajectories $\hat{\mathbf{p}}_c^1$, $\hat{\mathbf{p}}_c^2$, $\hat{\mathbf{p}}_c^3, \ldots$. In what follows we compute 3D surfels from these 2D points in order to obtain 3D trajectories that drive the deformation of our face template.

Surfel-based tracking step. The aim of one tracking step is to deform the reconstructed face template of frame $f-1$ to obtain a reconstruction at frame f. For this, the 2D trajectories computed from Equation 6.8, play a central role to establish temporal correspondences between the successive reconstructions. We start the surfel-based tracking by computing the initial 3D mesh $M^1 = (\mathbf{s}^1, T)$ from the views of the first frame as described in Section 5.5. Here, $\mathbf{s}^f$ encode the 3D vertex positions of the mesh reconstructed at frame f. For each image-sample $\hat{\mathbf{p}}_c^1$, we introduce an anchor point on the surface of M^1 by shooting a ray through $\hat{\mathbf{p}}_c^1$ (cf. Equation 2.25) and determining the intersection with one of the triangles of M^1. As depicted in Figure 6.7, this anchor point is a specific point on the surface and is uniquely defined by the intersected triangle $t \in T$ and the barycentric coordinates $[\alpha, \beta, \gamma]$ w.r.t. the vertices adjacent to that triangle. For any deformed version of the face template we now can compute

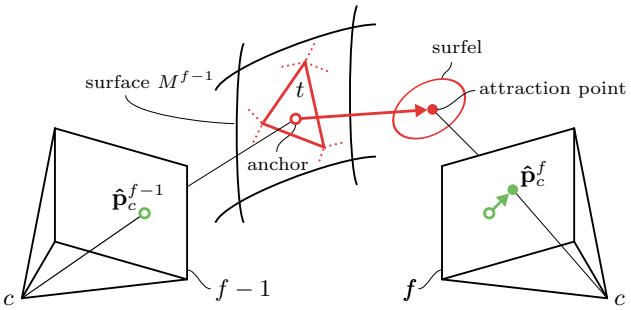


Figure 6.7.: An anchor point of a image-sample $\hat{\mathbf{p}}_c^1$ is obtained by intersecting a ray with the initial face template. During tracking this anchor point is draged towards its attraction point, which is computed by displacing the associated 2D sample in image space and reconstrucing its depth value using surfel fitting.

the 3D location of the anchor point by using the barycentric coordinates to combine the tree vertex positions encoded in $\mathbf{s}^f$.

We apply the *surfel fitting* technique to reconstruct small surface patches from the points of the 2D trajectories. As described in Section 2.5.3, for this we need a few ingredients: (a) the 2D position of the center of the correlation window within the reference view, (b) a set of comparison views and (c) a plane equation to initialize the optimization process. To reconstruct the surface element in frame f we employ $\hat{\mathbf{p}}_c^f$ to be the position of the reference correlation window, which makes its associated view c the reference view. To find the set of comparison images, we follow the approach presented in Section 5.4.1 and render the previously reconstructed mesh M^{f-1} to fill OpenGL's depth buffer [AMHH08]. This allows to efficiently check from which view of our camera rig the anchor point associated to the sample position $\hat{\mathbf{p}}_c^{f-1}$ if visible. In case the anchor point is not visible from the reference view, or the number of comparison images is zero, we discard the reconstruction of the surface element for this frame. To get the last ingredient for the *surfel fitting* technique we initialize the normalized plane equation $\mathbf{n}^f$ from the position of the anchor and the normal of the triangle, which are associated to the image-sample $\hat{\mathbf{p}}_c^{f-1}$ (cf. Figure 6.7). Then $\mathbf{n}^f$ is iteratively optimized (cf. Equation 2.34) which leads

to a new 3D position and orientation of the surface patch we tracked from frame $f-1$ to frame f. Shooting a ray through the pixel $\hat{\mathbf{p}}_c^f$ and computing its intersection with the reconstructed surface patch $\mathbf{n}^f$ (cf. Figure 6.7) leads to the position where the anchor point of $\hat{\mathbf{p}}_c^{f-1}$ should move to according to the 2D tracking from $\hat{\mathbf{p}}_c^{f-1}$ to $\hat{\mathbf{p}}_c^f$ and the 3D reconstruction for frame f.

Doing this for all visible image-samples leads to a set of 3D points where anchor points should be attracted to. We simulate this attraction using Laplace mesh editing (cf. Section 2.6.1). Since each (reconstructed) attraction point is associated to an anchor point, we can define face constraints for Laplace mesh editing analogous to Equation 2.37

$$\mathbf{h}_x = F \cdot \mathbf{s}_x^f - \mathbf{a}_x$$

where identical to Section 2.6.1, $\mathbf{a}_x$ encodes the concatenation of all x-components of the attraction points and F is the matrix storing the barycentric coordinates of the anchor point at the columns corresponding to the vertices of the face the anchor point is located on. Together with the basic constraint to maintain the original Laplace vectors $L \cdot \mathbf{s}_x^1$, where $\mathbf{s}_x^1$ are the concatenated x-components of the vertex positions of the initial mesh M^1, we can compute new vertex positions $\mathbf{s}_x^f$ by solving the linear system (cf. Equation 2.39)

$$(w_t \cdot F^T F + L^T L) \cdot \mathbf{s}_x^f = w_t \cdot F^T \cdot \mathbf{a}_x + L^T L \cdot \mathbf{s}_x^1$$

Notice that we need to solve two similar systems to additionally get the y and z-components of the new vertex positions encoding the shape of the face template M^f at frame f.

6.2.2. Scene flow based Tracking

Surfel based reconstruction tends to be more accurate than point based reconstructions (cf. Section 5.4.2), since it warps reference pixels perspectively correct to a pixel position in the comparison image. Nevertheless it can become unstable, since small deviations in the plane equation (parameters) can lead to large displacements and distortions of the correlation windows in the comparison images and thereby to huge variations of the energy function. In addition to this the optimization procedure is computationally quite involved and by

this the presented tracking system is tedious to use when reconstructing facial expressions from nearly 100.000 images as we did to construct our database of facial movements (cf. Chapter 7).

In order to overcome the time limitations and some robustness issues of the system we used in [SHK09, SHK11], we designed a new reconstruction system for facial expressions for the purpose to automatically reconstruct facial expressions from a very large set of videos produced by our camera rig. The first key improvement was the replacement of the *Additive Mesh Tracking* by the computationally more efficient *Inverse Compositional Mesh Tracking*, which is described in detail in Section 6.1. The second key improvement is the implementation of a computationally more efficient surface extraction, which is based on the GPU assisted point cloud reconstruction (cf. Section 5.4.2). From the reconstructed 3D surface, together with the 2D displacements obtained from the mesh tracking, we compute a dense scene flow, that drives the evolution of the face temple over time. In what follows we describe how we reconstruct the surface in a particular frame and how to combine this with the 2D motion vectors to obtain a dense field of 3D motion vectors between successive frames, that describes the scene flow. In the last paragraph we complete the description of the new tracking approach by using the motion vectors to deform the face template.

Surface extraction The fundamental technique for this reconstruction method is the point based reconstruction method already used in Section 5.4.2 to obtain a dense point cloud. The only difference to compute the point cloud regards the initialization step: Unlike for static faces, where we initialize the surface from triangulated facial feature points (cf. Section 5.3), we use the face template of the previous frame encoded by the mesh $M^{f-1} = (\mathbf{s}^{f-1}, T)$ to initialize the face template at frame f. Then we perform the same reconstruction steps involving (a) the pairwise rectifications of images, (b) the identification of corresponding pixels using normalized cross correlation (c) filtering the resulting depth map including smoothing and outlier removal operations. An example of the resulting point cloud can be seen in Figure 6.8 (b).

The computation of a 3D motion vector from any point in frame $f-1$ to a 3D point in frame f involves, as we have seen in Section 6.2.1, the projection

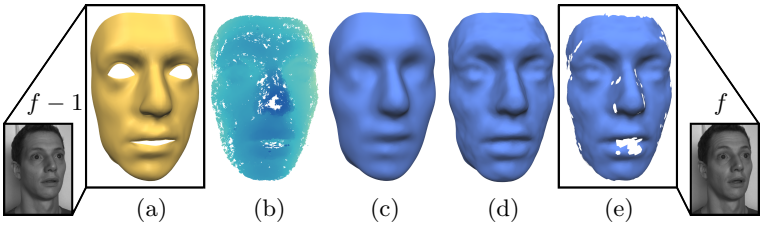


Figure 6.8.: Surface extraction for a new frame f of the video sequence. (a) To initialize depth values for the point based stereo reconstruction we use the reconstruction of the previous frame $f-1$. (b) Point cloud reconstruction. (c) The smooth surface approximation is used to detect outlier in the point cloud. (d) Surface which was tightly fitted to the remaining points. (e) 3D surface of the new frame, where triangles in purely reconstructed regions have been removed.

of an image point (which was tracked in 2D from frame $f-1$ to frame f) back to the surface. This back-projection is in principle the computation of the intersection of a ray with the point cloud, which can be solved using the moving least squares technique [ABCO*01, FCOS05]. Since this involves the approximation of small surface patches, the technique can sometimes become unstable in regions with low point density. In this work we prefer a more global approach to extract the surface from the given point cloud and tightly fit a mesh to the point cloud. Therefore we first of all close all holes of our face template [ACK13] and use isotropic re-meshing [BKP*10] to convert M^{f-1} to a mesh M', where an edge has approximately the length of the average distance between neighboring points of the point cloud (the topology of this mesh is visualized, e.g., in Figure 6.8 (c)). Second of all we perform the Laplace Mesh Editing two times in order to compute a deformed version of M' that tightly approximates the input point cloud.

Since the point cloud was pre-filtered, in the first step we use every point of the cloud to establish point-to-triangle correspondences to M' (cf. Equation 2.37). Then, using a large weighing factor for the part that reconstructs the local detail vectors of M' (cf. Equation 2.35) we compute new vertex positions from the point-to-triangle constraints (see Section 2.6.1 for details). This

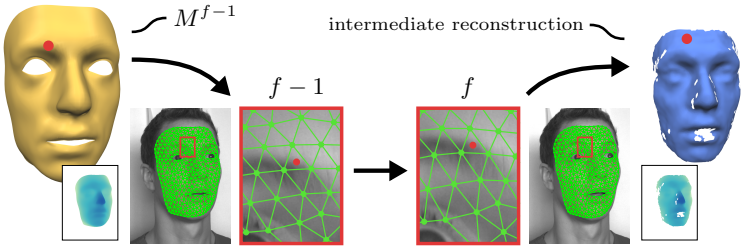


Figure 6.9.: Scene flow computation. A vertex of the deformed face template at frame M_c^{f-1} (left, orange surface) is projected to an image and tracked to the next frame f (middle). Then, the 3D coordinates of the sample at frame f is computed by intersecting a viewing ray with the intermediate reconstruction (right, blue surface).

first deformation is already close to the point cloud as seen in Figure 6.8 (c). We improve this result by computing an error statistic on the distances between all points an the deformed surface and discard those point-to-triangle correspondences which have a distance significantly larger than the standard deviation of the error distribution. By this we thin out the correspondences and only use correspondences for the second deformation step, which likely are not outliers. Since the remaining point-to-triangle correspondences are more reliable, we decrease the weighting factor for the reconstruction of the detail vectors (cf. Equation 2.35) and allow for a less smooth but more interpolating surface obtained by the second deformation step as shown in Figure 6.8 (d). In some regions, e.g., with specular highlights, the reconstruction of points can fail, which leaves some holes in the point cloud. It is likely that triangles in those regions are never used for a point-to-triangle correspondence. If this is the case, we consider the triangles to lie in such a badly reconstructed region and delete them from the intermediate surface reconstruction, such that M', used in subsequent steps of our tracking approach, will have some holes as depicted in Figure 6.8 (e).

Scene flow computation. Again the basis to compute 3D vectors that represent the motion of particular surface points, is the extraction of a 2D flow field for all videos taken from the different views. For this we use the *Inverse Compositional Mesh Tracking*, that was detailed in Section 6.1 and extract, analogous to Section 6.2.1, for each view a sequence of image meshes $M_c^f = (\hat{\mathbf{s}}_c^f, T_c)$ representing the 2D motion of surface points as seen in the video. To ease visibility checks, we render the face template M^{f-1} reconstructed in the previous frame $f-1$ to obtain depth images as depicted in Figure 6.9.

Assume a point $\mathbf{p}^{f-1} \in \mathbb{R}^3$ is located on the surface of M^{f-1}. As a first step to compute its motion to a new position we identify the views where $\mathbf{p}^{f-1}$ projects to a valid pixel location. A pixel of an image of view c at frame $f-1$ is considered to be valid, if it lies on a triangle of the image mesh M_c^{f-1} (cf. red area in Figure 6.9) and if the distance between camera center and $\mathbf{p}^{f-1}$ is not larger that the interpolated depth value at this location (visibility check). Let c be the view where $\mathbf{p}^{f-1}$ projects to a valid pixel location $\hat{\mathbf{p}}_c^{f-1} \in \mathbb{R}^2$. Since this point lies in a triangle of the image mesh at frame $f-1$ we can again define a matrix B_t holding the barycentric coordinates and a matrix N_t that selects the vertices of the triangle. Analogous to Equation 6.8 we displace this point by using the vertex locations $\hat{\mathbf{s}}_c^f$ of the image mesh at frame f which leads to

$$\hat{\mathbf{p}}_c^f = B_t \cdot N_t \cdot \hat{\mathbf{s}}_c^f$$

Shooting a ray through the new point position $\hat{\mathbf{p}}_c^f$ and intersecting it with the surface M_r^f, we already reconstructed for this frame (cf. Figure 6.9), leads to the new location of the surface point $\mathbf{p}^f$ we are searching for. Since the surface M_r^f can contain holes as described before, it is not guaranteed that there is an intersection between ray and surface. In this case we would consider $\mathbf{p}^{f-1}$ as an invalid point for estimating the local motion of the surface.

In some cases there might be multiple views which produce valid motions for a single point. In those cases we simply compute the average of the positions predicted by the different views. This has also the advantage that errors of the motion estimate can be reduced.

Deformation of the face template. The deformation of the face template now follows the approach described in Section 6.2.1 and uses Laplace mesh editing

(cf. Section 2.6.1) to deform the surface M^{f-1} by the estimated scene flow. But in this setting we use the more simpler vertex constraints to drive the deformation of the face template: For each vertex of the modifiable area of the template (cf. Figure 5.5) we try to estimate the motion vector as described before. If this leads to a valid position the vertex should move to according to the scene flow, we constrain the vertex to this new position, which leads to a set of constraints analogous to Equation 2.36

$$\mathbf{g}_x = C \cdot \mathbf{s}_x^f - \mathbf{v}_x$$

where $\mathbf{v}_x$ encodes all x-components of the positions the constraint vertices are moving to and C is the matrix selecting the constraint vertices. Again we include the basic requirement that all vertices should maintain their original detail vectors (cf. Equation 2.35) and solve the linear system

$$(w_v \cdot C^T C + L^T L) \cdot \mathbf{s}_x^f = w_v \cdot C^T \mathbf{v}_x + + L^T L \cdot \mathbf{s}_x^1$$

which is similar to the one stated in Equation 2.39.

6.2.3. Results

In what follows we show numerous examples for reconstructions produced by our tracking systems. Surfel based tracking (cf. Section 6.2.1) uses a morphable model to initialize the reconstruction for the first frame (cf. Section 5.3.1). The topology of the morphable model equals the smaller face template, which covers only a small portion of the face (cf. first and third rows of Figure 6.10 - 6.14). The second and forth row show the result of our scene flow based tracking. As discussed earlier we encounter sometimes problems for the larger face template, if an insufficient number of points were reconstructed in some regions as, e.g., under the chin. Then the face template does not perfectly approximate the facial shape, which can additionally result in a bad surface initialization from which the 3D reconstruction for the next frame becomes rather unstable. Since this is only the case if the person is slightly looking down (cf. first example in Figure 6.12) this could be fixed by attaching additional cameras to the rig with focus on the problematic regions.

To be able to visually inspect the reconstruction quality, we use the exact same camera parameters computed during the calibration phase to project the

final results into the images. In case the sequence was successfully reconstructed the results are visually very similar. In practice the surfel based approach is a bit more likely to fail, since it requires a better initialization than point based reconstruction, which searches for the optimal correspondence in a larger depth interval (cf. Section 5.4.1). In addition to this point based reconstruction is much faster as investigated in Table 5.4.3. All previously presented experiments regarding timings and quality and the results presented in the Figures 6.10 - 6.14 encourage us to summarize the following system design, optimal to extract the database needed in Chapter 7:

1. Initialize the face template for the first frame of a video sequence following the steps presented in Section 5.3.2, Section 5.4.2 and Section 5.5.2
2. Perform the inverse compositional mesh tracking for all views (cf. Section 6.1)
3. Deform the face template using the reconstructed scene flow (cf. Section 6.2.2)

This system needs for one frame on average around 11 seconds (a) to track image meshes in images produced by four synchronized cameras, (b) to compute the scene flow involving the 3D reconstruction of over 300.000 points and (c) to perform the deformation of the face template to approximate the facial shape shown in the next frame. In contrast to this the surfel based approach needs 6.5 seconds on average to only compute a much sparser surfel cloud. In [SHK09, SHK11] we additionally used the additive mesh tracking, which leads on an Intel®Core™i7-4770 with an NVIDIA GeForce GTX 770 to approximately 70 seconds for one frame.

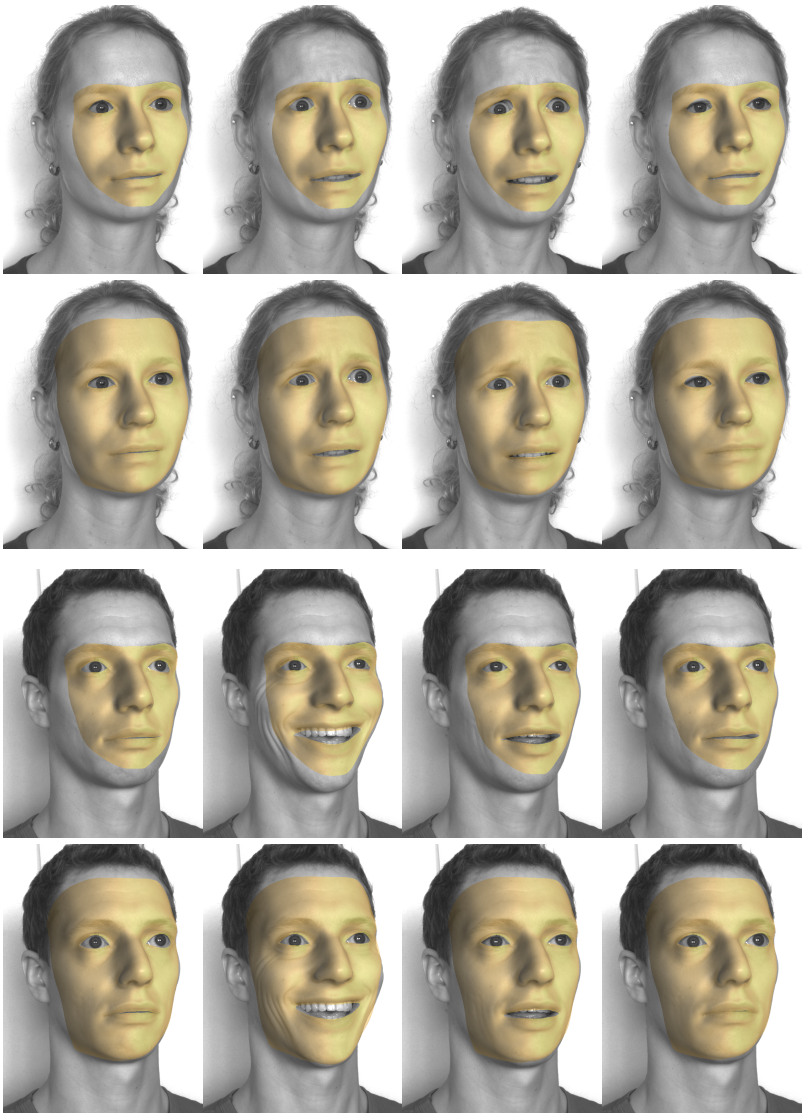


Figure 6.10.: Results produced by the proposed reconstruction systems.

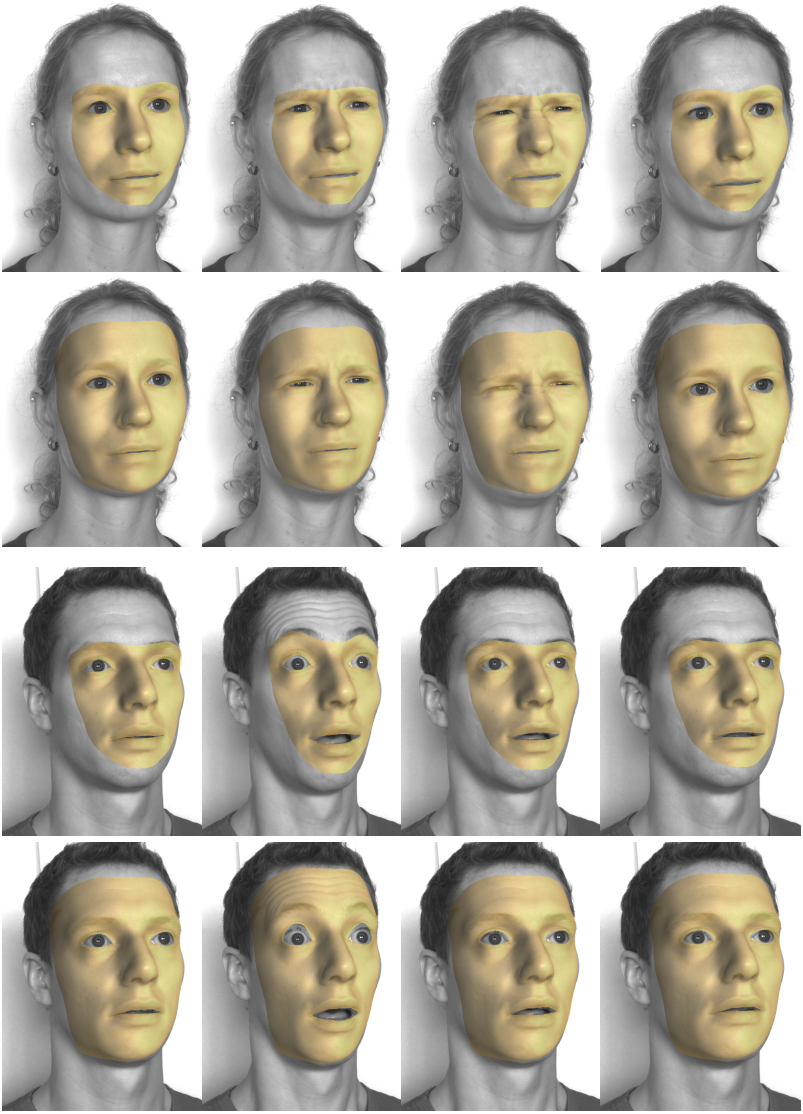


Figure 6.11.: Results produced by the proposed reconstruction systems.

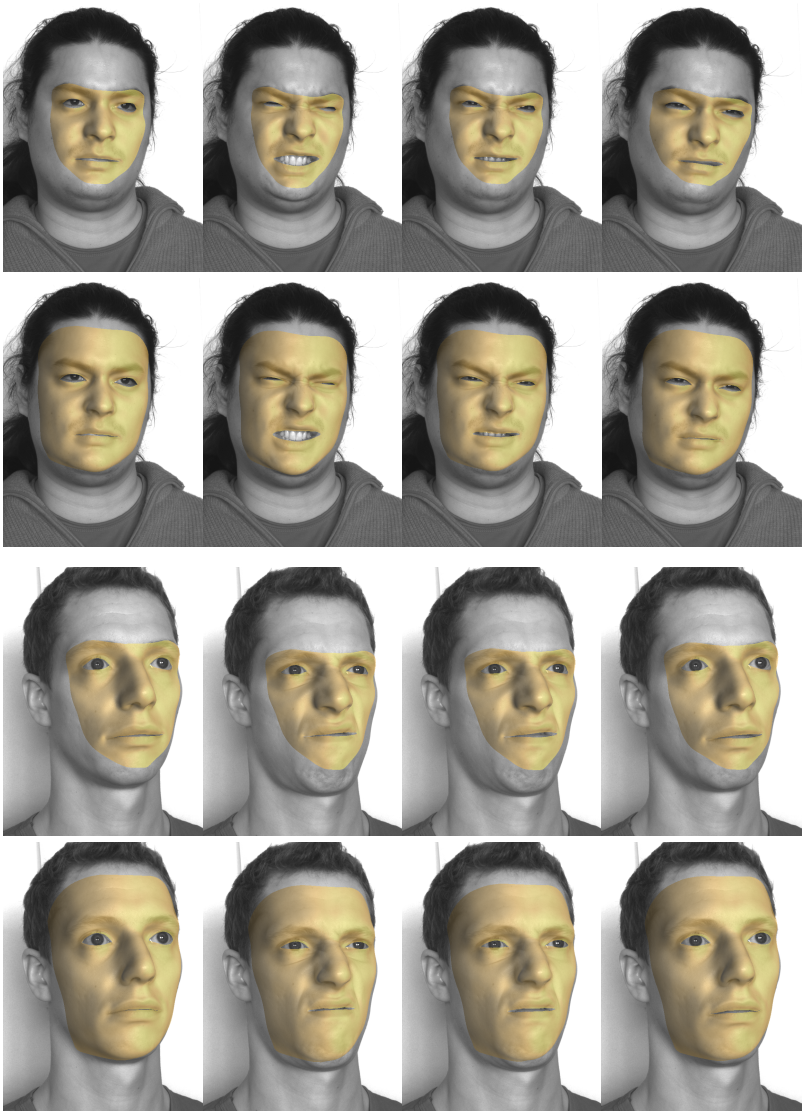


Figure 6.12.: Results produced by the proposed reconstruction systems.

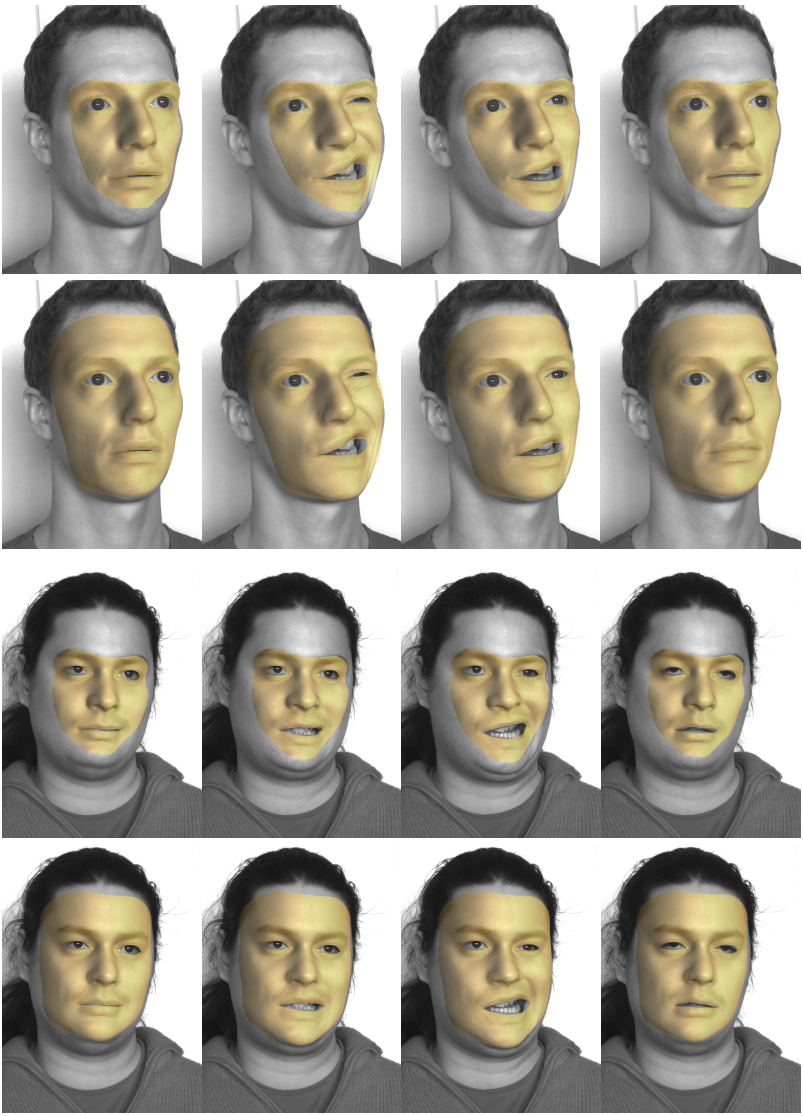


Figure 6.13.: Results produced by the proposed reconstruction systems.

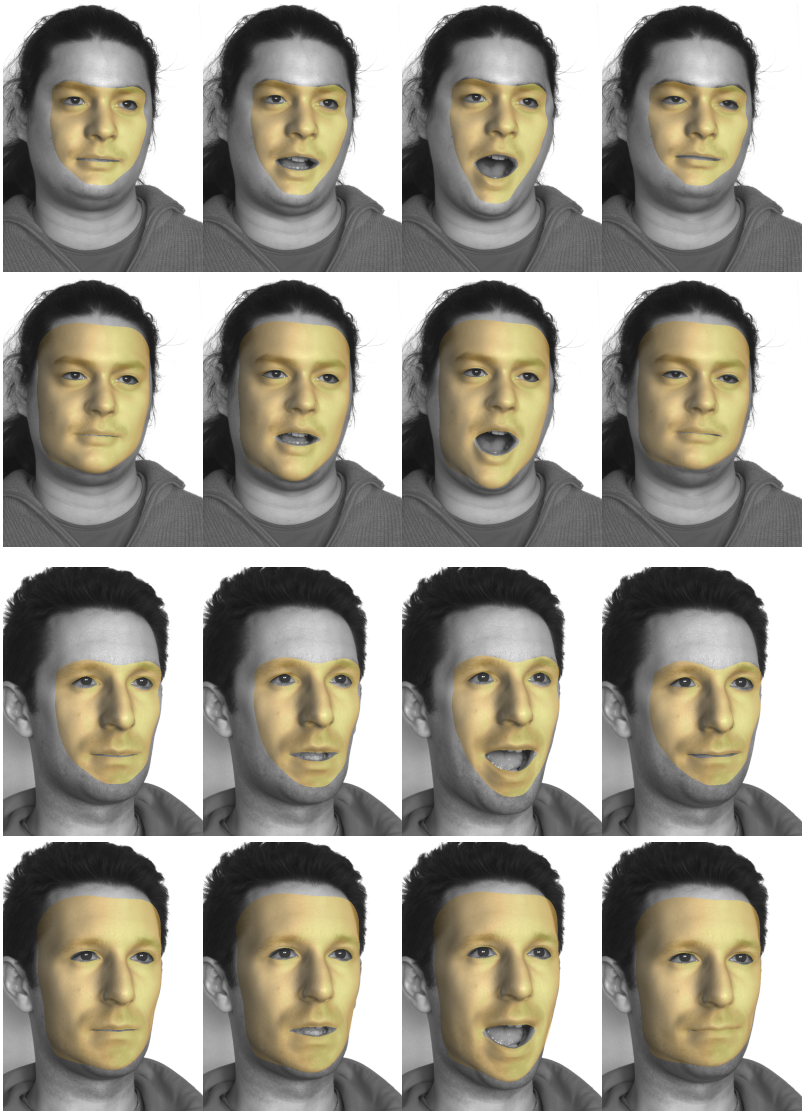


Figure 6.14.: Results produced by the proposed reconstruction systems.

6.3. Applications

Since we decided to use a predefined face template with a fixed mesh topology, we obtain meshes for each frame which are in full correspondence to each other. This enables a variety of applications. We demonstrate some of these applications introduced in [SHK11] and explain them in more detail in this section.

6.3.1. Eliminating rigid transformations

Due to head movements the captured face will certainly show some rigid transformations w.r.t. the first frame. For the applications described in the following sections it is desirable to separate these rigid transformations from the deformations.

Assuming the face does not scale, we compute a translation and rotation of the face in each frame w.r.t. the face shown in the first frame. For this we employ a variant of the algorithm provided by Horn [Hor87]. Note that the correspondences do not need to be calculated since vertex indices of our face template remain constant across frames. In general one can take all vertices into account to calculate the rigid transformations, but due to highly deformable regions on the face (like cheeks, forehead or mouth) we just consider vertices lying on the more rigid nose region. Advantages of this step are that we can easily replace the rigid transformation with different transformations, without changing the deformation itself.

6.3.2. Automatic model enhancement

Since we are able to remove rigid transformations the positions of eyes, bones, etc. remain constant across frames. Together with the fixed topology of the moving face template, this allows for the automatic placement of eyes, eye lids and lashes. In this section we describe a very simple way to model eyes. Note that this does not produce realistic looking eyes, but can be seen as a proof of concept for such an automated modeling procedure, which connects the boundary curve of the eye to lids and lashes.

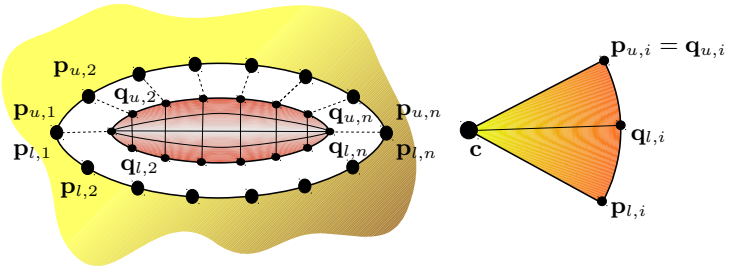


Figure 6.15.: Left: The eyelid and the boundary of the eye have the same amount of vertices, so the eyelid can be stitched to the top boundary. Right: The closing t determines the position of each lower point $\mathbf{q}_{l,i}$ on the eyeball. If $t = 1$ the lid covers the whole eye, i.e., $\mathbf{p}_{l,i} = \mathbf{q}_{l,i}$.

As depict in Figure 6.15, the boundary around an eye consists of a lower curve $\{\mathbf{p}_{l,1}, \ldots, \mathbf{p}_{l,n}\}$ and an upper curve $\{\mathbf{p}_{u,1}, \ldots, \mathbf{p}_{u,n}\}$. We model the eyeball as a simple sphere with a radius $r = 12$ mm, which is the average radius of a human eye [Kat98]. The center $\mathbf{c}$ of the eye is placed such that the following function is minimized:

$$F = \sum_{i=l,u} \sum_{j=1}^{n} (\mathbf{c} - \mathbf{p}_{i,j})^2 - r^2$$

subject to the constraints

$$||\mathbf{p}_{i,j} - \mathbf{c}|| \geq r$$

In order to model a lid we created a small polygon mesh with an upper and lower curve containing the vertices $\{\mathbf{q}_{u,1}, \ldots, \mathbf{q}_{u,n}\}$ and $\{\mathbf{q}_{l,1}, \ldots, \mathbf{q}_{l,n}\}$. This mesh can easily be connected to the upper boundary curve by setting $\mathbf{q}_{u,i} = \mathbf{p}_{u,i}$ (see Figure 6.15). We compute aperture angles for each pair of vertices on the boundary of the eye as

$$\theta_i = \arccos\left(\frac{(\mathbf{p}_{u,i} - \mathbf{c})^T(\mathbf{p}_{l,i} - \mathbf{c})}{||\mathbf{p}_{u,i} - \mathbf{c}|| \cdot ||\mathbf{p}_{l,i} - \mathbf{c}||}\right)$$

The positions of the lower points are calculated by interpolating the aperture angle. Let $t \in [0, 1]$ be the parameter controlling the closing of the eye. The

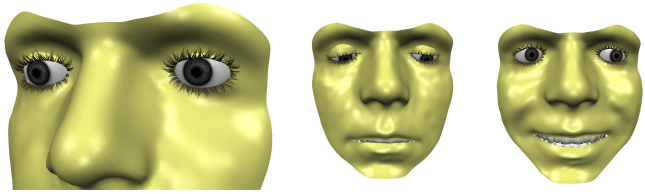


Figure 6.16.: Simple eye model with lid and lashes. The closing of the lid depends on the viewing direction of the eye: The lower vertices of the lid are close to the upper point of the iris. In this example we also modeled teeth.

lower points of the lid are computed as points lying on the eyeball (Figure 6.15) such that

$$t \cdot \theta_i = \arccos\left(\frac{(\mathbf{p}_{u,i} - \mathbf{c})^T(\mathbf{q}_{l,i} - \mathbf{c})}{||\mathbf{p}_{u,i} - \mathbf{c}|| \cdot ||\mathbf{q}_{l,i} - \mathbf{c}||}\right)$$

Note that the eye is closed, i.e., $\mathbf{q}_{l,i} = \mathbf{p}_{l,i}$, if $t = 1$. The inner vertices of the lid are smoothly distributed over the eyeball.

Lashes can be attached to the lower points $\mathbf{q}_{l,i}$ and its mesh is textured with a semitransparent image of eyelashes. The result of our automatic modeling example is depicted in Figure 6.16. Here we implemented a simple look-at function for the eyes which rotates the eyeballs. The opening of the lid is calculated such that the lower points of the lid are close to the upper point of the iris. In this way the lid closes when the eye is looking down. Further modeling steps could be the automatic placement of denture or the integration into a head model.

6.3.3. Automatic Expression Modeling

In this section we demonstrate versatile applications where expressions, i.e., the deformation of faces, themselves are manipulated. As in Section 6.3.2, correspondences between different faces and across frames are crucial for these applications. As a basic technique we use a variant of the deformation transfer for triangle meshes which was originally introduced by Sumner [SP04].

Figure 6.17.: Expression transfer. The smile $M^1, \ldots, M^F$ of one face is transferred to another subject to produce a sequence $\mathcal{T}^1, \ldots, \mathcal{T}^F$.

Expression transfer In the area of facial animation there are many different approaches for re-targeting facial expressions [NN01, NJ04, XCLT14], which usually involve some kind of fitting approach. Our capture system ensures the faces to be in full inter-subject as well as temporal correspondence such that we can use the formalism of deformation transfer (cf. Section 2.6.3) to carry an expression from one face over to another. Assume we used the markerless reconstruction technique to generate a sequence of meshes $M^1, \ldots, M^F$ from F frames. For each frame f a global deformation gradient matrix $\mathbf{S}^f \in \mathbb{R}^{3m \times 3}$ can be calculated which describes the deformation $M^1 \rightarrow M^f$. In order to transfer this expression sequence to another face $\mathcal{T}^1$ we simply compute a new sequence of F deformed meshes $\mathcal{T}^f$ with new vertex positions $\mathbf{p}_i^f$:

$$\begin{pmatrix} \mathbf{p}_1^{f\,T} \\ \vdots \\ \mathbf{p}_n^{f\,T} \end{pmatrix} = \underbrace{(\mathbf{G}^T\mathbf{D}\mathbf{G})^{-1}\mathbf{G}^T\mathbf{D}}_{=:\mathbf{K}(\mathcal{T}^1)}\mathbf{S}^f$$

where $1 < f \leq F$ and $\mathbf{K}(\mathcal{T}^1) \in \mathbb{R}^{n \times 3m}$ is a precomputed matrix only depending on $\mathcal{T}^1$ (cf. Equation 2.42). In Figure 6.17 we transferred the smile of one subject to another face.

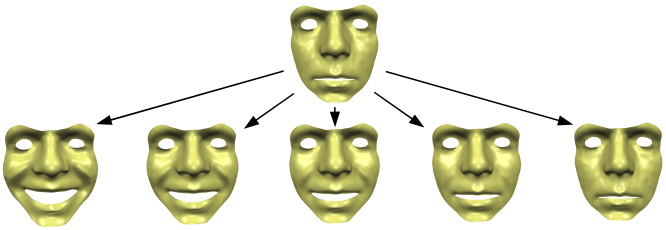


Figure 6.18.: Expression attenuation. The upper image shows the face at time point zero. The images in the lower row show attenuated versions of one of the expressions shown in Figure 6.17(top) at a particular time point > 0. To generate the images from left to right, we set α to 1, $\frac{3}{4}$, $\frac{1}{2}$, $\frac{1}{4}$ and 0.

Expression attenuation In the previous section we used the deformation gradients $\mathbf{S}^f$ from a source sequence to generate a new target sequence showing the same expression as the source sequence. When using the identity matrix $\mathbf{I}_3$ as source deformation gradients per triangle, the target mesh would not change at all. By linearly blending between both possibilities we can attenuate the expression. We start with a face $\mathcal{T}^1$ in neutral position which might be identical to M^1 and compute new vertex positions $\mathbf{p}_i^f$ as

$$\begin{pmatrix} \mathbf{p}_1^{fT} \\ \vdots \\ \mathbf{p}_n^{fT} \end{pmatrix} = \mathbf{K}(\mathcal{T}^1) \left(\alpha \cdot \begin{pmatrix} \mathbf{S}_1^{fT} \\ \vdots \\ \mathbf{S}_m^{fT} \end{pmatrix} + (1-\alpha) \cdot \begin{pmatrix} \mathbf{I}_3 \\ \vdots \\ \mathbf{I}_3 \end{pmatrix} \right)$$

where $\alpha \in [0, 1]$. In Figure 6.18 we show a smile in four different intensities by setting α to 0, $\frac{1}{3}$, $\frac{2}{3}$ and 1.

Expression blending In this scenario we compute a weighted sum of deformation gradients from different subjects and compose a new sequence from the sum of these weighted gradients. Besides the geometric correspondence we therefore also need temporal correspondence, i.e., the same number of frames in each sequence. In our experiments the subject performed facial expressions

Figure 6.19.: Expression blending. Left: Three 'smile' sequences captured by our system. Right: Each row shows images from a blended sequence, where $\mathcal{T}^1$ is the average shape of (M_1^1, M_2^1, M_3^1). The first row averages the smile of (M_1, M_2, M_3), the second row averages (M_2, M_3) and in the third and fourth row one can see the average of (M_1, M_3) and (M_1, M_2).

from neutral to an expressed face. Since this took almost the same time for every subject, we immediately get temporal coherence by a simple cut, common in video editing, without interpolating between frames or scaling time.

For expression blending we consider k source sequences $M_i = \{M_i^1, \ldots, M_i^F\}$ where $i \in 1, \ldots, k$. For each frame f in every sequence i we precompute a deformation gradient $\mathbf{S}_i^f \in \mathbb{R}^{3m \times 3}$ which describes the deformation $M_i^1 \rightarrow M_i^f$. Starting with a target mesh $\mathcal{T}^1$, which might be a new individual showing a neutral face, a blending between faces like done by a morphable model or simply the first mesh of one of the source sequences, we compute a new sequence from the convex combination of the deformation gradients. This generates new meshes $\mathcal{T}^f$ with new point positions:

$$\begin{pmatrix} \mathbf{p}_1^{fT} \\ \vdots \\ \mathbf{p}_n^{fT} \end{pmatrix} = \mathbf{K}(\mathcal{T}^1) \left(\sum_{i=1}^{k} w_i \cdot \mathbf{S}_i^f \right)$$

where $\sum_{i=1}^{k} w_i = 1$, $w_i \geq 0$ are some given weights. In Figure 6.19 we show the result of such a convex combination of the source deformation gradients.

7. Face Tracking from Single Images

In Chapter 5 and Chapter 6 we described systems to reconstruct dynamic facial expressions using a multi-camera rig. In general this approach produces high quality and detailed 3D surfaces, but unfortunately it needs a rather complex setup with calibrated cameras and laboratory lighting conditions. Therefore it can hardly be used in consumer level applications, where often only a single camera, like e.g. a webcam, is available. From the usability point of view a system would be preferable which tracks 3D faces from simple videos captured with only one consumer level camera.

In this Chapter we propose such a system which captures the facial performance of an actor from simple 2D images. Since this problem is ill-conditioned, we use the motion data contained in a large database of different facial expressions as a deformation prior to stabilize the tracking process. Different to other approaches we derive a deformation model from the database, which separates the facial movements from the individual face geometries. During the tracking this automatically ensures that we do not blend between different faces as a conventional shape model would do, but only deform an individual face according to our deformation model. One big advantage of this is, that the computed facial deformations can directly be applied to other faces, which instantaneously enables re-targeting of facial animations. Furthermore we analyze the deformation data in our database and compute a general time dependent movement model for facial expressions which is used as a temporal prior for facial movements. Similar to a shape model [BV99], which is able to reconstruct plausible facial shapes not contained in the input database, this movement model is also applicable to new persons and requires no individual training per person. The most related approach is the system proposed by Weise et al. [WBLP11]. But instead of using an active sensing method based on Microsofts Kinect [Zha12], our capture system is completely image based. Additionally we use a complete different motion model to predict plausible facial movements.

In what follows we present details of our database of facial expressions in Section 7.1.1 before we introduce the *deformation space* for faces and explain how the deformation parameters are related to the appearance of a face (cf. Section 7.1.2). In Section 7.2 we detail the actual tracking process which tries to find deformation parameters and a rigid transformation such that projected feature points are close to feature points observed by a 2D Active Appearance Model (cf. Section 2.3.2) and such that the previously reconstructed motion sequence is continued in a plausible way according to our motion model. In Section 7.3 we show how the computed deformation can be used for facial re-targeting and present results in Section 7.4. Parts of this chapter previously appeared in [SK15].

Contributions

- Instead of optimizing shape parameters and thereby blending between different faces, we run our optimization in a deformation space representation, which decouples the dynamic motion from the individual shapes of the faces, and which makes re-targeting of facial animations particularly easy.
- Our system is purely image based and computationally not involved. It can easily be integrated in a facial feature tracker like AAMs, which runs at high frame rates even on mobile devices.
- Our system uses a general deformation and motion model derived from a large database. It does not need an individual training phase per persons, which makes it interesting for simple consumer level applications.

7.1. Deformation Space

Our tracking approach is based on a new representation of facial expressions: Each expression is encoded as a deformation of the neutral reference face. The set of all possible deformations of all possible reference faces is called *deformation space*. In what follows we describe our input database, which samples the deformation space and explain how deformations are modeled and converted back to actual facial shapes.

7.1.1. Database of Facial Movements

To collect the motion data for our deformation model, we used a rig of 4 cameras to capture moving faces at 40 frames per second. We recorded 60 subjects performing 20 facial expressions related to the extreme poses suggested in [EF78]. On average a facial expression took 2 seconds to perform, which leads to a total number of about 96.000 frames to be reconstructed. To robustly and automatically perform this reconstruction and tracking task for this amount of frames we used a system similar to the one suggested in [SHK11] and generated for each subject and each dynamic facial expression a sequence of meshes M_f. Here f represents the frame number within the respective dynamic facial expressions. Sliding a time window of length k over an animation with F frames we can extract $F-k+1$ animation snippets of equal length k. Each animation in our database is split into such sets of animation snippets, each containing exactly k frames.

7.1.2. Deformation Space Representation

Given a triangle mesh A and a deformed version A' having the same mesh connectivity, we can compute a deformation gradient $\mathbf{S}(t) \in \mathbb{R}^{3\times3}$ for each triangle t as presented in detail in Section 2.6.3. The precondition to compute deformation gradients is that both meshes are in full correspondence. That is why we designed our facial capture system to be able to produce the required consistent mesh topology where each mesh has m triangles and n vertices. The concatenation of all gradients to a matrix $\mathbf{S} \in \mathbb{R}^{3m\times3}$ encodes the deformation of A to A' [BSPG06].

Since deformation gradients represent motion only and factor out the underlying facial geometry, we encode each reconstructed mesh M_f in a sequence by its deformation gradient $\mathbf{S}_f$ w.r.t. the neutral pose M_0 of the respective person. We do this for all sequences of our database. The resulting matrix $\mathbf{S}_f$ for each frame can be represented as a $9 \cdot m$ dimensional vector. We feed the entire database of 96.000 frames into a Principal Component Analysis (PCA). By this we extract the average deformation gradient $\bar{\mathbf{S}} \in \mathbb{R}^{3m\times3}$ and a matrix of the l most important eigen-gradients $\mathbf{S}_{eg} \in \mathbb{R}^{3m\times3l}$ encoding the main deviations from $\bar{\mathbf{S}}$ (Fig. 7.1).

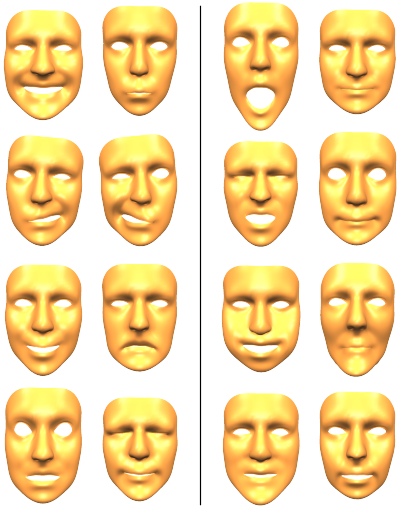


Figure 7.1.: The eight largest eigen-gradients represent meaningful facial actions. For each pair the left and the right face show the lower and upper range of a specific deformation parameter.

Given l deformation parameters $(s_1, \ldots, s_l)$, we assemble a $3l \times 3$ parameter matrix

$$\mathbf{s} = \begin{bmatrix} s_1 & & \\ & s_1 & \\ & & s_1 \\ \cdots & & \\ s_l & & \\ & s_l & \\ & & s_l \end{bmatrix}$$

and express an deformation gradient $\mathbf{S}$ as a linear combination of eigen-gradients:

$$\mathbf{S} = \bar{\mathbf{S}} + \mathbf{S}_{eg} \cdot \mathbf{s}$$

We approximate each deformation gradient $\mathbf{S}_f$ by such a low dimensional parameter matrix $\mathbf{s}_f$, represented as a point $\mathbf{s}_f = (s_{f,1}, \ldots, s_{f,l})^T$ in deformation space. In the following derivation we switch between the matrix and vector notation where appropriate, but since a deformation gradient $\mathbf{S}$ is always encoded as a matrix it will become clear from the context which notation is meant. An

animation snippet then is a trajectory in this deformation space represented as a $k \cdot l$ dimensional vector $(\mathbf{s}_1^T, \ldots, \mathbf{s}_k^T)^T$.

7.1.3. Transforming Deformations to Shapes

For our face tracking algorithm we need to be able to compute the actual geometry, i.e. the vertex positions of a mesh, given a point in deformation space. As described in [BSPG06], deformation gradients $\mathbf{S}$ computed from a pair (A, A') can be applied to a new mesh B – having the same topology as A – to produce a mesh B' showing a similar deformation. The vertex positions $\mathbf{p}' \in \mathbb{R}^{n \times 3}$ of the mesh B' are computed by solving the linear system

$$\mathbf{G}^T\mathbf{D}\mathbf{G}\mathbf{p}' = \mathbf{G}^T\mathbf{D}\mathbf{S} \ ,$$

where $\mathbf{G} \in \mathbb{R}^{3m \times n}$ and $\mathbf{D} \in \mathbb{R}^{3m \times 3m}$ are the gradient and area matrix of B. Observe that $\mathbf{G}^T\mathbf{D}\mathbf{G}$ evaluates to the well known cotangent Laplace matrix. Since this matrix does not have full rank, one normally augments the linear system with additional constraints, e.g., by fixing some vertex to a certain position in space. Instead of fixing the position of an arbitrary vertex, we suggest to use a Tikhonov regularization [PTVF07], which adds the identity matrix $\mathbf{I}$ scaled by a small value ε to the system. We select $\varepsilon = trace(\mathbf{G}^T\mathbf{D}\mathbf{G}) \cdot 10^{-6}$, which affects the solution of the system only slightly:

$$(\mathbf{G}^T\mathbf{D}\mathbf{G} + \varepsilon \cdot \mathbf{I})\mathbf{p} = \mathbf{G}^T\mathbf{D}\mathbf{S} \ .$$

Now the system is invertible and we can express the new vertex positions as a linear combination of deformation parameters $\mathbf{s}$:

$$\begin{aligned} \mathbf{p} &= (\mathbf{G}^T\mathbf{D}\mathbf{G} + \varepsilon \cdot \mathbf{I})^{-1}\mathbf{G}^T\mathbf{D}(\bar{\mathbf{S}} + \mathbf{S}_{eg} \cdot \mathbf{s}) \\ &= \mathbf{M}_{DS} \cdot \mathbf{s} + \mathbf{b}_{DS} \end{aligned}$$

with

$$\begin{aligned} \mathbf{M}_{DS} &= (\mathbf{G}^T\mathbf{D}\mathbf{G} + \varepsilon \cdot \mathbf{I})^{-1}\mathbf{G}^T\mathbf{D}\mathbf{S}_{eg} \quad \in \mathbb{R}^{n \times 3l} \\ \mathbf{b}_{DS} &= (\mathbf{G}^T\mathbf{D}\mathbf{G} + \varepsilon \cdot \mathbf{I})^{-1}\mathbf{G}^T\mathbf{D}\bar{\mathbf{S}} \quad \in \mathbb{R}^{n \times 3} \ . \end{aligned}$$

Note that the gradient and area matrices need to be derived only once from the reference mesh – i.e. the mesh of the neutral face of a specific person –

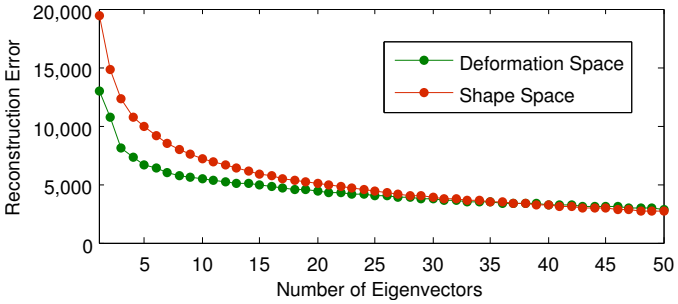


Figure 7.2.: Approximation error obtained using different numbers of eigenvectors in a standard shape space and in our suggested deformation space. For a small number of parameters the deformation space is able to approximate faces significantly better than a standard shape model.

and so $\mathbf{M}_{DS}$ and $\mathbf{b}_{DS}$ have to be precomputed once for each person but the deformation gradient model $(\bar{\mathbf{S}}, \mathbf{S}_{eg})$ itself is independent from the individual person.

A standard shape model, computed from the distribution of the vertex positions, simultaneously encodes individual shapes as well as deformations. Therefore the variance in such a model is higher than the variance of the pure deformation gradient data, where we factor out the "shape" component leading to more coherence. This in turn results in a more compact representation in deformation space and thus requires fewer parameters for the actual tracking procedure. In Figure 7.2 we compare the approximation power of the deformation gradient space with that of shape space. For this we successively used more and more eigenvectors to approximate the faces in our database and measured the error based on the vertex distances between reconstruction and original mesh. To compute the matrices $\mathbf{M}_{DS}$ and $\mathbf{b}_{DS}$ we used the mesh of the neutral face of the respective person. When employing only a few eigen-gradients, the deformation space representation actually shows a much better approximation error than a standard shape model which allows us to use fewer parameters during tracking. Thus our deformation space representation is a very natural

model for face tracking where the deformation state of the face changes over the sequence but not the underlying physiognomy.

7.2. Expression tracking

The input to our tracking procedure is a sequence of facial images. To get the initial shape of the neutral face, as seen in the first frame, one can, e.g., use an approach similar to [BV99] to optimize shape parameters of a morphable model obtained from the neutral expressions of the database. From such a reconstruction we compute the constant gradient and diagonal matrices $\mathbf{G}$ and $\mathbf{D}$, and thereby $\mathbf{M}_{DS}$ and $\mathbf{b}_{DS}$.

To each frame of our video sequence we fit an 2D AAM [MB04] in order to detect the facial features around the eyes, the nose and the mouth. This captures the individual features of the person to be tracked. The remaining part of the tracking algorithm is completely independent from the individual face. Let the 2D feature positions in the current frame f be $\mathbf{a} = (\mathbf{a}_1, \ldots, \mathbf{a}_{|a|})$. For the rigid motion we allow three degrees of freedom for the translation $\mathbf{t} = (t_x, t_y, t_z)^T$ and three degrees of freedom for the rotations, parametrized by Euler angles $\boldsymbol{\omega} = (\alpha, \beta, \gamma)^T$, such that rigid motion parameters are defined by $\mathbf{r} = (\alpha, \beta, \gamma, t_x, t_y, t_z)^T$. Given the 2D feature positions we want to compute the most probable deformation parameters $\mathbf{s}$ as well as rigid motion parameters $\mathbf{r}$, such that

1. The projections of the 3D feature points – i.e. vertices of the resulting face template corresponding to the 2D features – are close to the points $\mathbf{a}$ in image space.

2. The sequence $\tilde{\mathbf{r}} = (\mathbf{r}_{f-k+1}^T, \ldots, \mathbf{r}_{f-1}^T)^T$ of rigid motion parameters already computed for the $k-1$ previous frames is smoothly extended

3. The sequence $\tilde{\mathbf{s}} = (\mathbf{s}_{f-k+1}^T, \ldots, \mathbf{s}_{f-1}^T)^T$ of deformation parameters encoding the already computed animation of the $k-1$ previous frames is reasonably continued

Similar to Weise [WBLP11], we formulate this optimization as a maximum a posteriori estimation

$$(\mathbf{s}, \mathbf{r})^* = \arg\max_{\mathbf{s},\mathbf{r}} p(\mathbf{s}, \mathbf{r} | \mathbf{a}, \tilde{\mathbf{s}}, \tilde{\mathbf{r}})$$

where $p(\cdot|\cdot)$ represents the conditional probability. Using Bayes' rule and the independence of rigid motion and deformation we can split this into three probabilities

$$(\mathbf{s}, \mathbf{r})^* = \arg\max_{\mathbf{s},\mathbf{r}} \underbrace{p(\mathbf{a}|\mathbf{s}, \mathbf{r})}_{\text{likelihood}} \cdot \underbrace{p(\mathbf{r}, \tilde{\mathbf{r}})}_{\text{rigid prior}} \cdot \underbrace{p(\mathbf{s}, \tilde{\mathbf{s}})}_{\text{shape prior}}$$

and iteratively maximize this probability by computing the minimum of its negative logarithm:

$$E(\mathbf{s}, \mathbf{r}) = -\ln p(\mathbf{a}|\mathbf{s}, \mathbf{r}) - \ln p(\mathbf{r}, \tilde{\mathbf{r}}) - \ln p(\mathbf{s}, \tilde{\mathbf{s}}) \tag{7.1}$$

In what follows we explain the individual terms of this energy function and show how to minimize it using least squares optimization (cf. Section 2.1). Therefore we express the terms by residual functions and give their derivatives w.r.t. the parameters $\mathbf{s}$ and $\mathbf{r}$.

7.2.1. Likelihood of the AAM

Intuitively, the observed facial feature positions $\mathbf{a}_i$ are a probable explanation of the projected feature points $\mathbf{q}_i$, if the distances between those points in image space are small. This can be modeled by the probability

$$p(\mathbf{a}|\mathbf{s}, \mathbf{r}) = \prod_{i=1}^{|a|} \frac{1}{2\pi\sigma_{aam}} e^{-\frac{\|\mathbf{q}_i - \mathbf{a}_i\|^2}{(2\sigma_{aam})^2}} \quad ,$$

where σ_{aam} controls the kernel size of the probability function. The negative logarithm of this expression evaluates to an energy term of the form

$$E_{aam}(\mathbf{s}, \mathbf{r}) = \lambda_{aam} \cdot \sum_{i=1}^{|a|} \|\mathbf{q}_i - \mathbf{a}_i\|^2 = \lambda_{aam} \cdot \mathbf{g}^T \mathbf{g}$$

where λ_{aam} is a constant dependent on the kernel parameter σ_{aam} and $\mathbf{g}$ is the concatenation of the residuals $\mathbf{g}_i = \mathbf{q}_i - \mathbf{a}_i$. In order to compute the 2D

positions $\mathbf{q}_i$ we only need to project those 3D vertices $\mathbf{p}_j$ of the face model, corresponding to the facial feature points $\mathbf{a}_i$. Therefore we define a selection matrix $N \in \mathbb{R}^{|a| \times n}$ as

$$N_{i,j} = \begin{cases} 1 & \text{if } \mathbf{a}_i \text{ corresponds to } \mathbf{p}_j \\ 0 & \text{otherwise} \end{cases}$$

By taking the respective rows $N_{i,\cdot}$ we can directly compute the 3D positions of the i-th facial feature point given a set of deformation parameters

$$\mathbf{p}_j = [\ N_{i,\cdot}\ \cdot (\mathbf{M}_{DS} \cdot \mathbf{s} + \mathbf{b}_{DS})\ ]^T$$

Then the points are transformed according to the rigid motion parameters and projected into the image by using the camera's intrinsic calibration matrix $\mathbf{K} \in \mathbb{R}^{3\times3}$, such that

$$\mathbf{q}_i = \left(\frac{a}{c}, \frac{b}{c}\right)^T \tag{7.2}$$

where

$$\begin{pmatrix} a \\ b \\ c \end{pmatrix} = \mathbf{K} \cdot (\mathbf{R}_0 \cdot \mathbf{R}_\alpha \cdot \mathbf{R}_\beta \cdot \mathbf{R}_\gamma \cdot \mathbf{p}_j + \mathbf{t}) \tag{7.3}$$

In order to avoid the problem of a Gimbal lock, we compute an initial rotation $\mathbf{R}_0 \in \mathbb{R}^{3\times3}$ for the first frame of the sequence and start the tracking with $\alpha = \beta = \gamma = 0$.

Jacobian of the residual function. Since the positions $\mathbf{a}_i$ of the 2D feature points are constant, the gradient of the residual function $\mathbf{g}(\mathbf{s}, \mathbf{r})$ simply evaluates to

$$\frac{\partial \mathbf{g}_j}{\partial(\mathbf{s}, \mathbf{r})} = \frac{\partial \mathbf{q}_j}{\partial(\mathbf{s}, \mathbf{r})} \in \mathbb{R}^{2\times(l+6)}$$

where we restrict the computation to only one feature point i. With Equation 7.2 and 7.3, the gradient of the projected vertex positions w.r.t. any parameter $x \in (\mathbf{s}, \mathbf{r})$ can be computed as

$$\frac{\partial \mathbf{q}_j}{\partial x} = \begin{bmatrix} \frac{\frac{\partial a}{\partial x} \cdot c - \frac{\partial c}{\partial x} \cdot a}{c^2} \\ \frac{\frac{\partial b}{\partial x} \cdot c - \frac{\partial c}{\partial x} \cdot b}{c^2} \end{bmatrix}$$

While the derivative w.r.t. to the parameter α (the remaining parameters of the rigid motion $\mathbf{r}$ are computed analogously) is given by

$$\frac{\partial \begin{pmatrix} a \\ b \\ c \end{pmatrix}}{\partial \alpha} = \mathbf{K} \cdot \mathbf{R}_0 \cdot \frac{\partial \mathbf{R}_\alpha}{\partial \alpha} \cdot \mathbf{R}_\beta \cdot \mathbf{R}_\gamma \cdot \mathbf{p}_j$$

the derivative w.r.t. to the deformation parameters $\mathbf{s}$ evaluates to

$$\frac{\partial \begin{pmatrix} a \\ b \\ c \end{pmatrix}}{\partial s_i} = \mathbf{K} \cdot \mathbf{R}_0 \cdot \mathbf{R}_\alpha \cdot \mathbf{R}_\beta \cdot \mathbf{R}_\gamma \cdot \begin{pmatrix} \mathbf{M}_{DS}(j, 3 \cdot i) \\ \mathbf{M}_{DS}(j, 3 \cdot i + 1) \\ \mathbf{M}_{DS}(j, 3 \cdot i + 2) \end{pmatrix}$$

7.2.2. Rigid motion prior

We consider the rigid motion of an animation as probable, if linear and rotational acceleration are small. Acceleration can be estimated by computing the second derivative of the position and rotation w.r.t. time. Instead of using a higher order interpolation taking $k-1$ previous frames into account we decided to use a simple backwards scheme to estimate the acceleration from two previous frames only:

$$\begin{aligned} \ddot{\mathbf{t}} &= \frac{\mathbf{t} - 2\mathbf{t}_{f-1} + \mathbf{t}_{f-2}}{dt^2} \\ \ddot{\boldsymbol{\omega}} &= \frac{\boldsymbol{\omega} - 2\boldsymbol{\omega}_{f-1} + \boldsymbol{\omega}_{f-2}}{dt^2} \end{aligned}$$

The probability of a plausible rigid movement, i.e. with low acceleration, then is given by

$$p(\mathbf{r}, \tilde{\mathbf{r}}) = \frac{1}{(2\pi)^3 \cdot \sqrt{\sigma_t \cdot \sigma_\omega}} \cdot e^{-\frac{\|\ddot{\mathbf{t}}\|^2}{2\sigma_t} - \frac{\|\ddot{\boldsymbol{\omega}}\|^2}{2\sigma_\omega}}$$

which again boils down to a simple energy term E_{rigid} when applying the negative logarithm

$$E_{rigid}(\mathbf{r}, \tilde{\mathbf{r}}) = \lambda_t \cdot \ddot{\mathbf{t}}^T \cdot \ddot{\mathbf{t}} + \lambda_\omega \cdot \ddot{\boldsymbol{\omega}}^T \cdot \ddot{\boldsymbol{\omega}}$$

where λ_t and λ_ω are constants controlling the influence of this energy term and depending on the kernel parameters σ_r and σ_ω.

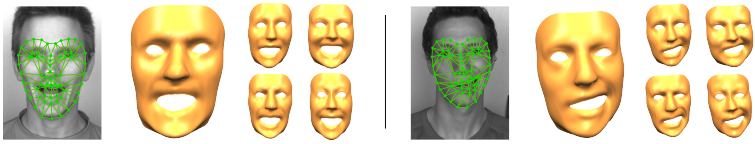


Figure 7.3.: Our tracking approach allows to directly transfer an expression to other faces. In this example we demonstrate the re-targeting of facial expressions by carrying over an tracking result of an intermediate frame to four different facial shapes.

Jacobians of the residual functions. The Jacobians of the residual functions $\ddot{\mathbf{t}}$ and $\ddot{\omega}$ are both of dimension $\mathbb{R}^{3\times(6+l)}$. Since both functions depend only on the rigid parameters $\mathbf{t}$ or ω they can be handled analogously and for simplicity we only present the Jacobean of $\ddot{\mathbf{t}}$ w.r.t to the parameter t_x which evaluates to

$$\frac{\partial \ddot{\mathbf{t}}}{\partial t_x} = \frac{1}{dt^2}$$

7.2.3. Shape deformation prior

In Section 7.1.1 we explained how to split the animations in our database into snippets. Each snippet can be represented by concatenating its sequence of deformation parameters $(\tilde{\mathbf{s}}^T, \mathbf{s}^T)^T$ to a $k \cdot l$ dimensional vector. Similar to Weise et al. [WBLP11] we compute a Gaussian Mixture Model (cf. Section 2.2.2) to estimate the likelihood of a given animation snippet w.r.t. our observed animation data stored in the database. The main difference to the original approach of Weise is that our deformation gradient model is independent of individual faces and needs to be computed only once.

A Gaussian Mixture Model is able to approximate complex data distributions by a set of local Gaussian models. Given weights π_i, center positions of local Gaussian kernels $\boldsymbol{\mu}_i$ and covariance matrices $\boldsymbol{\Sigma}_i$ one can compute the probability of a point $\mathbf{x} \in \mathbb{R}^d$ as a sum of normal distributions

$$p_{GM}(\mathbf{x}) = \sum_i \pi_i \frac{1}{\sqrt{(2\pi)^d |\boldsymbol{\Sigma}_i|}} e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_i)^T \boldsymbol{\Sigma}_i^{-1}(\mathbf{x}-\boldsymbol{\mu}_i)} \tag{7.4}$$

In our setting the goal is to compute a probability value for a given animation snippet. The snippet representation still contains a lot of redundancy and with typical values for $k = 3$ and $l = 15$ is rather high dimensional. Silverman theoretically derived that the number of samples, needed to compute a density distribution function, grows exponentially with the dimensions of the samples [Sil86]. So we decided to learn the Gaussian Mixture Model in a space of reduced dimensionality obtained by a PCA and thereby stabilize the learning procedure.

Training the parameters of the mixture model. Performing a PCA on the snippet coefficients $\mathbf{y} = (\tilde{\mathbf{s}}^T, \mathbf{s}^T)^T$ results in a vector $\bar{\mathbf{y}} \in \mathbb{R}^{k \cdot l}$ representing the average coefficient and a matrix $\mathcal{A} \in \mathbb{R}^{(k \cdot l) \times d}$ storing the main variations modes w.r.t. the average deformation. Keeping 99% of the energy reduces the dimensionality with the above parameters to $d = 20$. We define the sample points $\mathbf{x}$ to be the compact representation of the snippet coefficients $\mathbf{y}$

$$\mathbf{x} = \mathcal{A}^T \cdot (\mathbf{y} - \bar{\mathbf{y}})$$

and employ the EM Algorithm [Bil98] to robustly learn the parameters $\pi_i \in \mathbb{R}$, $\boldsymbol{\mu}_i \in \mathbb{R}^d$ and $\boldsymbol{\Sigma}_i \in \mathbb{R}^{d \times d}$ in this reduced space. Finally we define the last energy term as

$$E_{shape}(\mathbf{s}, \tilde{\mathbf{s}}) = -\ln(p_{GM}(\mathcal{A}^T \cdot (\mathbf{y} - \bar{\mathbf{y}})))$$

Considering this equation, it becomes clear, that it does not directly fit the requirements of a quadratic function, needed to directly apply the Levenberg Marquard algorithm. We therefore introduce a the residual function

$$h = \sqrt{-\ln(p_{GM}(\mathcal{A}^T \cdot (\mathbf{y} - \bar{\mathbf{y}})))} \in \mathbb{R}$$

and write the energy of the shape deformation prior as $E_{shape} = h^T \cdot h$ instead.

Jacobian of the residual function. Similar to separating the snippet coefficients $\mathbf{y} \in \mathbb{R}^{k \cdot l}$ into a part $\tilde{\mathbf{s}}$ (previous frames) and a part $\mathbf{s}$ (actual frame) we similarly divide the matrix $\mathcal{A} = \begin{bmatrix} \mathcal{A}_{\tilde{\mathbf{s}}} \\ \mathcal{A}_{\mathbf{s}} \end{bmatrix} \in \mathbb{R}^{k \cdot l \times d}$ and the vector $\bar{\mathbf{y}} =$

$\begin{bmatrix} \bar{\mathbf{y}}_{\tilde{\mathbf{s}}} \\ \bar{\mathbf{y}}_{\mathbf{s}} \end{bmatrix} \in \mathbb{R}^{k \cdot l}$, where $\mathcal{A}_{\mathbf{s}} \in \mathbb{R}^{l \times d}$ and $\bar{\mathbf{y}}_{\mathbf{s}} \in \mathbb{R}^{l}$. Then we can rewrite the exponent of Equation 7.4 and express this as a quadratic function of the form

$$\begin{aligned} f(\mathbf{y}) &= (\mathbf{x} - \boldsymbol{\mu}_i)^T \boldsymbol{\Sigma}_i^{-1} (\mathbf{x} - \boldsymbol{\mu}_i) \\ &= \mathbf{s}^T \boldsymbol{E}_i \mathbf{s} + 2 \cdot \boldsymbol{F}_i^T \mathbf{s} + b_i \end{aligned}$$

where

$$\begin{aligned} \boldsymbol{E}_i &= \mathcal{A}_{\mathbf{s}} \cdot \boldsymbol{\Sigma}_i^{-1} \cdot \mathcal{A}_{\mathbf{s}}^T \in \mathbb{R}^{l \times l} \\ \boldsymbol{F}_i &= \mathcal{A}_{\mathbf{s}} \cdot \boldsymbol{\Sigma}_i^{-1} \cdot \boldsymbol{\lambda}_i \in \mathbb{R}^{l} \\ b_i &= \boldsymbol{\lambda}_i^T \cdot \boldsymbol{\Sigma}_i^{-1} \cdot \boldsymbol{\lambda}_i \in \mathbb{R} \quad \text{and} \\ \boldsymbol{\lambda}_i &= \mathcal{A}_{\tilde{\mathbf{s}}}^T \cdot \tilde{\mathbf{s}} + \mathcal{A}^T \cdot \bar{\mathbf{y}} - \boldsymbol{\mu}_i \in \mathbb{R}^{d} \end{aligned}$$

are constant matrices and vectors. Since the derivative of the residual function h w.r.t. to the rigid parameters is zero, we restrict the computation of the Jacobian to the deformation parameters $\mathbf{s}$, which is given by

$$\frac{h(\mathbf{s}, \tilde{\mathbf{s}})}{\partial \mathbf{s}} = \frac{1}{2\sqrt{E_{shape}(\mathbf{s}, \tilde{\mathbf{s}})} \cdot p_{GM}(\mathbf{x})} \cdot \sum_i \pi_i \frac{e^{-\frac{1}{2} f(\mathbf{y})}}{\sqrt{(2\pi)^d |\boldsymbol{\Sigma}_i|}} \cdot \left(\mathbf{s}^T \mathbf{E}_i + \mathbf{F_i}^T \right) \in \mathbb{R}^{1 \times l}$$

7.2.4. Optimization

Using the residual functions and their Jacobians defined in the previous sections, we can rewrite Equation 7.1 and formulate the tracking as a least squares optimization problem

$$E(\mathbf{r}, \mathbf{s}) = \lambda_{aam} \cdot \mathbf{g}^T \mathbf{g} + \lambda_t \cdot \ddot{\mathbf{t}}^T \cdot \ddot{\mathbf{t}} + \lambda_\omega \cdot \ddot{\boldsymbol{\omega}}^T \cdot \ddot{\boldsymbol{\omega}} + h^T h$$

We use the Levenberg Marquard Algorithm, which is detailed in Section 2.1, to iteratively compute updates $(\Delta\mathbf{s}, \Delta\mathbf{t})$ for the rigid transformation and the deformation parameters.

7.3. Re-Targeting Facial Animations

Mapping the tracking result $(\mathbf{s}, \mathbf{r})$ of a single frame to another physiognomy encoded as a mesh B is peculiar easy. Therefore we only need to precompute

the matrix $\mathbf{M}_{DS}$ and the vector $\mathbf{b}_{DS}$ depending on the shape of B (cf. Section 7.1.3) and compute the new vertex positions of B as

$$\mathbf{p}(\mathbf{s}, \mathbf{r}) = \mathbf{R}_\alpha \cdot \mathbf{R}_\beta \cdot \mathbf{R}_\gamma \cdot (\mathbf{M}_{DS} \cdot \mathbf{s} + \mathbf{b}_{DS}) + \mathbf{t}$$

In Figure 7.3 we exemplarily re-target a facial expression from an intermediate frame to the faces of four different persons.

7.4. Results

We run all our experiments on an Intel® Core™ i7 CPU with 2.67GHz and used a Stingray F-046B camera for the capture. Since the suggested system has only to fulfill a few constraints (AAM vertex positions) and is minimally parametrized using the deformation space formulation, our Levenberg-Marquard [PTVF07] implementation only needs 16 milliseconds on average to optimize the energy function in each frame. Using the approach of Matthews [MB04] fitting an AAM to one frame takes 20ms. By parallelizing the 3D face tracking in a given frame with the AAM computation of the next frame we obtain a frame rate of 40 fps with one frame latency.

Our experiments indicated that using rather small snippets ($k = 3$) is a good choice to predict the facial movement. This is not surprising when considering the fact that facial movements are quite fast: even at capture rates of 40 frames it takes only 3 to 5 frames to change a facial expression from, e.g., "neutral" to "surprised". For all our experiments we used 45 Gaussians to train the Gaussian Mixture Model.

In Figure 7.5 and the accompanying video we used sequences of a length of up to 10 seconds and show results produced with our system. In each experiment we excluded the facial movements of the tracked person from the database. The left image in each example shows one image of a video sequence overlayed with the detected facial feature points. The middle image shows the result produced by our tracking approach. Since we used our own multiview stereo capture system to acquire the database we are able to compare our result with a ground truth 3D stereo reconstruction obtained from a 4 cameras rig (right image). We use the stereo reconstruction of the neutral face as reference mesh to initialize the tracking procedure. One can see that the resulting deformations are quite

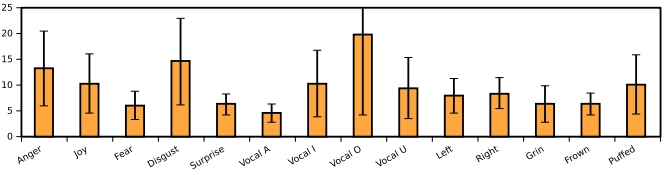


Figure 7.4.: Average distance and standard deviation (in mm) between vertices of the tracked mesh and a ground truth stereo reconstruction.

similar to the ground truth reconstructions. To quantify this a bit more, we took the tracking result of one subject performing different facial expressions and measured in each frame the error as the average distance between vertices of the tracked mesh and their corresponding vertices of the stereo reconstruction. This allows us to compute an average error and its standard deviation for each sequence of approximately 2 seconds length, as shown in Figure 7.4. Except for the expression "Vocal O" we observe a rather small average error of about 10-15 millimeters.

Limitations Due to the resolution of our face template, the presented approach is not intended to generate high quality 3D models from videos and fails to reproduce details like e.g. wrinkles. In this work we focus on the benefit of our deformation model for tracking and re-targeting of facial expressions and only incorporate a simplistic AAM model, which is not trained for large head rotations and does not incorporate visibility checks. Due to this the tracking can become unstable if large head rotations are involved.

Figure 7.5.: Results produced with our system. Each block contains an image of the video sequence (left), the result produced with our system (middle) and the corresponding 3D stereo reconstruction as a ground truth comparison. The results of our system are quite similar to the baseline.

8. Conclusion

In this thesis we studied image based techniques to realistically reconstruct and synthesize dynamic facial expressions. We designed new reconstruction systems for the purpose to obtain a large database of facial expressions performed by a wide variety of different actors. Based on this data, we proposed a new dynamic model for facial movements and studied its applicability for facial performance capture. In what follows we will summarize our contributions in detail and discuss possible future work as an outlook.

Summary

To better solve the initialization process of any 3D stereo reconstruction approach, which involves the triangulation of facial feature points (i.e. extraordinary points around the nose, the eyes and the mouth), we proposed a new method to localize those facial features in 2D image space. Adopting the framework of Kozakaya [KSYY10], we introduced a new way to deduce offset vectors pointing to facial feature, without using any time consuming search algorithm. The key contribution was a compact codebook learned by performing a principal component analysis on a set of HOGs and offset vectors pointing towards the facial features. After the codebook was algorithmically learned, we could use it to obtain the most probable offset vectors from a specific image segment towards facial features. Doing this for many image segments leads to many votes for the location of facial features, from which we could precisely determine the positions of the facial features. By using PCA to compute the compact codebook we can control the tradeoff between storage size and accuracy of the localization process and show that even on low memory consumption, it can compete with state-of-the-art detection methods regarding accuracy and detection time. Our experiments additionally showed that the approach is able to handle partial occlusion of the face very robustly.

We introduced and compared new systems to marker-less reconstruct dynamic facial expressions. Both systems are able to establish inter-subject correspondences as well as temporal correspondence and we showed that this property is important for application like automatic facial modeling, expression blending, attenuation and transfer. The systems use 3D stereo reconstruction to compute 3D surfaces from 2D images. Since this requires accurately calibrated cameras we introduced a new, practical and fully automatic calibration technique, which is based on Zhang's camera calibration method [Zha00]. It robustly detects LED lamps embedded in a planar surface and automatically computes the intrinsic and extrinsic camera parameters for a multi-camera rig.

The quality of most 3D stereo approaches usually strongly depends on the initially estimated depth of the surface to be reconstructed w.r.t. to the camera. To stabilize stereo reconstruction for human skin, which often appears quite homogeneous in low resolution images, we introduced initialization methods which are based on triangulated facial feature points. We showed that a morphable model for a static, neutral face is able to produce a more accurate initialization than simple deformation techniques. Nevertheless, it comes with the expense of a more complicated setup, which involves the scanning of several different faces, followed by a PCA-based learning approach. Seen from a practical point of view, we come to the conclusion, that in combination with a simple point based stereo reconstruction technique, the initialization based on an as-rigid-as-possible deformation step is superior to the more accurate morphable model initialization, combined with a sufel based reconstruction approach. Although the surfel based reconstruction is mathematicaly more elegant and potentially produces much more accurate reconstructions, which we showed in our experiments, in practice it has more problems with bad surface initializations than the point based approach. This is because the surfel based technique can only find the optimal correspondences, if the correlation window partially overlaps with the image region, which contains the optimal correspondence. In contrast to this, point based reconstruction is searching for the best correspondences by traversing different image regions.

Instead of using optical flow methods [HS80] to establish temporal correspondences, we proposed a new mesh based tracking approach. It has the advantage that it can handle discontinuities at the eyes and the mouth very naturally: By

changing the topology of the image mesh we insert holes at mouth and the eyes, such that we simply remove the smoothness constraint in those regions. More concretely this means that upper and lower parts of the eyes and lips can move independently from each other. Our new mesh based tracking approach is based on the inverse compositional image alignment [BM04]. By changing the role of reference and the comparison mesh, we significantly improved the running time, while still obtaining accurate tracking results.

Our final system design uses this tracking approach in combination with point based stereo reconstruction to compute a scene flow that drives the deformation of a generic face template. The system maintains temporal correspondences and we showed that this allows us to automatically enhance our model by placing eyes, lids and lashes. It additionally allows us to use deformation transfer for triangle meshes [SP04] to transfer expressions from one subject to another, to attenuate expressions or to blend expressions of different subjects.

We used the proposed system design and reconstructed facial expressions from over 60 different persons. Besides the 6 basic emotions, each persons performed additional facial expressions, exploiting the numerous degrees of freedoms of the human face. Finally, the resulting database of facial expressions was used to introduce a new dynamic morphable face model. By analyzing per triangle deformation gradients, which encode the difference between a reference (neutral) face to a face showing a facial expression, we decouple the dynamic movements of a facial expression from the individual facial shapes. This allows us to compute a general motion model for facial expressions from the given database. We show that this model can be used to extrapolate the 3D motion from the tracking of 2D facial features seen in a single video stream.

Outlook

In future work it would be interesting to further investigate the suggested deformation space and the derived motion model. Since it is independent from individual shapes, it could be used to create better indicator functions to recognize facial expressions [Bet12]. It would also be interesting to have an even more complete database containing all possible visems [Fis68] a human face is able to do. From this database we then could learn more general models for

each of the visems. In combination with speech recognition or speech synthesis, these models could be used to instantaneously transfer facial actions to any facial shape. By this we could drive any avatar in a computer game, a movie or in a virtual chatroom.

Our current tracking system focus on the facial region only. But just creating a realistic facial animation is not a guarantee to overcome the uncanny valley. For this the whole appearance of the avatar has to be right. This also includes realistic movements of the head, the eyes and also the neck [LT06, KBB*08]. Therefore it would be interesting to extend the reconstruction system to reliably reconstruct the neck part and to better deal with large head rotations, which are currently assumed to be rather small. To handle large head rotations and to capture the neck, the camera rig has to be significantly extended. Additionally we need automatic reinitialization methods which are able to identify facial features to compensate for drift and occlusions during video tracking (cf. Section 6.1).

Besides the installation of additional cameras, the cameras itself should be improved. To additionally capture fine details of the facial area and the neck, it is necessary to increase the resolution of the cameras (currently we only capture images with a resolution of 780×580). We focused in this thesis on pure image based methods, which are generally able to record data at higher frame rates than RGB-d cameras (in our setting we captured at 40 fps). But in order to realistically capture faces including important micro expressions and saccades [RBQ14, LBB02] one should prefer a setup capable to capture faces at even higher frame rates.

With additional movements of the head, eyes and the neck at high temporal and spacial resolutions, the deformation space would be sampled more accurate and more densely by the captured database. This in turn would not only stabilize and improve our suggested face tracking approach from single images but would also produce much more realistic virtual avatars in general.

Bibliography

[ABCO*01] Alexa M., Behr J., Cohen-Or D., Fleishman S., Levin D., Silva C. T.: Point set surfaces. In *Proceedings of the Conference on Visualization '01*, Washington, DC, USA, 2001, VIS '01, IEEE Computer Society, pp. 21–28.

[ACK13] Attene M., Campen M., Kobbelt L.: Polygon mesh repairing: An application perspective. *ACM Comput. Surv. 45*, 2, 2013, 15:1–15:33.

[AMHH08] Akenine-Möller T., Haines E., Hoffman N.: *Real-Time Rendering 3rd Edition*. A. K. Peters, Ltd., Natick, MA, USA, 2008.

[AMN*94] Arya S., Mount D. M., Netanyahu N. S., Silverman R., Wu A. Y.: An Optimal Algorithm for Approximate Nearest Neighbor Searching in Fixed Dimensions. In *ACM-SIAM Symposium on Discrete Algorithms*, 1994, pp. 573–582.

[ARL*09] Alexander O., Rogers M., Lambeth W., Chiang M., Debevec P.: The Digital Emily project: photoreal facial modeling and animation. In *ACM SIGGRAPH Courses*, NY, USA, 2009, ACM, pp. 12:1–12:15.

[BBA*07] Bickel B., Botsch M., Angst R., Matusik W., Otaduy M., Pfister H., Gross M.: Multi-scale capture of facial geometry and motion. *TOG 26*, 3, 2007, 33.

[BBH08] Bradley D., Boubekeur T., Heidrich W.: Accurate multiview reconstruction using robust binocular stereo and surface meshing. In *In PROC. OF CVPR*, 2008.

[BCWG09] Ben-Chen M., Weber O., Gotsman C.: Variational Harmonic Maps for Space Deformation. *TOG*, 2009.

[Bet12] Bettadapura V.: Face expression recognition and analysis: The state of the art. *CoRR abs/1203.6722*, 2012.

[BHPS10] Bradley D., Heidrich W., Popa T., Sheffer A.: High resolution passive facial performance capture. *TOG 29*, 4, 2010.

[Bil98] Bilmes J.: *A Gentle Tutorial of the EM Algorithm and its Application to Parameter Estimation for Gaussian Mixture and Hidden Markov Models.* Tech. rep., 1998.

[Bis06] Bishop C. M.: *Pattern Recognition and Machine Learning (Information Science and Statistics).* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[BKP*10] Botsch M., Kobbelt L., Pauly M., Alliez P., Levy B.: *Polygon Mesh Processing.* Ak Peters Series. Taylor & Francis, 2010.

[BM04] Baker S., Matthews I.: Lucas-Kanade 20 Years On: A Unifying Framework. *Int. Journal of Computer Vision 56*, 3, 2004, 221–255.

[BMBM10] Bourdev L., Maji S., Brox T., Malik J.: Detecting People Using Mutually Consistent Poselet Activations. In *ECCV*, 2010, vol. 6316, Springer, pp. 168–181.

[Bou08] Bouguet J. Y.: Camera calibration toolbox for Matlab, 2008.

[BPL*05] Borshukov G., Piponi D., Larsen O., Lewis J. P., Tempelaar-Lietz C.: Universal capture - image-based facial animation for "The Matrix Reloaded". In *ACM SIGGRAPH Courses*, 2005, ACM.

[Bro66] Brown D. C.: Decentering distortion of lenses. *Photometric Engineering 32*, 3, 1966, 444–462.

[BSPG06] Botsch M., Sumner R., Pauly M., Gross M.: Deformation transfer for detail-preserving surface editing. In *Vision, Modeling and Visualization*, 2006, pp. 357–364.

[BV99] Blanz V., Vetter T.: A Morphable Model for the Synthesis of 3D Faces. In *Siggraph 1999*, 1999, Rockwood A., (Ed.), pp. 187–194.

[BV03] Blanz V., Vetter T.: Face Recognition Based on Fitting a 3D Morphable Model. *PAMI 25*, 9, 2003, 1063–1074.

[CB02] Chuang E., Bregler C.: *Performance driven facial animation using blendshape interpolation.* Tech. rep., Stanford University, 2002.

[CBK*06] Curio C., Breidt M., Kleiner Q. C. V. M., Giese M. A., Blthoff H. H.: Semantic 3D motion retargeting for facial animation. In *Applied perception in graphics and visualization*, USA, 2006, pp. 77–84.

[CBMK*06] Curio C., Breidt M., M. Kleiner Q. C. V., Giese M. A., Bülthoff H. H.: Semantic 3d motion retargeting for facial animation. In *Applied perception in graphics and visualization*, USA, 2006, pp. 77–84.

[CDHR08] Chen Y., Davis T. A., Hager W. W., Rajamanickam S.: Algorithm 887: Cholmod, supernodal sparse cholesky factorization and update/downdate. *ACM Trans. Math. Softw. 35*, 3, 2008, 22:1–22:14.

[CET98] Cootes T. F., Edwards G. J., Taylor C. J.: Active Appearance Models. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1998, Springer, pp. 484–498.

[CET01] Cootes T. F., Edwards G. J., Taylor C. J.: Active Appearance Models. *IEEE Transactions on Pattern Analysis and Machine Intelligence 23*, 2001, 681–685.

[Che87] Chew L. P.: Constrained delaunay triangulations. In *Proceedings of the Third Annual Symposium on Computational Geometry*, New York, NY, USA, 1987, SCG '87, ACM, pp. 215–222.

[CHZ14] Cao C., Hou Q., Zhou K.: Displaced dynamic expression regression for real-time facial tracking and animation. *ACM Trans. Graph. 33*, 4, July 2014, 43:1–43:10.

[CO10] Choi H.-C., Oh S.-Y.: Modified efficient second order minimization for AAM fitting. *IET Electronics Letters 46*, 13, 2010, 913–914.

[CSK15] Choi H.-C., Sibbing D., Kobbelt L.: Non-parametric facial feature localization using segment-based eigen features. *Computational Intelligence and Neuroscience*, 2015.

[CTCG95] Cootes T. F., Taylor C. J., Cooper D. H., Graham J.: Active shape modelstheir training and application. *CVIU 61*, 1, 1995, 38–59.

[CWLZ13] Cao C., Weng Y., Lin S., Zhou K.: 3d shape regression for real-time facial animation. *ACM Trans. Graph. 32*, 4, 2013, 41:1–41:10.

[CWZ*14] Cao C., Weng Y., Zhou S., Tong Y., Zhou K.: Facewarehouse: A 3d facial expression database for visual computing. *IEEE Transactions on Visualization and Computer Graphics 20*, 3, Mar. 2014, 413–425.

[CXH03] Chai J.-x., Xiao J., Hodgins J.: Vision-based control of 3D facial animation. In *Symposium on Computer animation*, 2003, SCA '03, Eurographics Association, pp. 193–206.

[CZLZ04] Chen L., Zhang L., Li M., Zhang H.: A Novel Facial Feature Localization Method Using Probabilistic-like Output. In *Proc. of Asian Conf. on Computer Vision*, 2004.

[DC76] Do-Carmo M. P.: *Differential Geometry of Curves and Surfaces*, first ed. Prentice Hall, 1976.

[DM96] DeCarlo D., Metaxas D.: The Integration of Optical Flow and Deformable Models with Applications to Human Face Shape and Motion Estimation. In *CVPR*, Washington, DC, USA, 1996, CVPR '96, IEEE Computer Society.

[DN] Deng Z., Neumann U.: *Data-Driven 3D Facial Animation*, 1 ed.

[DPT*08] Dellepiane M., Pietroni N., Tsingos N., Asselot M., Scopigno R.: Reconstructing head models from photographs for individualized 3D-audio processing. *Proc. Pacific Graphics 27*, 7, 2008, 1719–1727.

[DT05] Dalal N., Triggs B.: Histograms of Oriented Gradients for Human Detection. In *CVPR*, 2005, pp. 886–893.

[EF78] Ekman P., Friesen W. V.: Manual for the Facial Action Coding System. *Consulting Psychology Press*, 1978.

[ES04] Esteban C. H., Schmitt F.: Silhouette and stereo fusion for 3D object modeling. *CVIU 96*, 3, 2004, 367–392.

[ESN06] Engels C., Stewénius H., Nistér D.: Bundle adjustment rules. In *In Photogrammetric Computer Vision*, 2006.

[FCOS05] Fleishman S., Cohen-Or D., Silva C. T.: Robust moving least-squares fitting with sharp features. In *ACM SIGGRAPH 2005 Papers*, New York, NY, USA, 2005, SIGGRAPH '05, ACM, pp. 544–552.

[Fis68] Fisher C. G.: Confusions among visually perceived consonants. *Journal of Speech, Language, and Hearing Research 11*, 4, 1968, 796–804.

[FJ05] Frigo M., Johnson S. G.: The design and implementation of FFTW3. *Proceedings of the IEEE 93*, 2, 2005, 216–231. Special issue on "Program Generation, Optimization, and Platform Adaptation".

[FP09] Furukawa Y., Ponce J.: Dense 3D Motion Capture for Human Faces. In *Proc. of Computer Vision and Pattern Recognition (CVPR)*, 2009.

[Goo91] Goodall C.: Procrustes Methods in the Statistical Analysis of Shape. *Journal of the Royal Statistical Society. Series B (Methodological) 53*, 2, 1991, pp. 285–339.

[Han05] Han J.: *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.

[HDH*10] Hornung A., Dekkers E., Habbecke M., Gross M., Kobbelt L.: *Character Reconstruction and Animation from Uncalibrated Video*. Technical report, RWTH Aachen University, 2010.

[HK06] Habbecke M., Kobbelt L.: Iterative Multi-View Plane Fitting. In *Proc. of Vision, Modeling, and Visualization (VMV)*, 2006, pp. 73–80.

[HK09] Habbecke M., Kobbelt L.: An Intuitive Interface for Interactive High Quality Image-Based Modeling. In *Computer Graphics Forum*, 2009, vol. 28, Wiley Online Library, pp. 1765–1772.

[HLL11] Heo Y. S., Lee K. M., Lee S. U.: Robust stereo matching using adaptive normalized cross-correlation. *IEEE Trans. on Pattern Analysis and Machine Intelligence 33*, 4, April 2011, 807–822.

[Hor87] Horn B. K. P.: Closed-form solution of absolute orientation using unit quaternions. *Journal of the Optical Society of America. A 4*, 1987, 629–642.

[HS80] Horn B. K., Schunck B. G.: *Determining Optical Flow*. Tech. rep., Cambridge, MA, USA, 1980.

[HVB*07] Hernndez C., Vogiatzis G., Brostow G. J., Stenger B., Cipolla R.: Non-Rigid Photometric Stereo with Colored Lights. In *Intl. Conf. on Comp. Vision*, 2007.

[HZ03] Hartley R., Zisserman A.: *Multiple View Geometry in Computer Vision*, second ed. Cambridge University Press, ISBN: 0521540518, 2003.

[JJC*14] Joyal C., Jacob L., Cigna M.-H., Guay J.-P., RENAUD P.: Virtual facial expressions of emotions: An initial concomitant and construct validity study. *Frontiers in Human Neuroscience 8*, 787, 2014.

[JMD*07] Joshi P., Meyer M., DeRose T., Green B., Sanocki T.: Harmonic coordinates for character articulation. *TOG 26*, 3, 2007, 71.

[JW76] Jenkins F., White H.: *Fundamentals of Optics*. International student edition. McGraw-Hill, 1976.

[Kat98] Katz L.: *Magill's Medical Guide Revised Edition*. Salem Press, 1998.

[KBB*08] Kohlbecher S., Bardinst S., Bartl K., Schneider E., Poitschke T., Ablassmeier M.: Calibration-free eye tracking by reconstruction of the pupil ellipse in 3d space. In *Proceedings of the 2008 Symposium on Eye Tracking Research & Applications*, New York, NY, USA, 2008, ETRA '08, ACM, pp. 135–138.

[KBM*05] Koterba S. C., Baker S., Matthews I., Hu C., Xiao J., Cohn J., Kanade T.: Multi-View AAM Fitting and Camera Calibration. In *Proc. ICCV*, 2005, vol. 1, pp. 511–518.

[KHYS02] Khler K., Haber J., Yamauchi H., Seidel H.-P.: Head shop: generating animated head models with anatomical structure. In *Proc. SCA*, NY, USA, 2002.

[KSTN08] Kozakaya T., Shibata T., Takeguchi T., Nishiura M.: Fully automatic feature localization for medical images using a global vector concentration approach. In *CVPR*, jun. 2008, pp. 1–6.

[KSYY10] Kozakaya T., Shibata T., Yuasa M., Yamaguchi O.: Facial feature localization using weighted vector concentration approach. *IVC 28*, 5, 2010, 772–780.

[LBB02] Lee S. P., Badler J. B., Badler N. I.: Eyes alive. *ACM Trans. Graph. 21*, 3, 2002, 637–644.

[LLCO08] Lipman Y., Levin D., Cohen-Or D.: Green Coordinates. *TOG 27*, 3, 2008, 1–10.

[LLS04] Leibe B., Leonardis A., Schiele B.: Combined Object Categorization and Segmentation With An Implicit Shape Model. In *ECCV workshop on stat. learning in computer vision*, 2004, pp. 17–32.

[LO05] Lin I.-C., Ouhyoung M.: Mirror MoCap: Automatic and efficient capture of dense 3D facial motion parameters from video. *The Visual Computer 21*, 6, 2005, 355–372.

[LRF93] Li H., Roivainen P., Forcheimer R.: 3-D Motion Estimation in Model-Based Facial Image Coding. *IEEE Transactions on Pattern Analysis and Machine Intelligence 15*, 1993, 545–555.

[LS08] Li R., Sclaroff S.: Multi-scale 3D scene flow from binocular stereo sequences. *CVIU 110*, 1, 2008, 75–90.

[LT06] Lee S.-H., Terzopoulos D.: Heads up!: Biomechanical modeling and neuromuscular control of the neck. In *ACM SIGGRAPH 2006 Papers*, New York, NY, USA, 2006, SIGGRAPH '06, ACM, pp. 1188–1198.

[LTW93] Lee Y., Terzopoulos D., Waters K.: Constructing Physics-Based Facial Models of Individuals. In *In Proc. Graphics Interface*, 1993, pp. 1–8.

[LWC*09] Lucey S., Wang Y., Cox M., Sridharan S., Cohn J. F.: Efficient constrained local model fitting for non-rigid face alignment. *IVC 27*, 12, 2009, 1804–1813.

[LYYB13] Li H., Yu J., Ye Y., Bregler C.: Realtime facial animation with on-the-fly correctives. *ACM Trans. Graph. 32*, 4, July 2013, 42:1–42:10.

[MB04] Matthews I., Baker S.: Active Appearance Models Revisited. *Int. J. Comput. Vision 60*, 2, 2004, 135–164.

[MDR*07] Moser E., Derntl B., Robinson S., Fink B., Gur R., Grammer K.: Amygdala activation at 3t in response to human and avatar facial expressions of emotions. *Journal of Neuroscience Methods 161*, 1, 2007, 126–133.

[MHP*07] Ma W.-C., Hawkins T., Peers P., Chabert C.-F., Weiss M., Debevec P. E.: Rapid Acquisition of Specular and Diffuse Normal Maps from Polarized Spherical Gradient Illumination. In *Rendering Techniques*, 2007, Eurographics Association, pp. 183–194.

[MJC*08] Ma W.-C., Jones A., Chiang J.-Y., Hawkins T., Frederiksen S., Peers P., Vukovic M., Ouhyoung M., Debevec P.: Facial performance synthesis using deformation-driven polynomial displacement maps. *TOG 27*, 5, 2008, 1–10.

[MN99] Magnus J., Neudecker H.: *Matrix Differential Calculus with Applications in Statistics and Econometrics*. Wiley Series in

Probability and Statistics: Texts and References Section. Wiley, 1999.

[MN08] Milborrow S., Nicolls F.: Locating Facial Features with an Extended Active Shape Model. In *Proc. of European Conf. on Computer Vision*, Berlin, Heidelberg, 2008, Springer-Verlag, pp. 504–513.

[Mor70] Mori M.: Bukimi no tani [The uncanny valley]. *Energy 7*, 4, 1970, 33–35.

[NJ04] Na K., Jung M.: Hierarchical Retargetting of Fine Facial Motions. *Comput. Graph. Forum 23*, 3, 2004, 687–695.

[NN01] Noh J.-y., Neumann U.: Expression cloning. In *ACM Siggraph*, 2001, ACM, pp. 277–288.

[NW06] Nocedal J., Wright S.: *Numerical Optimization*, 2nd ed. Springer, 2006.

[OOB03] Odisio M., Odisio M., Bailly G.: Shape and Appearance Models of Talking Faces for Model-Based Tracking. In *Proc. AVSP*, 2003, pp. 105–110.

[PL06] Pighin F., Lewis J.: Performance-driven facial animation. In *ACM SIGGRAPH 2006 Courses*, 2006, ACM.

[PMRR00] Phillips P. J., Moon H., Rizvi S. A., Rauss P. J.: The FERET Evaluation Methodology for Face-Recognition Algorithms. *PAMI 22*, 10, 2000, 1090–1104.

[PSS99] Pighin F. H., Szeliski R., Salesin D.: Resynthesizing Facial Animation through 3D Model-based Tracking. In *ICCV*, 1999, pp. 143–150.

[PTVF07] Press W. H., Teukolsky S. A., Vetterling W. T., Flannery B. P.: *Numerical Recipes 3rd Edition: The Art of Scientific Computing*, 3 ed. Cambridge University Press, NY, USA, 2007.

[QSS07] Quarteroni A., Sacco R., Saleri F.: *Numerical Mathematics*, 2nd ed. Springer, 2007.

[RBQ14] Rossana B. Queiroz Soraia R. Musse N. I. B.: Investigating macroexpressions and microexpressions in computer graphics animated faces, 2014.

[RL87] Rousseeuw P. J., Leroy A. M.: *Robust regression and outlier detection*. John Wiley & Sons, Inc., USA, 1987.

[RMK*14] Riedl R., Mohr P. N. C., Kenning P. H., Davis F. D., Heekeren H. R.: Trusting humans and avatars : A brain imaging study based on evolution theory. *Journal of Management Information Systems 30*, 2014, 83–114.

[SA07] Sorkine O., Alexa M.: As-rigid-as-possible surface modeling. In *Proceedings of the Fifth Eurographics Symposium on Geometry Processing*, Aire-la-Ville, Switzerland, Switzerland, 2007, SGP '07, Eurographics Association, pp. 109–116.

[SCD*06] Seitz S., Curless B., Diebel J., Scharstein D., Szeliski R.: A Comparison and Evaluation of Multi-View Stereo Reconstruction Algorithms. In *Proc. of CVPR*, 2006, vol. 1, pp. 519–526.

[SCOL*04] Sorkine O., Cohen-Or D., Lipman Y., Alexa M., Rssl C., Seidel H.-P.: Laplacian surface editing. In *Proc. SGP*, NY, USA, 2004, pp. 175–184.

[SHK09] Sibbing D., Habbecke M., Kobbelt L.: Markerless reconstruction of dynamic facial expressions. In *2009 IEEE 12th International Conference on Computer Vision Workshops (ICCV Workshop 3DIM)*, 2009, pp. 1778–1785.

[SHK11] Sibbing D., Habbecke M., Kobbelt L.: Markerless reconstruction and synthesis of dynamic facial expressions. *Comput. Vis. Image Underst. 115*, 5, May 2011, 668–680.

[SIA*13] Sakai S., Ito K., Aoki T., Masuda T., Unten H.: An efficient image matching method for multi-view stereo. In *Proceedings of the 11th Asian Conference on Computer Vision - Volume Part IV*, Berlin, Heidelberg, 2013, ACCV'12, Springer-Verlag, pp. 283–296.

[Sil86] Silverman B.: *Density Estimation for Statistics and Data Analysis*. Monographs on Statistics and Applied Probability. Taylor & Francis, 1986.

[SK] Sibbing D., Kobbelt L.: Building a large database of facial movements for deformation model based 3d face tracking. *Computer Graphics Forum*, to appear.

[SK10] Sanders J., Kandrot E.: *CUDA by Example: An Introduction to General-Purpose GPU Programming*, 1st ed. Addison-Wesley Professional, 2010.

[SK15] Sibbing D., , Kobbelt L.: Data driven 3d face tracking based on a facial deformation model. In *International Symposium on Vision, Modeling and Visualization*, 2015.

[SNF05] Sifakis E., Neverov I., Fedkiw R.: Automatic determination of facial muscle activations from sparse motion capture marker data. *TOG 24*, 3, 2005, 417–425.

[SP04] Sumner R. W., Popovi J.: Deformation transfer for triangle meshes. In *TOG*, 2004, ACM, pp. 399–405.

[Sze10] Szeliski R.: *Computer Vision: Algorithms and Applications*, 1st ed. Springer-Verlag New York, Inc., New York, NY, USA, 2010.

[VBPP05] Vlasic D., Brand M., Pfister H., Popovi J.: Face transfer with multilinear models. *TOG 24*, 3, 2005, 426–433.

[VBR*99] Vedula S., Baker S., Rander P., Collins R., Kanade T.: Three-Dimensional Scene Flow. In *Proc. ICCV*, September 1999, vol. 2, pp. 722–729.

[Vic01] Vicci L.: *Quaternions and Rotations in 3-Space: The Algebra and Its Geometric Interpretation.* Tech. rep., Chapel Hill, NC, USA, 2001.

[VJ01] Viola P., Jones M.: Rapid Object Detection Using a Boosted Cascade of Simple Features. In *CVPR*, 2001, pp. 511–518.

[Wat87] Waters K.: A muscle model for animation three-dimensional facial expression. In *Computer graphics and interactive techniques*, USA, 1987, pp. 17–24.

[WBBP06] Weickert J., Bruhn A., Brox T., Papenberg N.: A survey on variational optic flow methods for small displacements. In *Mathematical Models for Registration and Applications to Medical Imaging*, Scherzer O., (Ed.), vol. 10 of *Mathematics in industry*. Springer Berlin Heidelberg, 2006, pp. 103–136.

[WBLP11] Weise T., Bouaziz S., Li H., Pauly M.: Realtime Performance-based Facial Animation. *ACM Transactions on Graphics (SIGGRAPH 2011)*, 2011.

[WBV*11] Wedel A., Brox T., Vaudrey T., Rabe C., Franke U., Cremers D.: Stereoscopic scene flow computation for 3d motion understanding. *Int. J. Comput. Vision 95*, 1, 2011, 29–51.

[WFKvdM97] Wiskott L., Fellous J.-M., Krger N., von der Malsburg C.: Face recognition by elastic bunch graph matching. In *CAIP*, 1997, vol. 1296, Springer, pp. 456–463.

[WGP*09] Wieser M. J., Gerdes A. B. M., Pauli P., Weyers P., Breuer F., Mühlberger A.: Almost human: Neural activation to avatar and human emotional facial expressions. *Journal of Cognitive Neuroscience 21*, 2009, 92.

[WL90] Williams, Lance: Performance-driven facial animation. In *Computer graphics and interactive techniques*, USA, 1990, pp. 235–242.

[WLG07] Weise T., Leibe B., Gool L. V.: Fast 3D Scanning with Automatic Motion Compensation. In *Proc. CVPR*, June 2007.

[WLVP09] Weise T., Li H., Van Gool L., Pauly M.: Face/Off: live facial puppetry. In *Symposium on Computer Animation*, 2009, ACM, pp. 7–16.

[WMKG07] Wardetzky M., Mathur S., Klberer F., Grinspun E.: Discrete laplace operators: no free lunch. In *Proc. Symposium on Geometry Processing (SGP)*, 2007, pp. 33–37.

[XBMK04] Xiao J., Baker S., Matthews I., Kanade T.: Real-Time Combined 2D+3D Active Appearance Models. In *CVPR*, June 2004, vol. 2, pp. 535–542.

[XCLT14] Xu F., Chai J., Liu Y., Tong X.: Controllable high-fidelity facial performance transfer. *ACM Trans. Graph. 33*, 4, July 2014, 42:1–42:11.

[Zha00] Zhang Z.: A Flexible New Technique for Camera Calibration. *IEEE Trans. Pattern Anal. Mach. Intell. 22*, 11, 2000, 1330–1334.

[Zha12] Zhang Z.: Microsoft kinect sensor and its effect. *IEEE MultiMedia 19*, 2, Apr. 2012, 4–10.

[ZSCS04] Zhang L., Snavely N., Curless B., Seitz S. M.: Spacetime Faces: High-Resolution Capture for Modeling and Animation. In *ACM Annual Conference on Computer Graphics*, 2004, pp. 548–558.

Curriculum Vitae

Dominik Sibbing

E-Mail	Dominik.Sibbing@rwth-aachen.de
Date of Birth	18.05.1980
Place of Birth	Gronau, Germany
Citizenship	German

Academic Education

Apr 2006 - Dec 2016	Doctoral Student at RWTH Aachen University, Computer Graphics Group. Degree: Dr. rer. nat. Supervisor: Prof. Dr. Leif Kobbelt.
Oct 2002 - Mar 2006	Computer Science Studies at RWTH Aachen University. Degree: Dipl. Inform. Supervisor: Prof. Dr. Leif Kobbelt.

Publications

Dominik Sibbing and Leif Kobbelt: Building a Large Database of Facial Movements for Deformation Model based 3D Face Tracking, Computer Graphics Forum (to appear)

Dominik Sibbing and Leif Kobbelt: Data Driven 3D Face Tracking Based on a Facial Deformation Model, Proceedings of Vision, Modeling and Visualization, 2015

Hyun-Chul Choi, Dominik Sibbing, and Leif Kobbelt: Non-Parametric Facial Feature Localization using Segment-based Eigen Features, Computational Intelligence and Neuroscience, 2015

Dominik Sibbing, Henrik Zimmer, Robin Tomcin, Leif Kobbelt: Interactive Volume-Based Visualization and Exploration for Diffusion Fiber Tracking, Bildverarbeitung für die Medizin, 2014

Robin Tomcin, Dominik Sibbing, Leif Kobbelt: Efficient Enforcement of Hard Articulation Constraints in the Presence of Closed Loops and Contacts, Proceedings of Eurographics, 2014

Lars Krecklau, Dominik Sibbing, Torsten Sattler, Ming Li, Martin Habbecke, Leif Kobbelt: Rekonstruktion urbaner Umgebungen und ihre Anwendungen Exploring Virtuality, Springer Fachmedien Wiesbaden, pp. 199-213, 2014

Dominik Sibbing, Torsten Sattler, Bastian Leibe, Leif Kobbelt: SIFT-Realistic Rendering, Proceedings of Three-dimensional Vision(3DV), 2013

Dominik Sibbing, Hans-Christian Ebke, Kai Ingo Esser, Leif Kobbelt: Topology aware Quad Dominant Meshing for Vascular Structures Lecture Notes in Computer Science, Proceedings of MeshMed, 2012

Dominik Sibbing, Martin Habbecke, Leif Kobbelt: Markerless Reconstruction and Synthesis of Dynamic Facial Expressions, Computer Vision and Image Understanding, Volume 115, Issue 5, Special issue on 3D Imaging and Modelling, May 2011

Dominik Sibbing, Leif Kobbelt: High-Speed Circles Algorithms Unplugged, Springer Berlin Heidelberg, pp. 285-293, 2011

Dominik Sibbing, Darko Pavic, Leif Kobbelt: Image Synthesis for Branching Structures, Computer Graphics Forum, Special Issue of Pacific Graphics, 2010

Dominik Sibbing, Martin Habbecke, Leif Kobbelt: Markerless Reconstruction of Dynamic Facial Expressions, IEEE International Conference on Computer Vision Workshop 3DIM, 2009

Dominik Sibbing, Leif Kobbelt: Kreise zeichnen mit Turbo, Taschenbuch der Algorithmen, Springer Berlin Heidelberg, pp. 303-312, 2008

Dominik Sibbing, Leif Kobbelt: Fast Interactive Region of Interest Selection for Volume Visualization, Bildverarbeitung für die Medizin, 2007

Statement of Originality

Many of the ideas, approaches and the proposed system designs were influenced by many fruitful discussions with members of the Computer Graphics Group led by Professor Dr. Leif Kobbelt. In what follows I will detail my contributions to articles that are relevant for this thesis and which previously have been published.

[SHK09] As the main developer of this project I implemented nearly all modules of the presented system and I was responsible for evaluating the results. I was the main author writing the paper.

[SHK11] As the main developer of this project I implemented nearly all modules of the presented system. I was responsible for implementing the presented applications and for evaluating the results. I was the main author writing the paper.

[CSK15] I equally contributed to the development of the idea and the writing of the article.

[SK15] As the exclusive developer of this project I implemented the presented single-view tracking system and did the qualitative and quantitative analysis of the approach. I was the main author writing the paper.

[SK] This paper extends the work presented in [SK15]. As the exclusive developer of this project I implemented the presented multi-view tracking system and did the qualitative and quantitative analysis of this approach. I was the main author writing the paper.