Aalborg University
Copenhagen Institute of Technology
Department of Media Technology and Engineering Science

Master Thesis

Facial Motion Capture with Sparse 2D-to-3D Active Appearance Models

by

Esben Plenge

Supervisor: Dr. Ing. Daniel Grest

COPENHAGEN 2008

Preface

This thesis was prepared at the *Computer Vision and Machine Intelligence Lab* under the Department of Media Technology and Engineering Science of Aalborg University Copenhagen as a partial fulfillment of the requirements for aacquiring the degree Master of Science in Enginnering, M.Sc.Eng.

The thesis deals with aspects of 3D facial motion capture. It introduces a means of acquiring 3D face shape data for statistical model training, a method capable of synthesizing 3D facial geometry from images has been developed, together with an animation scheme visualizing the captured facial data.

It is assumed that the reader has a basic knowledge in the area of statistics, image analysis and computer graphics.

Copenhagen, October 2008

Esben Plenge

Abstract

This thesis presents a framework for markerless 3D facial motion capture. In particular it considers, if emotions can be conveyed by 3D facial motion capture from sparse geometry. A method dubbed *2D-to-3D Active Appearance Models* has been developed for retrieving sparse 3D geometry from 2D images.

The established motion capture framework is modular. In its current implementation it includes a tool, developed in Matlab, for acquiring 3D facial geometry data for training a statistical model, a method for synthesizing 3D facial geometry from 2D images has been written in C++, and a 3D animation front end, that visualizes the captured face data, also written in C++. The *AAM-API*, an open source C++ implementation of Active Appearance Models, has been used in this work. A thorough description of the theory behind the framework is given in the thesis, and its implementation is discussed in detail.

The emotion conveyance capability of the framework implementation is tested, and it is shown, that the emotions embedded in the captured and animated facial expressions can be interpreted correctly.

Keywords: Facial Motion Capture, Stereopis, Facial Animation, 3D Geomety Synthesis, Active Appearance Models, Human-Computer Interfaces.

Acknowledgements

I would like to thank the following people for their support and assistance during my preparation of the work presented in this thesis:

First and foremost, thanks to my supervisor Dr. Ing. Daniel Grest, for his support throughout this thesis. His pointing out directions of possible solutions to problems at hand, has been valuable and educational.

I would also like to thank my co-students, Christian Gleerup and Uni Dahl, for discussions on subject matter concerning the thesis work. My conversations with them has helped clarify problems and solutions during the work process, and I have enjoyed their good spirit.

Thanks also, to Sanmohan, phd. student and teacher at the Department of Media Technology and Engineering Science under Aalborg University Copenhagen. He has always taken the time to help with advice in case I approached him with problems concerning statistics, Matlab, LaTex and more.

Finally, I would like to thank my professor Volker Krüger, for establishing the Computer Vision and Graphics education in Copenhagen. As this thesis concludes my two years at the master study, I can say sincerely, that I have come a long way since the beginning of the study.

Contents

Chapter 1

Introduction

"My approach to computer vision is best characterized as inverse computer graphics. In computer graphics, the world is represented in sufficient detail so that the image forming process can be numerically simulated to generate synthetic television images; in the inverse, perceived television pictures are analyzed to compute detailed geometric models."

- Bruce G. Baumgart [1].

A fundamental problem in computer graphics is the construction of 3D geometric models consistent with the real world. In computer vision the same problem can be formulated as that of *constructing consistent 3D geometric models from two dimensional images*.

This thesis considers how to synthesize the 3D geometry of human facial expressions from two dimensional images.

The problem of capturing facial expressions has been thoroughly studied by computer vision researchers through the last part of the 20th century and up until today. A early landmark in this effort was the 1985 animation short *Tony de Peltrie* [2]. For the first time 3d facial animation was based on motion capture, also called performance-based animation.

If the "Turing test of facial animation" is the test, where a person is to judge whether an animated face is that of a real person or that of an avatar, it seems to be close to passable today. Most recently, the company *Image Metrics* specializing in performance-based facial animation presented *The Emily Project* at *SIG-*

GRAPH 2008[*]. By applying proprietaty software to a video of an actress, the actress' face was synthesized with close-to-none visible difference between the actress and her graphical alter ego.

Image Metrics is a good example of the state of the facial motion capture industry. Their technology is providing realistic facial animation to the latest triple A game titles like *Grand Theft Auto IV* and *Unreal Tournament 3*, to feature films like *The Mummy: Tomb of the Dragon Emperor*, and to commercials and music videos.

One of the subfields of facial motion capture that has received attention from the academic research community is that of *markerless* tracking. The research conducted can be accounted under the general computer vision problem of segmentation of non-rigid objects. A family of approaches to this called *deformable template models* has been developed. Among the most notable are the *active appearance models* (AAM) method [6] and the *morphable model* scheme [3].

Thanks to the well documented state of the AAM method [7], [6], [21] it is a convenient choice when applying deformable templates to facial motion capture. In this work the AAM algorithm contitutes the core *motion capture engine*.

Being a model-based method, deformable template methods rely on training data. The company *Cyberware* started providing three dimensional laser scans in the start 1990's and much of the development of face motion capture and synthesis in those years can be ascribed to their pioneering technology. However, for the un-funded parts of the research community this type of data is not available. Thus, a major part of this thesis is dedicated to describing the process of generating three dimensional face data.

1.1 Motivation and Objectives

An increasing number of laptops ship with webcam and everyday applications such as video telephony and V-logging are based on the web camera as interface. As the processing power of the household laptop keeps increasing as well, this suggests that the camera as a human-computer interface has great potential in a not so distant future. It is therefore of great relevance that technologies and methods based on the camera as a human-computer interface (HCI) are being explored and matured.

An obvious application of the webcam as an HCI is in virtual worlds. An increasing number of hours of peoples' lives are lived as avatars in online 3D universes like *World of Warcraft* and *Second Life*. A main attribute of these worlds is their function as social fora. People cooperate and communicate about solving tasks or simply to gain friendship. The communication is in most cases restricted to instant messaging

[*]Special Interest Group on GRAPHics and Interactive Techniques

and headsets. A natural extension of the communication in the virtual worlds is the animation of the avatar. In this context methods of markerless face motion capture are called for.

While the active appearance models method has been extensively applied in segmentation and tracking of objects in 2D images, and to some extent in modeling and synthesis in the 3D domain, the approach persued in this work differs from previous efforts, by attempting to train a model from sparse 3D face shape data and corresponding 2D textures, thus being capable of synthesizing sparse geometric 3D face structures from mono-view 2D textures. The approach adopts its name from exactly this capability and is in this thesis called *sparse 2D-to-3D active appearance models*.

A such approach is sensible in the application context outlined above, since conveying emotions via an avatar does not necessarily require a high geometric resolution on the side of the motion capture. One of the reasons to this is that the avatar face typically will be constrained by a physically based muscle model, and therefore possible to configure in considerable detail from a relatively sparse set of face control points.

Thus, the objectives of this thesis are:

- To discuss and summarize the process of facial motion capture
- To describe a means of acquiring 3D face data.
- To describe in particular the application of the proposed sparse 2D-to-3D AAM method in facial motion capture.
- To show that facial expressions can be captured and conveyed by the method of sparse 2D-to-3D AAM.

An additional aim of this project is to implement a modular framework for further research and exploration of sparse geometry 3D facial motion capture.

While motion capture in general can be described as the task of capturing the position and orientation of the more or less rigid structures of the human body, facial motion capture is in this project defined as the process of capturing human facial expressions. The position and orientation of the face as a whole is not considered.

1.2 Thesis Overview

This thesis is structured into three parts. The content of each part is summarized here.

Part I Theory

This part provides a theoretical background for appreciating the work of this thesis. Stereopsis, active appearance models, and animation techiques are presented.

Part II Implementation

The implementation part is is constituted by detailed descriptions of the contributions of the thesis. These are:

- a stereo face image database
- a stereo annotation tool
- the 2D-to-3D extension of the AAM approach to geometry synthesis
- an animation front end to the motion capture framework

The part is concluded by a test section.

Part III Discussion

In this part the contributions of the thesis are discussed and propositions for future work are presented.

1.3 Mathematical Notation

Unless other is specified, the following mathematical notation conventions are used throughout this thesis:

Scalar values are denoted ny lower-case Latin or Greek letters:

$$x$$

Vectors are denoted by lower-case, non-italic bold Latin or Greek letters.

$$\mathbf{x} = [x_1, x_2, ..., x_N]$$

Matrices are denoted by capital, non-italic, bold Latin or Greek Letters:

$$\mathbf{X} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

The mean vector of a specific data set is denoted by lower-case, non-italic, bold Latin or Greek letters with a bar:

$$\overline{\mathbf{x}}$$

Chapter 2

Related Work

Facial motion capture is an area that exists in the nexus of computer vision and animation. Combining expertice and techniques from both domains the synthesis of 3D facial geometry from images has become increasingly realistic over the last few decades.

Already Charles Darwin dealt with the human facial expressions in his publication *The expression of emotions of man and animals* from 1872 [8]. More than 100 years later, in 1978, Ekman and Friesen published their work on the Facial Action Coding System (FACS) [9], which decomposed all facial expressions into a set of primitives or *action units*. The work has been of profound importance to the human facial expression analysis and synthesis methods developed since, and hence, to facial animation.

Also during the 1970's, Frederic I. Parke did his pioneering work in the area of 3D computer facial animation. Among other things, he produced the first *parametric facial 3D model* in 1974 [16].

In 1980, a master thesis [18] published that included one of the first descriptions of physically based muscle controlled facial animation. Muscle simulation in animation has later been further developed (e.g.[22]) and has become one of the standard animation methods used today.

The animation short film *Tony de Peltrie* from 1985 was a landmark in 3D facial animation. The 3D facial animation in it, was based on *photogrammetric digitization* of human facial expressions, i.e. multiple view photos were used to determine the 3D surfaces of human actors' faces. Other landmarks in 3D facial animation has been *Tin Toy*(1988) from Pixar animation studio and *Casper the Friendly Ghost*(1995), and more recently *The Matrix Trilogy*.

As it appears, technology has been a key driver in the development of 3D animation techniques. In 1990, *Cyberware Laboratory* introduced their optical laser scanner, which made high resolution 3D data sets of human faces available on a commercial basis. This technology has since been a corner stone of face synthesis in the movie as well as in the game industry.

Laser scans from Cyberware was used in Blanz and Vetter's work on 3D morphable models (3DMM) from 1999 [3]. The morphable models method relies on parameterized 3D shape and texture models to synthesize faces, either from image examples of existing faces, or by controlling the model parameters to create entirely new face instances.

Blanz and Vetters method belongs to the family of deformable template models. This family includes the active appearance model method (AAM) developed by Cootes and Edwards [6]. The method carries a large resemblance to the 3DMM but makes the parameterization more compact by combining the shape and model parameters into a single *appearance model*. Though the AAM algorithm in principle is applicable in an arbitrary number of dimensions, it is designed for 2D and has mainly been explored in the context of 2D segmentation in images [7], [21].

One of the efforts to extend AAM to 3D was done in [23]. The approach is called combined 2D+3D AAM and uses a non-rigid structure-from-motion algorithm to extract 3D information from the 2D training shapes. It is shown that the combined 2D+3D AAM can represent anything that the 3DMM can.

A very interesting approach that combines computer vision and animation methods was developed in [13]. By aligning a graphical 3D face puppet controlled by muscle and anthropometric parameters to training face images, a correspondance between the image domain and the 3D puppet domain is built into the AAM.

For a further introduction to motion capture and animation's hand-in-hand development, refer to [5].

Part I

Theory

Chapter 3

Stereo Imaging

3.1 Overview

Stereopsis allows humans to perceive depth in a scene. This ability is mimiced in computer vision by a variety of algorithms and mathematical methods. In this chapter the geometrical framework of stereopsis is established. In that process the camera model, perspective projection and epipolar geometry is explained. It is shown how 3D coordinate data can be derived from a calibrated stereo camera.

3.2 The Camera Model

3.2.1 The Pinhole Camera

In its most simplified form, a camera can be described as a box with an infinitely small hole, called a *pinhole*, in it on the one side and a translucent plate on the opposite side. The image is formed by light rays emitted from the scene, entering the pinhole, and passing the translucent plate. As the light rays in this ideal and constrained scenerio are not bent by any forces, it is assumed that the rays propagate along straight lines. The image will display only the part of the scene that is placed within a *view cone* extending from the pinhole and out into the scene. The conical angle is determined by the size of the translucent plate and by the distance between the translucent plate and the pinhole. This distance is called the *focal length* of the camera. The translucent plate is from hereon referred to as the *image plane*. In Figure 3.1 the pinhole camera model is depicted.

The formulation of the pinhole camera is extended geometrically as follows. A coordinate system $(O, \mathbf{i}, \mathbf{j}, \mathbf{k})$ is attached to the pinhole model. O is the origin of the coordinate system and coincides with the pinhole. The basis vectors **i**, **j**, **k** form the 3D space in which the camera is located, where **i** and **j** form a plane parallel to the image plane, R. The image plane is located at a positive distance, f, from the pinhole. The line perpendiclur to the image plane and passing through the pinhole is called the *optical axis*, and the point, C, where the optical axis pierces the image plane is called the *image center* and is often used as the origin of the image coordinate system.

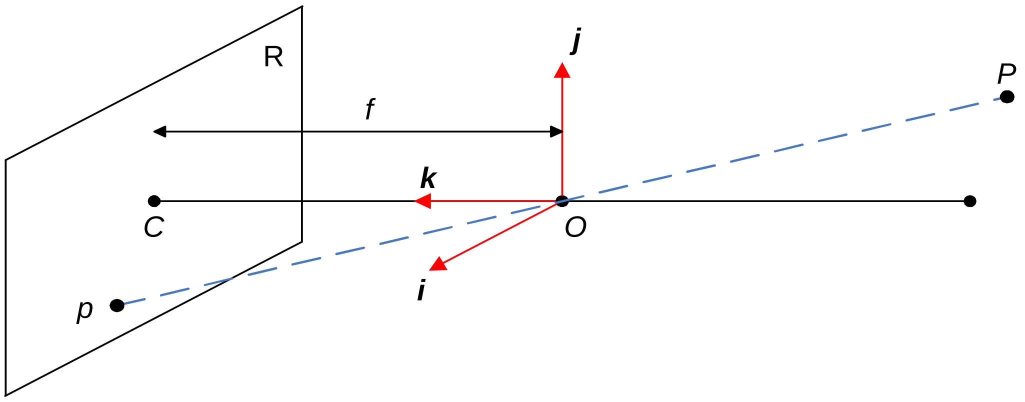


Figure 3.1: Pinhole camera model.

3.2.2 Perspective Projection

It is well known that on an image of two parallel lines extending towards the horizon, the lines will appear to intersect at the horizon. Put in another way, distances seem smaller, the further away they are from the spectator. This is what we call perspective.

A set of equations expresses the perspective projection of 3D points onto a 2D image. Let $P = (x, y, z)$ be a point in the scene, and $p = (x', y', z')$ be a point in the image. The depth, z', of the image plane is uniform and equal to the focal length, f. Since P, O and p are colinear, $\overrightarrow{Op} = \lambda\overrightarrow{Op}$ for some number λ. Hence, the projected point in the image is

$$\begin{aligned} x' &= \lambda x \\ y' &= \lambda y \\ z' = f &= \lambda z \end{aligned} \iff \lambda = \frac{x'}{x} = \frac{y'}{y} = \frac{f}{z} \tag{3.1}$$

and therefore

$$\begin{aligned} x' &= f\frac{x}{z} \\ y' &= f\frac{y}{z} \end{aligned} \tag{3.2}$$

3.2.3 Intrinsic Parameters

Since the pinhole is a point, exactly one light ray will pass through it and hit the translucent plate for each scene point within the camera view. However, no such thing as a hole the size of a point exists, and in the world beyond that of mathematics, we are forced to replace the non-existing point hole with an optical lens. Not surprisingly, this introduces various sources of imperfection.

For digital cameras the image plane is partitioned into pixels. In the following this plane is referred to as the *digital image*.

The pixels are generally rectangular but not necessarily square. Therefore a set of scale parameters, k and l, one for each image dimension, is introduced. The unit of the scale parameters is $\frac{pixel}{meter}$.

Let a pixel have the digital image coordinates (u, v) in pixel units. Considering the scale parameters, the image coordinates can be transformed from the (x', y') positions of the pinhole camera into pixel units.

$$\begin{aligned} u &= kf\frac{x}{z} \\ v &= lf\frac{y}{z} \end{aligned} \tag{3.3}$$

Generally, the origin of a digital image coordinate system is the top or bottom left corner, and not the image center as defined for the pinhole camera above. In addition to being scaled, the image coordinates therefore need to be translated by a value corresponding to the digital image center. If the digital image center is $C = (u_c, v_c)$, then the digital image coordinates in pixels units become

$$\begin{aligned} u &= kf\frac{x}{z} + u_c \\ v &= lf\frac{y}{z} + v_c \end{aligned} \tag{3.4}$$

The final intrinsic parameter to consider, when disregarding lens distortion, is *skew*. The skew, S, is defined as cosine to the angle between the digital image axes, which due to manufacturing errors is not always $90°$.

$$\begin{aligned} u &= kf\frac{x}{z} - kf\cot(\theta)\frac{y}{z} + u_c \\ v &= \frac{lf}{\sin(\theta)}\frac{y}{z} + v_c \end{aligned} \tag{3.5}$$

Now, if a point in a *normalized image plane*, parallel to the pinhole camera image plane but with unit focal length, is defined as follows

$$\begin{aligned} \hat{u} &= \frac{x}{z} \\ \hat{v} &= \frac{y}{z} \end{aligned} \tag{3.6}$$

and the homogenous vector $\hat{p} = (\hat{u}, \hat{v}, 1)^T$, then Eq. (3.5) can be expressed as a linear transformation between the normalized image and the digital image.

$$p = \mathbf{K}\hat{p}, \quad \text{where} \quad p = \begin{pmatrix} \hat{u} \\ \hat{v} \\ 1 \end{pmatrix} \quad \text{and} \quad \mathbf{K} = \begin{pmatrix} kf & -kf\cot(\theta) & u_c \\ 0 & \frac{lf}{\sin(\theta)} & v_c \\ 0 & 0 & 1 \end{pmatrix} \tag{3.7}$$

Next, consider how the normalized image point $\hat{p}$ can be expressed in terms of the scene point $P = (x, y, z)^T$.

$$\hat{p} = \frac{1}{z}\begin{pmatrix} Id & \mathbf{0} \end{pmatrix}\begin{pmatrix} P \\ 1 \end{pmatrix} \tag{3.8}$$

Combining Eq. (3.7) and Eq. (3.8) allows for a system describing the entire transformation from scene point to digital image point

$$p = \frac{1}{z}\begin{pmatrix} \mathbf{K} & \mathbf{0} \end{pmatrix}\begin{pmatrix} P \\ 1 \end{pmatrix} \tag{3.9}$$

In homogenous coordinates the *perspective projection matrix* is

$$\mathbf{M} = \begin{pmatrix} \mathbf{K} & \mathbf{0} \end{pmatrix} = \begin{pmatrix} kf & -kf\cot(\theta) & u_c & 0 \\ 0 & \frac{lf}{\sin(\theta)} & v_c & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{3.10}$$

3.2.4 Extrinsic Parameters

Having established the notion of a perspective projection matrix we are ready to extend this to involve *extrinsic parameters* as well. The extrinsic parameters $\mathbf{H}$ are, as the name suggests, a set of parameters describing to the camera's relation to the world outside itself, to be more exact, the camera's position in the world.

$$\mathbf{H} = \begin{pmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0} & 1 \end{pmatrix} \tag{3.11}$$

where $\mathbf{R}$ is the rotation matrix aligning the camera's coordinate system with the world's coordinate system, and $\mathbf{t}$ is the translation vector that moves the optical center of the camera to the world origin.

Combining $\mathbf{M}$ and $\mathbf{H}$ leads to the general formulation of the perspective projection matrix, $\mathcal{M}$.

$$\mathcal{M} = \mathbf{M}\cdot\mathbf{H} \tag{3.12}$$

3.3 Stereopsis

Obtaining 3D data from images requires several images taken from more than one position. No depth information is available from a single image, but with at least two images taken from different positions depth information becomes available through triangulation. Multiple images from different positions will constrain the 3D structure of the pictured scene. In the case of stereo vision the contraint is called the *epipolar constraint* (see Section 3.3.2). The epipolar constraint is used in the calibration of a stereo view.

3.3.1 Calibration

The goal of calibration is here to estimate both cameras' positions and orientations in the world and their internal distortion parameters, which is exactly the extrinsic and intrinsic parameters described in Section 3.2.4 and 3.2.3.

Calibration of a camera is essentially an optimization process, that minimizes the difference between the observed image features and their predicted positions, with respect to the intrinsic and extrinsic parameters. It is not in the scope of this thesis to go into the details of calibration, which is a vast field of research that has resulted in a myriad of different methods. For an introduction to calibration approaches refer to [10].

Having a calibrated stereo camera it is possible to obtain 3D information about the scene points that are included in both images. A common approach to this is *rectification*. Rectification can be understood as the warping of one image into the coordinate frame of the other. The purpose is to align the image data such that corresponding scene points are placed on the same row in the pixel matrix. In the following section rectification and its basis in *epipolar geometry* is introduced.

3.3.2 Epipolar Geometry

Consider a stereo rig composed by two pinhole cameras as shown in Figure 3.2. C_{left} and C_{right} are the optical centers of the left and right cameras, respectively. A scene point P is projected onto both image planes, to points p_{left} and p_{right}. The points p_{left} and p_{right} are said to constitute a *conjugate pair*.

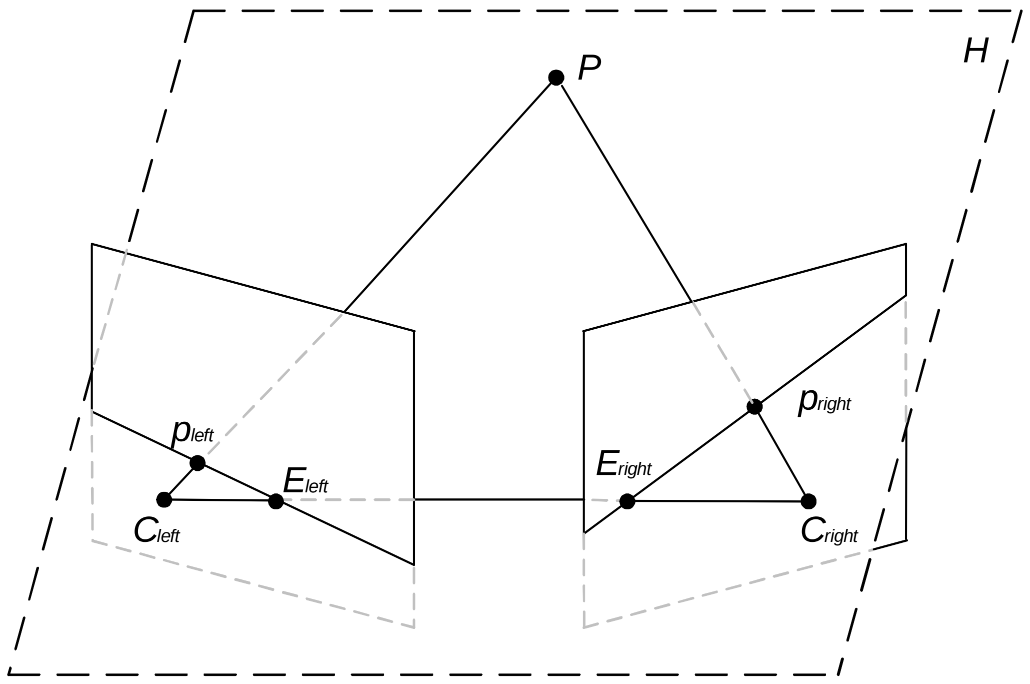


Figure 3.2: Epipolar geometry.

The *optical ray* of p_{left} is the halfline going from C_{left} through P. It is clear that p_{left} may be the 2D projection of any scene point along the optical ray of p_{left}. However, the optical ray of p_{left} helps constrain the position of its conjugate point on the right image in the following way: If a plane, H, is formed by the optical ray of p_{left} and the *baseline*, B, which is the line going from C_{left} to C_{right}, then the intersection of this plane and the right image is called the *epipolar line* of p_{left}. The conjugate point of p_{left} in the right image, p_{right}, is constrained to lie on the epipolar line of p_{left}.

All the epipolar lines in one image plane pass through a common point, E, called the *epipole*. The epipole of an image plane is the projection of the optical center of the other camera onto the image plane.

3.3.3 Rectification

The *focal plane* of a camera is the virtual plane parallel to the image plane that contains the optical center. When C_{left} is in the focal plane of the right camera, the right epipole is at infinity, and the epipolar lines form a bundle of parallel lines in the right image.

A special case is when both epipoles are at infinity. That happens when the baseline, $C_{left}\ C_{right}$, is contained in both focal planes, i.e., the image planes are parallel to the baseline. Epipolar lines then form a bundle of parallel lines in both images. Any pair of images can be transformed such that epipolar lines are parallel and horizontal in each image. This transformation is called rectification [11].

Having a calibrated stereo rig, that is, knowing the perspective projection matrices $\mathbf{P}_{old_1}$ and $\mathbf{P}_{old_2}$ of the two cameras, rectification is basically the process of defining two new projection matrices $\mathbf{P}_{new_1}$ and $\mathbf{P}_{new_2}$. The new projection matrices are obtained by rotating the old ones around their optical centers until the focal planes becomes coplanar, thereby containing the baseline. This ensures that the epipoles are at infinity and epipolar lines are parallel.

To align the epipolar lines horizontally, the baseline must be parallel to the new x-axis of both cameras. In addition, conjugate points must have the same vertical coordinate. This is obtained by requiring that the new projection matrices have the same intrinsic parameters. For an algorithmic mathematical description of rectification refer to [11].

3.3.4 Triangulation

On a pair of rectified stereo images, the depth of a scene point, P, can be calculated by triangulation, if the conjugate pair $\{p_{left}, p_{right}\}$ of P's projection onto the two image planes, is known. In the following section the triangluation equation is derived.

Remember, that the optical ray of an image point is the halfline starting at the camera's optical center, C, and going through P. The linear extrapolation of the optical ray in the opposite direction of P will pass through the image point itself. Hence, two points on each linear extrapolation of the optical rays are known: the optical center and the image point.

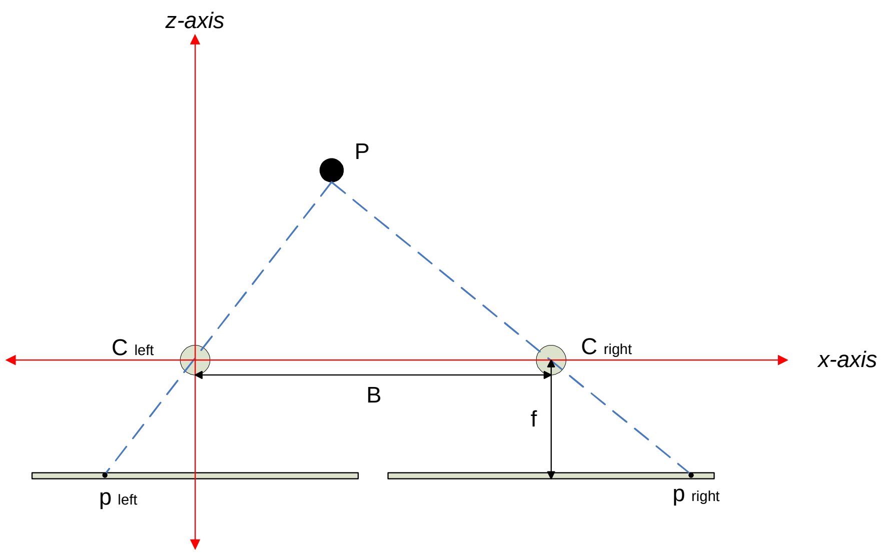


Figure 3.3: Triangulation coordinate system.

Since it is known that the left and right image points are horizontally aligned (due to rectification), the y-dimension can be disregarded. Thus, the points can be considered as being two dimensional, with an x- and a z-component.

By convention the optical center of the left camera is set as the origin of the triangulation coordinate system, with the optical axis aligned to the z-axis. The translation between the optical centers of the two cameras is denoted B, and is one dimensional (along x-axis only).

Since the images are rectified they have identical intrinsic parameters, including focal lengths. The z-components of the image points are equal to the focal length, f, of the cameras.

If d_{left} and d_{right} are the distances between the optical centers and the projections of P in the left and right image, respectively, then the *disparity* of a rectified stereo view is defined by $d = d_{left} - d_{right}$. Now the two points on each optical ray can be defined:

$$\begin{aligned} C_{left} &= (0,0) \quad p_{left} = (d_{left}, -f). \\ C_{right} &= (B,0) \quad p_{right} = (B + d_{right}, -f). \end{aligned} \tag{3.13}$$

As a straight line is fully defined by two points on the line, we can find the optical rays of the two cameras:

$$a_{left} = \frac{-f-0}{d_{left}-0} \qquad b_{left} = 0 \tag{3.14}$$

$$a_{right} = \frac{-f-0}{(B+d_{right})-B} \qquad b_{right} = \frac{f}{d_{right}}B \tag{3.15}$$

The two optical rays in the xz-plane are thus given by:

$$z = \frac{-f}{d_{left}} \cdot x \tag{3.16}$$

$$z = \frac{-f}{d_{right}} \cdot x + \frac{f}{d_{right}}B \tag{3.17}$$

Now, the depth of P can be obtained by calculating the crossing point of the two optical rays. Setting the two expressions equal yields:

$$\begin{aligned} z = \frac{-f}{d_{left}}x &= \frac{-f}{d_{right}}x + \frac{f}{d_{right}}B \Leftrightarrow \\ x \cdot \left(\frac{-f}{d_{left}} + \frac{f}{d_{right}} \right) &= \frac{f}{d_{right}}B \Leftrightarrow \\ x \cdot \left(\frac{-f \cdot d_{right}}{d_{left} \cdot d_{right}} + \frac{f \cdot d_{left}}{d_{left} \cdot d_{right}} \right) &= \frac{f}{d_{right}}B \Leftrightarrow \\ x \cdot \left(\frac{-f \cdot d_{right} + f \cdot d_{left}}{d_{left} \cdot d_{right}} \right) &= \frac{f}{d_{right}}B \Leftrightarrow \\ \frac{\frac{f}{d_{right}} \cdot B \cdot d_{left} \cdot d_{right}}{-f \cdot d_{right} + f \cdot d_{left}} &= x \Leftrightarrow \\ \frac{f \cdot B \cdot d_{left}}{f \cdot d_{left} - f \cdot d_{right}} = \frac{B \cdot d_{left}}{d_{left} - d_{right}} &= x \end{aligned} \tag{3.18}$$

The result of (3.18) is inserted into the equation for the left line (3.16):

$$z = \frac{-f}{d_{left}} \cdot \frac{B \cdot d_{left}}{d_{left} - d_{right}} = \frac{-f \cdot B}{d} \tag{3.19}$$

It is instructive in this way to understand the relationship between the image points and the scene point. The result derived in Eq.(3.19) for z can, together with simular results for the x- and y-coordinates of P, be collected in a so-called *reprojection* matrix.

3.3.5 Reprojection

An image point, p, of the left image can be reprojected out into 3D space if the disparity d is known. The reprojection matrix is defined by $\mathbf{Q}$.

$$\mathbf{Q} = \begin{pmatrix} 1 & 0 & 0 & C_x \\ 0 & 1 & 0 & C_y \\ 0 & 0 & 0 & f \\ 0 & 0 & -\frac{1}{B} & 0 \end{pmatrix} \tag{3.20}$$

If the image point is arranged in a homogenous vector together with the disparity, the 3D point is

$$P = \begin{pmatrix} X \\ Y \\ Z \\ W \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & -C_x \\ 0 & 1 & 0 & -C_y \\ 0 & 0 & 0 & f \\ 0 & 0 & -\frac{1}{B} & 0 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ d \\ 1 \end{pmatrix} \tag{3.21}$$

where the actual 3D point is X/W, Y/W, Z/W.

3.4 Facial Features

A prerequisite for calculating the 3D position of an image point is of course to find the conjugate point in the other stereo image. This problem is known in stereopsis as *the correspondance problem* [10]. As described in above the epipolar geometry constrains the search for a conjugate point. A point in the second image will be located on the epipolar line of its conjugate point in the first image, and vice versa.

In the special case of rectified images where the epipolar lines are aligned with the image baseline and conjugate point pairs have the same y-coordinate, the correspondance problem is reduced to a template matching problem in the second image along the row of the point in the first image. To accomdate template matching, points of high curvature, or *texturedness*, are preferrable as feature points [20]. See Figure 3.4.

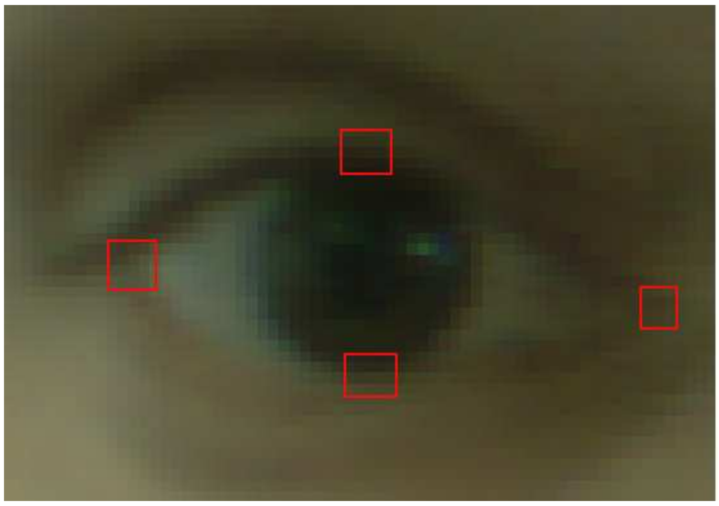


Figure 3.4: Four typical feature points around an eye. The points are of relatively high curvature compared with surroundings.

Chapter 4

Active Appearance Models

4.1 Overview

AAM is a statistical learning method, trained on instances of a certain class of non-rigid (or rigid) objects, here faces. It works by fitting a parameterized model to an image of certain type of object by an optimization search. The goal of the search is to reach a parameterized description of the object in the image. AAM belongs to the family of *deformable template models*, and can be understood as a method of advanced template matching.

After a brief introduction to the concepts of *shapes*, this chapter explains AAM in three sections of the method's overall steps:

1. Alignment of shape data
2. Generation of a statistical parameterized model from the data
3. AAM search, the actual "template matching" procedure.

The AAM method is designed for the purpose of image segmentation and its general formulation therefore applies to two dimensions. But though the method here is described for 2D face modelling, it should be noted that it, at least in principle, is valid for in any type of object, in any number of dimensions [7].

4.2 Shapes

A 2D shape is defined, in this work, as the 2D vertices and edges that describe the spatial structure of an object. In theory, the number of vertices and edges may be infinite, while practically, vertices are chosen such that they optimally, by some measure, at some level of resolution, describe the shape.

Dealing with *shape classes*, in this thesis human faces, *landmarks* are introduced. In general, landmarks, or *feature points* as they are often called, are points of high curvature, as described in Section 3.4. Furthermore, landmarks correspond semantically over all instances of the class. For triangles landmarks might be the triangle corners, for faces they could be eye corners, mouth corners, points along the eye brows and so on.

Note here, that shapes thus can describe both rigid and non-rigid objects as long as the landmarks are consistent within the shape class. In the framework of AAM this implies, that a model trained on frontal face shapes are capable of describing faces where *all* frontal face landmarks are visible but *not*, say, profile face shapes.

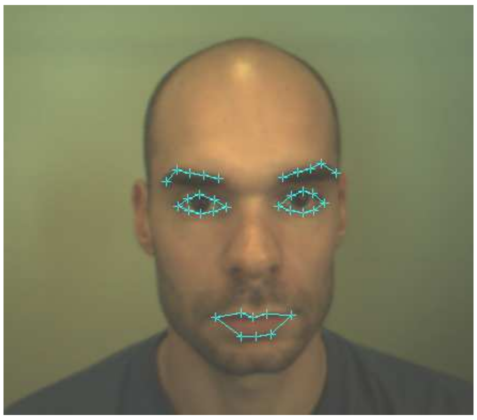


Figure 4.1: A face image with landmarks and shape plotted onto it.

Mathematically, a 2D shape can be described by a vector, $\mathbf{x}_i$, concatenating the x- and y-coordinates of the landmark point data:

$$\mathbf{x}_i = [x_1, x_2, \cdots, x_N, y_1, y_2, \cdots, y_N] \tag{4.1}$$

where N is the number of landmarks in the shape.

4.3 Shape Alignment

In geometry shapes are usually considered to be independent of translation, rotation and scale [4]. That is, two shapes are the same, if one of them, by any combination of translation, rotation and scaling, can be transformed into the other. To obtain a model from a set of shapes some normalization is initially called for. For this, AAM adopts the method of Procrustes analysis.

Procrustes analysis is a method of statistical shape analysis that can be used to align shapes. It does so by removing the translation, rotation and scaling components. Aligning two shapes using the Procrustes metric takes the following steps:

- 1. Translate each shape by subtracting the shape's center of gravity (COG) from all N points i in shape.

$$(x_i', y_i') = \big((x_i - \overline{x}), (y_i - \overline{y})\big) \tag{4.2}$$

where

$$(\overline{x}, \overline{y}) = \left(\frac{1}{N}\sum_i x_i, \frac{1}{N}\sum_i y_i\right) \tag{4.3}$$

- 2. Scale each shape, $\mathbf{x}_k$, to unit size by L2-norm.

$$(x_i'', y_i'') = \frac{1}{|\mathbf{x}_k|}(x_i', y_i') \tag{4.4}$$

where

$$|\mathbf{x}_k| = \sqrt{\sum_i \left(x_i^2 + y_i^2\right)} \tag{4.5}$$

Removing the rotation component between two shapes is a bit more complex. The following technique, using singular value decomposition (SVD), was suggested by [4]:

- 3.1. Arrange two shapes, $\mathbf{x}_1$ and $\mathbf{x}_2$, normalized as described in step 1 and 2, in two N x 2 matrices.

- 3.2. Calculate the SVD of $\mathbf{x}_1^T\mathbf{x}_2, \mathbf{VDU}^T$

- 3.3. The rotation matrix that optimally superimposes $\mathbf{x}_1$ upon $\mathbf{x}_2$ is then:

$$\mathbf{VU}^T = \begin{bmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{bmatrix} \tag{4.6}$$

To align a set of shapes the following iterative approach was suggested by [21]:

- 1. Choose a shape arbitrarily to be an estimate of the mean shape.
- 2. Align each remaining shape to the mean shape by the steps described above.
- 3. Re-calculate the mean estimate from the aligned shapes.
- 4. If first iteration, then the orientation of the new mean estimate is corrected, such that the rotation of the training shapes in order to be aligned with the new mean estimate is on average zero.
- 5. Subtract the new mean estimate from the old mean estimate. If difference is above threshold go to 2.

4.4 Statistical modeling

The AAM method relies on a parameter model that can describe intra-class shape and texture variations. The core part of building an appearance model is principal component analysis (PCA). The PCA performs an Eigen-analysis of the covariance matrices of the aligned training shape vectors and of the textures defined by the shapes in the training images. The Eigenvectors obtained in the analysis thus describe the shape and texture variations. Distinct PCA's are performed on shape and texture data, followed by yet a PCA on the combined shape and texture models.

4.4.1 Shape Model

If $\mathbf{x}_k$ is a shape vector of length N and M is the number of shape samples from a class, then this set, or *collection*, of training shapes can be described by an N x M matrix

$$\mathbf{S} = \Big[(\mathbf{x}_1 - \overline{\mathbf{x}})(\mathbf{x}_2 - \overline{\mathbf{x}}) \cdots (\mathbf{x}_M - \overline{\mathbf{x}})\Big], \quad \text{where } \overline{\mathbf{x}} = \frac{1}{M}\sum_{k=0}^{M} \mathbf{x}_k \text{ is the mean shape}$$

The covariance of the shape data set is then expressed by the N x N matrix $\mathbf{\Sigma}_s = \frac{1}{N}\mathbf{S}\mathbf{S}^T$. Eigen-analysis of $\mathbf{\Sigma}_s$ concludes the principal component analysis:

$$\boldsymbol{\Sigma}_s\Phi_s = \Phi_s\lambda_s \tag{4.7}$$

Φ_s and λ_s are the shape collection's Eigenvectors, organized as columns in a matrix, and Eigenvalues, placed on the diagonal of a diagonal matrix, respectively. The Eigenvectors represent a new N-dimensional orthogonal basis for the shape data, where the variance is maximized. Each dimension in the new basis optimally represents the modes of shape variation within the training set. The Eigenvalues quantify the variance and thus the significance of their corresponding Eigenvectors.

As a shape belongs to a certain class of objects a great degree of redundancy can be expected. By ordering the set of Eigenvectors according to their respective Eigenvalues and discarding Eigenvectors with close to zero significance, the dimensionality of the new space is likely to be substantially reduced. This dimensionality reduction is treated further in the section 4.4.2. For now, consider how a new shape instance, $\mathbf{x}$, can be generated by applying an N-dimensional parameter vector, $\mathbf{b}_s$:

$$\mathbf{x} = \overline{\mathbf{x}} + \Phi_s b_s \tag{4.8}$$

The parameters, $\mathbf{b}_s$, can be interpreted as weights determining how much each of the shape modes should be present in a new instance of the model.

4.4.2 Texture Model

In the framework of AAM, a texture is defined as the set of pixel intensity values across the object framed by the shape landmarks. As with the shape, the texture variations can also be modeled using PCA. The steps towards obtaining a vector describing the texture of an object are somewhat more intricate than for shapes, since such a vector must be spatially consistent over all training images. For this purpose warping is used.

Warping is basically a transformation of one spatial configuration into another. One way of establishing a spatial reference between the training samples is by Delaunay triangulation of the landmark points in each image. By superimposing a mesh of triangles upon the texture, the position of each pixel can be described as a weighted linear combination of the three landmarks that constitute the triangle that surrounds the pixel.

By warping the textures of all objects into one reference mesh, a set of spatially consistent, shape-normalized, textures is obtained. This allows calculating the mean and variance statistics. For more on warping refer to [21].

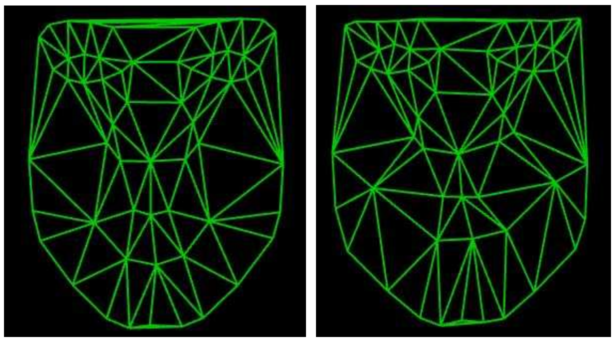


Figure 4.2: Two Delaunay triangulated meshes used for texture warping. (Note that, the example here is based on a different point annotation model than the one generally used in this project).

Each texture is organized in a vector of size K, where K is the number of pixels inside the area covered by the reference shape. By subtracting mean and inserting all texture vectors into a K x M matrix, $\mathbf{G}$, where M is the number of texture samples, PCA can be applied in a straight forward manner as described for the shapes, refer to Section 4.2. The result is an orthogonal basis matrix, Φ_g, of texture intensity modes that can be controlled by a parameter vector, $\mathbf{b}_g$:

$$\mathbf{g} = \overline{\mathbf{g}} + \Phi_g \mathbf{b}_g \tag{4.9}$$

where $\overline{\mathbf{g}}$ is the mean texture.

As the number of pixels in a detailed texture will not be small, the dimensionality of the texture space is often considerably higher than the number of sample textures. In this case the *Eckart-Young theorem* can be applied, to greatly reduce the dimensionality. The trick of the Eckart-Young theorem lies in the calculation of the covariance matrix $\mathbf{\Sigma}_g$. Having a matrix, $\mathbf{G}$, of dimensions K x M , $\mathbf{\Sigma}_{g_{large}} = \frac{1}{M}\mathbf{G}\mathbf{G}^T$ will be of dimension K x K. Instead one can calculate $\mathbf{\Sigma}_{g_{small}} = \frac{1}{M}\mathbf{G}^T\mathbf{G}$, of dimension M x M. The Eckart-Young theorem then shows that

$$\mathbf{\Sigma}_{g_{large}} = \mathbf{G}\mathbf{\Sigma}_{g_{small}} \tag{4.10}$$

4.4.3 Combined Model

To further remove redundancy and to combine the shape and texture models into one compact parameterized *appearance* model, yet a PCA is applied. This time the PCA is performed on the parameter vectors, $\mathbf{b}_s$ and $\mathbf{b}_g$, describing the training objects. A new orthogonal basis, Φ_c, that can be controlled by a parameter vector, $\mathbf{c}$, is thus obtained:

$$\mathbf{b} = \Phi_c \cdot \mathbf{c} \tag{4.11}$$

To compensate for the difference in units between the parameter vectors (one being in 2D coordinates, the other in grayscale intensity values), a weight matrix, $\mathbf{W}_s$ is multiplied with the shape parameters, $\mathbf{b}_s$. The complete model can be described in the following fashion:

$$\mathbf{b} = \begin{pmatrix} \mathbf{W}_s \mathbf{b}_s \\ \mathbf{b}_g \end{pmatrix} = \begin{pmatrix} \mathbf{W}_s \Phi_s^T (\mathbf{x} - \overline{\mathbf{x}}) \\ \Phi_g^T (\mathbf{g} - \overline{\mathbf{g}}) \end{pmatrix} = \begin{pmatrix} \Phi_{c_s} \\ \Phi_{c_g} \end{pmatrix} \cdot \mathbf{c} = \Phi_c \cdot \mathbf{c} \tag{4.12}$$

A straight forward way of estimating the weight, $\mathbf{W}_s$, is proposed by [21]. The weight should normalize the ranges of the pixel coordinate values and the pixel intensity values. This range information is described by the variances, which is already calculated above, as the Eigenvalues of the shape and texture data. Hence, we have

$$\mathbf{W}_s = r \cdot \mathbf{I}, \quad \text{where} \quad r = \frac{\lambda_g}{\lambda_s}, \quad \lambda_g = \sum_i \lambda_{g_i}, \quad \lambda_s = \sum_i \lambda_{s_i} \tag{4.13}$$

For any given set of model parameters, c, the shape and texture can now be generated by

$$x = \overline{x} + \Phi_s W_s^{-1} \Phi_{c,s} \cdot c \tag{4.14}$$

and

$$g = \overline{g} + \Phi_g \Phi_{c,g} \cdot c \tag{4.15}$$

Where

$$\Phi_c = \begin{pmatrix} \Phi_{c,s} \\ \Phi_{c,g} \end{pmatrix} \tag{4.16}$$

This completes the AAM model description. In the next section the AAM optimization search is considered.

4.5 AAM Optimization

Finding a shape of a certain class in a new image is an optimization problem. The objective is to minimize the difference between the input image and the texture instance synthesized by the AAM model. To accomodate this, a difference vector is defined:

$$\delta\mathbf{g} = \mathbf{g}_i - \mathbf{g}_m \tag{4.17}$$

Here $\mathbf{g}_i$ is the vector of grayscale intensity values of the region in the input image that is covered by the current model shape, and $\mathbf{g}_m$ is the vector of grayscale intensity values of the current model texture. See Figure 4.3.

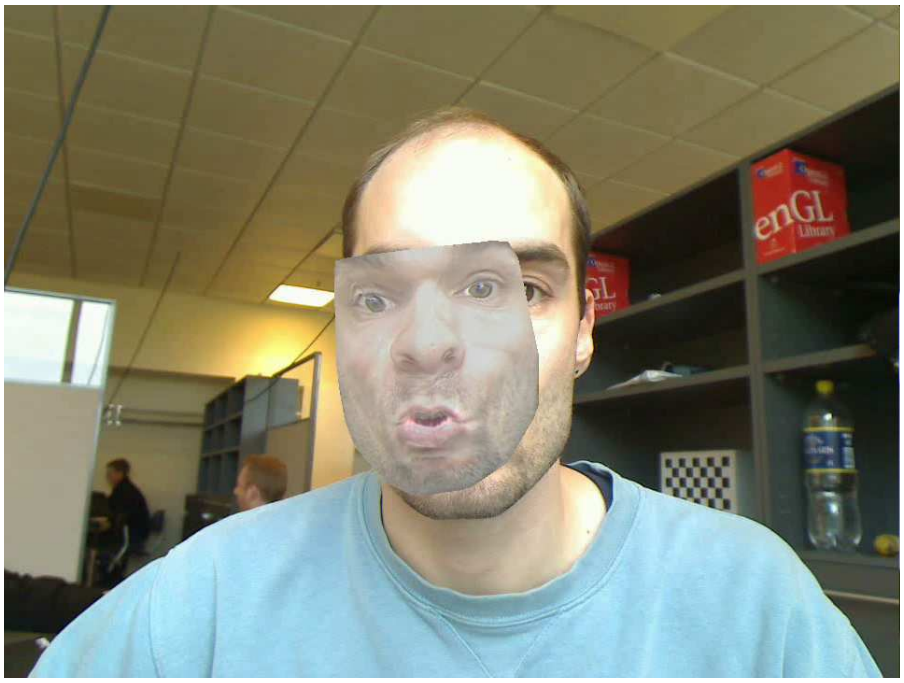


Figure 4.3: An AAM model instance. The goal of the AAM optimization is to minimize the difference between the model texture and the image texture covered by the model shape.

Since each attempt to fit the AAM model to a new input image follows the same optimization procedure, Cootes [7] proposes that a prediction model that governs the adjustments applied to the AAM model parameters for a better fit, can be built offline from the training data.

4.5.1 Offline Parameter Prediction Optimization

Fitting a model to an object in an image, it is assumed that a linear relationship exists between the needed adjustments in model parameters, $\delta\mathbf{c}$, and the error, $\delta\mathbf{g}$. Cootes show in [7] how this is a valid assumption. The relationship is expressed by

$$\delta\mathbf{c} = \mathbf{R}\cdot\, \delta\mathbf{g} \tag{4.18}$$

As an AAM model instance is composed of both its model parameters, $\mathbf{c}$, and its *pose*, $\mathbf{t}$, including position, scale and rotation, the above assumption of linearity is extended to both these domains.

Hence, two models must be built: One, describing the relationship between the change in pose parameters of the model instance, $\delta\mathbf{t}$, and $\delta\mathbf{g}$. This model is denoted $\mathbf{R}_t$. The other, describing the relationship between the change in model parameters, $\delta\mathbf{c}$, and $\delta\mathbf{g}$. Thi model is denoted $\mathbf{R}_c$.

To build the $\mathbf{R}_t$ and $\mathbf{R}_c$ prediction models, a set of experiments is conducted. Each experiment consists of displacing and deforming the AAM model by applying known changes to the pose and model parameters. An in-depth treatment of the generation of the prediction model [21] .

4.5.2 Iterative Model Refinement

Applying the AAM model on an input image stream will return an estimate of the shape and texture parameters for each frame. The procedure is described in the following.

Given an input image and some prior initialization step, an estimate of a model instance, $\mathbf{c}_0$, is found in the image. By applying Eq. (4.14) and Eq. (4.15) a the shape and texture of $\mathbf{c}_0$ are calculated. The image is then sampled over the area in the image covered by the shape of $\mathbf{c}_0$. By subtracting the sampled texture by the texture of $\mathbf{c}_0$, the difference vector, $\delta\mathbf{g}_0$, is calculated. Using $\delta\mathbf{g}_0$ and the prediction model, $\mathbf{R}_c$ and $\mathbf{R}_t$, we can predict the displacements, $\delta\mathbf{c}$ and $\delta\mathbf{t}$, that will optimize the model fitness to the image.

Below, a sequence of steps summarize how the optimization is done for the model parameters. A simular sequence applies to the pose parameters.

1. Evaluate the difference vector $\delta\mathbf{g}_0 = \mathbf{g}_s - \mathbf{g}_m$
2. Evaluate the current error $E_0 = |\delta\mathbf{g}_0|^2$
3. Compute the predicted displacement, $\delta\mathbf{c} = \mathbf{R}_c\delta\mathbf{g}_0$
4. Set $k = 1$
5. Let $\mathbf{c}_1 = \mathbf{c}_0 k\delta\mathbf{c}$
6. Sample the image at this new prediction, and calculate a new difference vector, $\delta\mathbf{g}_1$

7. If $|\delta \mathbf{g}_1|^2 < E_0$, then accept the new estimate, $\mathbf{c}_1$
8. Otherwise try at $k = 1 : 5$, $k = 0 : 5$, $k = 0 : 25$ etc.

The above steps are performed iteratively until no improvement is observed in the model fit.

In Figure 4.4 an example of an AAM search is shown. It is seen how the textured shape is fitted to an input face image over several iterations.

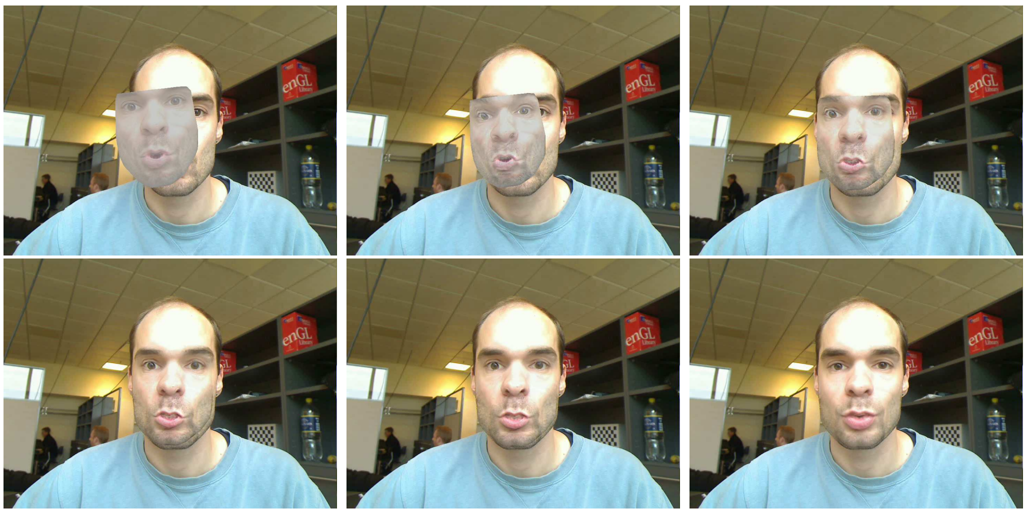


Figure 4.4: Six iterations of an AAM search.

Chapter 5

3D Facial Animation

5.1 Overview

The goal of facial animation is to manipulate the surface of a graphic face representation over time, such that desired facial expressions appear. This chapter gives a brief introduction to the basics of facial surface modeling and introduces a group of facial animation techniques known as *performance animation*, where face animations are directly or indirectly are controlled by human actors.

5.2 Face Modeling

The geometry of a 3D face model and the models animation capabilities are inseparably intertwined. The actions that a model must perform determines the face models geometric structure. The most common representation of 3D graphic objects is by *Polygonal surfaces*. Other ways of describing graphical surfaces include *implicit* and *parametric* surfaces.

The polygonal *topology* refers to the way the polygons of a surface are connected. The topology may be either in the form of a *regular mesh* or as an *arbitrary network*, see Figure 5.1. In a regular mesh the polygon vertices are aligned in rectangular arrays, in an arbitrary network the polygons are connected as needed for a given modeling or animation purpose.

Developing a face topology, one must consider the following [15]:

- Polygon structure must allow natural face-like flexibility

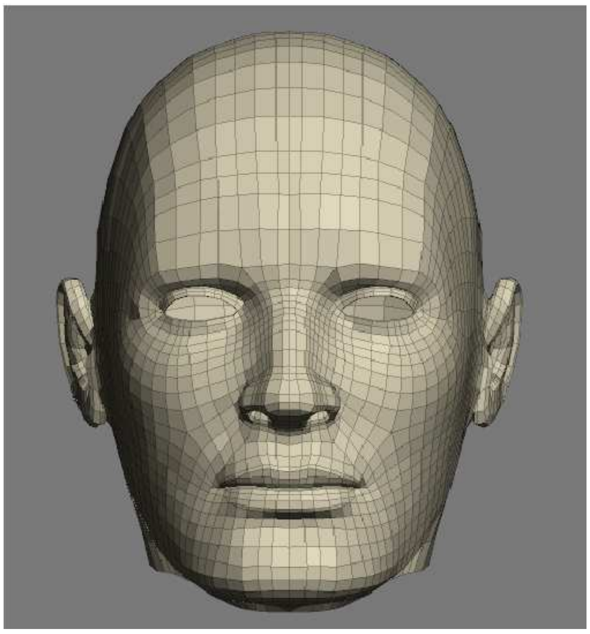


Figure 5.1: Example of a polygonal surface. The topology is an *arbitrary mesh* structure.

- Areas of high curvature need high density of polygons to define the surface
- Areas that communicate emotional information, especially mouth and eye surroundings, must be carefully structured to allow detail
- Polygon edges must coincide with natural creases and color boundaries of face
- Polygon structure must be symmetric
- Use the smallest number of polygons consistent with a satisfying result

5.3 Facial Animation

Animation of a face is the process of, directly or indirectly, manipulating the position of the polygons' vertices over time. Several methods exists for this purpose.

In parameterized animation, a set of parameters control the facial expressions. The tools that implement this type of animation will often represent the parameters as a set of sliders, that can be adjusted by the animator to obtain the wanted expression. Another commonly used approach is muscle based animation, where a physical model describes the relation between muscles and the face surface. Specifying how each muscle expand and contract, the vertices of the skin can be manipulated. The probably most widely used method used in animation is interpolation between key-expressions. The positions of all vertices are specified for a set of key-frames. After that, a facial expression can be generated by interpolating the key-frames, i.e. interpolating the positions of the vertices in the key-frames.

5.3.1 Performance Animation

In this project a fourth method called *performance animation* is considered. Performance animation relies on information obtained by measuring human actions, for instance via a camera. The information retrieved is then used to control the graphic character. The method is also referred to as *digital puppetry*.

By making a correspondance between the neutral expression of a human *puppeteer* face and the neutral expression of a graphic *puppet* face, the graphic face can be controlled. In 11.3 a such correspondance can be seen.

A mapping of extracted human face *landmarks* to corresponding points in the graphic face can be set up. However, as the number of landmarks retrieved from the human face is likely to be much smaller than the number of vertices comprising the graphic face, a 1-to-many correspondance is established. A damping factor can be applied to the vertices associated to a certain feature point, based on their distance to the vertex directly corresponding to the feature point [15].

Now, to capture the performance of the human actor, the displacements of the human face landmarks relative to their neutral starting points can be recorded. These displacements are used to perform corresponding displacements of the vertices in the graphic face.

Performance animation techniques have merged into the field more generally known as motion capture. Motion capture is increasingly applied in the animation and computer game industry, mainly because it saves production time. The motion capture animation techniques often relies on physical models of muscles and skin to both constrain and provide the proper degree of realism/exaggeration in the facial expression. Physically based muscle controlled animation has not been considered in this project. For a classic article on the subject, refer to [22].

Part II

Implementation

Chapter 6

Introduction

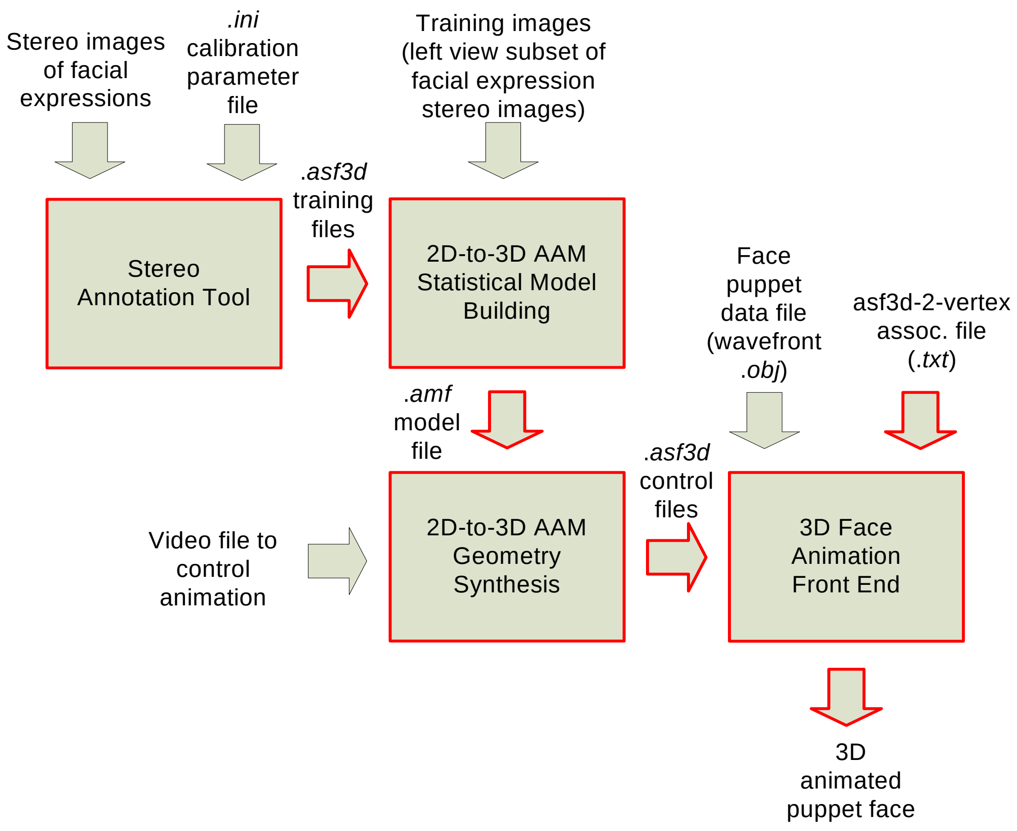

Stereo images of facial expressions
.ini calibration parameter file
Training images (left view subset of facial expression stereo images)
.asf3d training files
Stereo Annotation Tool
2D-to-3D AAM Statistical Model Building
Face puppet data file (wavefront *.obj*)
asf3d-2-vertex assoc. file (*.txt*)
.amf model file
.asf3d control files
Video file to control animation
2D-to-3D AAM Geometry Synthesis
3D Face Animation Front End
3D animated puppet face

Figure 6.1: Face motion capture pipeline. Entities marked by a red outline either part of the motion capture framework software, or produced by it

In this part of the thesis, the implementational details of the face motion capture framework established are treated in detail. The description is partitioned into a series chapters describing the framework pipeline. Ths pipeling is illustrated in Figure 6.1.

Three distinct pieces of software have been produced to establish a complete motion capture framework. First, an annotation tool by which 3D data can be obtained from stereo images has been implemented in Matlab. Second, to build a parameterized model from the 3D data for the purpose of 3D face motion capture, an implementation of the proposed *sparse 2D-to-3D AAM* method was realized as a modification of the open source AAM-API project [21] written in C++. Finally, to visualize and test the output of the motion capture, an animation program was written in C++ with the use of OpenGL. To communicate between these modules a file format containing 3D shape information has been defined. The file format is named *.asf3d* and is based on the *.asf* format of the AAM-API.

In Figure 6.1 the the overall pipeline of the motion capture framework can be appreciated. Entities in the diagram that are marked by red are either part of the framework software, or produced by it. The four main entities are a stereo annotation tool, a statistical model building program, a 2D-to-3D AAM geometry synthesizer, and finally, a 3D face animation front end.

The main entities receive inputs external to the framework (image, video, and face puppet data) as well as inputs internal to the framework (i.e. files produced by components of the framework itself). To communicate between the framework entities two file formats are used. Before going into the details of the implementation of the main entities, it is appropriate to describe the how they are interfaced by the two file formats.

However, the first chapter of this part concerns the stereo face image database built to provide the data needed for a 2D-to-3D AAM model construction.

Chapter 7

The Stereo Face Image database

7.1 Overview

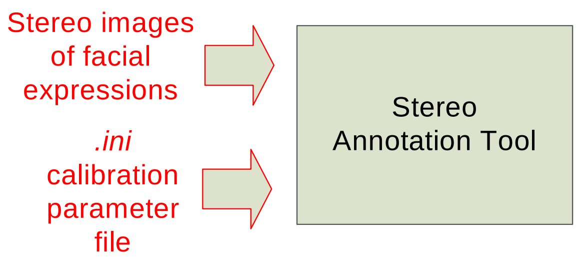

Stereo images of facial expressions
.ini calibration parameter file
Stereo Annotation Tool

Figure 7.1: Context diagram of face database

In the computer vision research community a variety of face databases has been built over the years. Their purpose has been to support research in a range of areas such as face recognition, pose estimation, face synthesis and emotion analysis. From these databases statistical models can be built and tested, within certain constrained scenarios.

To build a statistical model capable of describing 3D facial expressions one needs a set of training data containing the variation that needs to be synthesized. Currently a few 3D datasets exists, but none of them provide sufficient variation in the facial expressions to be usable in this project. It has therefore been decided to enrich the world of computer vision with yet a face database, be it a pretty small one of the kind.

7.2 Description of the Image Data

The stereo images used for this project are captured using a 640 x 480 pixel stereo camera of type STH-DCSG-STOC-C from *Videre Design*. The camera has no zooming capability but focus can be adjusted manually for each of the two camera lenses. A stereo image consists of a *pair* of images, conveniently called a *stereo image pair*, see Figure 7.2. The two images are refered to as the *left view* and right view image. For many purposes one of the images is defined as the reference image, conventionally this is the left view image.

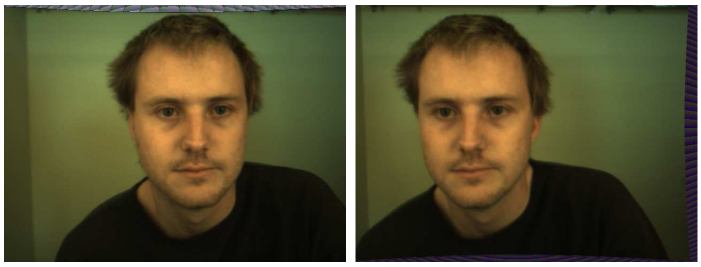


Figure 7.2: An example pair of stereo images from the database.

The stereo images have been captured over several sessions where no effort was done in optimizing lighting conditions. Three human face models were used, all male and of age 20-30 years. The face models were positioned with their face fronting the stereo camera at a distance of 40-50 cm. They were asked to perform a series of six facial configurations, expressing *joy*, *sadness*, *fear*, *anger*, *surprise*, *disgust*, plus a seventh *neutral* facial expression. The six former expressions are defined in psychological research as the six *universal facial expressions* [9], i.e. they are consistent and recognizable across cultures. They were chosen for this database because of this property.

As the most expressive parts of the human face are eyes, mouth and eyebrows, it was important to have substantial variation in these face regions over the series of expressions. The face models were therefore asked to interpret the six emotions in *exaggerated* facial expressions. The face models should ideally be actors, trained in conveying emotions through their face. Unfortunately, resources for this project allowed only models recruited from nearby laboratories.

With *Videre Design*'s stereo cameras comes a software tool called *Small Vision System* (*SVS*). *SVS* provides GUI interfaces for calibration and image capture. It performs rectification and outputs an *.ini* file with the calibration parameters, including projection and rectification matrices. The *.ini* files are included in the face database.

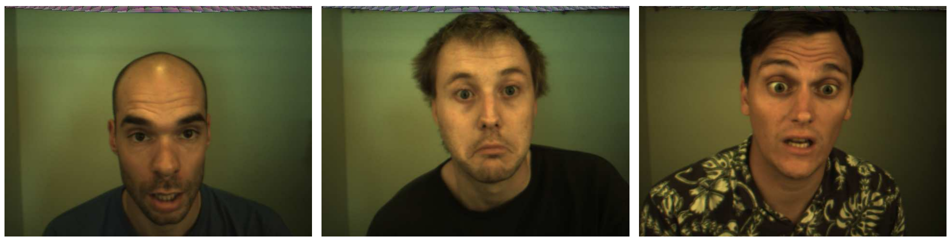


Figure 7.3: Three interpretations of *surprise*.

Chapter 8

Interface Formats

8.1 Overview

This chapter introduces the two file formats, *.amf* and *.asf3d*, that are used to interface the main entities of the implemented motion capture framework.

8.2 The .amf file format

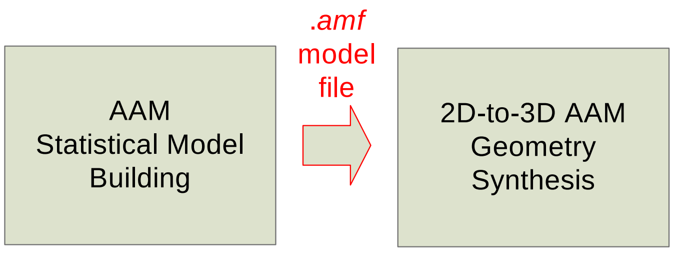


Figure 8.1: Context diagram of *.amf* file.

The *.amf* file is a binary file that carries the information produced by the modeling program that is needed to synthesize a model instances during tracking. This information includes the model eigenspace, parameterized mean shape and texture, parameter prediction matrices and more.

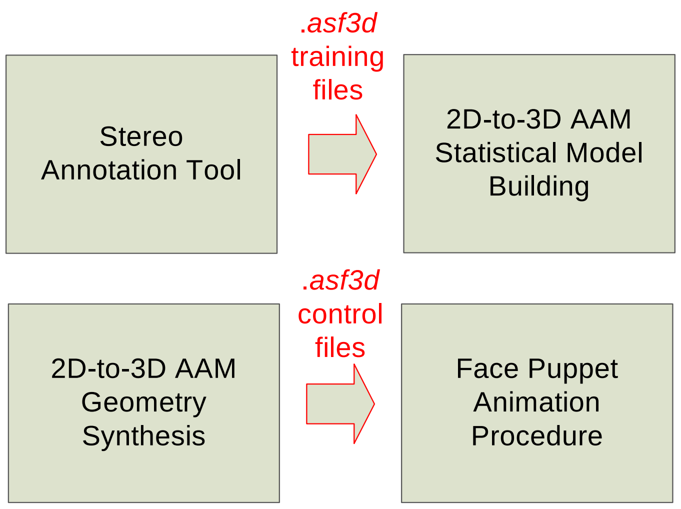
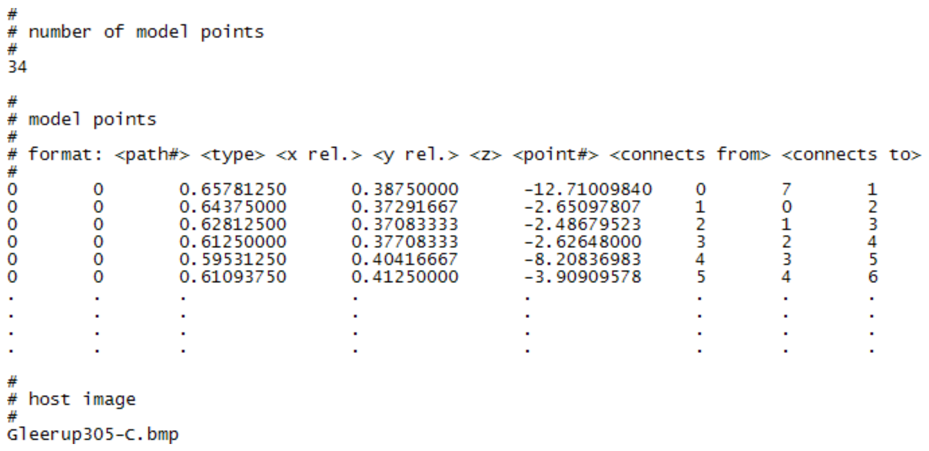

.asf3d training files
Stereo Annotation Tool
2D-to-3D AAM Statistical Model Building
.asf3d control files
2D-to-3D AAM Geometry Synthesis
Face Puppet Animation Procedure

Figure 8.2: Context diagrams of *.asf3d* training and control files.

8.3 The .asf3d file format

The *.asf3d* file is a text file defining the interface between the annotation tool and the AAM model building entities, and between the AAM tracking and the animation entities. In doing so the *.asf3d* file constitutes a cornerstone of the framework, and the entities are designed to comply with the data format defined by the file.

The format is a 3D extension of the 2D *.asf* format defined in the *AAM-API* [21]. The normalized x- and y-coordinates and the un-normalized z-coordinates of a shape's feature points are written to this file together with information of the points' connectivity and the image file association. A 1-to-1 relationship exits between *.asf3d* files and images.



```
#
# number of model points
#
34

#
# model points
#
# format: <path#> <type> <x rel.> <y rel.> <z> <point#> <connects from> <connects to>
#
0       0       0.65781250      0.38750000      -12.71009840    0       7       1
0       0       0.64375000      0.37291667      -2.65097807     1       0       2
0       0       0.62812500      0.37083333      -2.48679523     2       1       3
0       0       0.61250000      0.37708333      -2.62648000     3       2       4
0       0       0.59531250      0.40416667      -8.20836983     4       3       5
0       0       0.61093750      0.41250000      -3.90909578     5       4       6
.       .       .               .               .               .       .       .
.       .       .               .               .               .       .       .
.       .       .               .               .               .       .       .

#
# host image
#
Gleerup305-C.bmp
```

Figure 8.3: Example of .asf3d file.

As seen in Figure 6.1 the file format carries two types of information: training shape data from the annotation tool to the AAM model building component, and control shape data from the AAM video tracking to the animation program. In the former case, one *.asf3d* file is associated with one of the training images. In the latter case, one *.asf3d* file is generated per frame in the video of the human face actor.

Chapter 9

The Stereo Annotation Tool

9.1 Overview

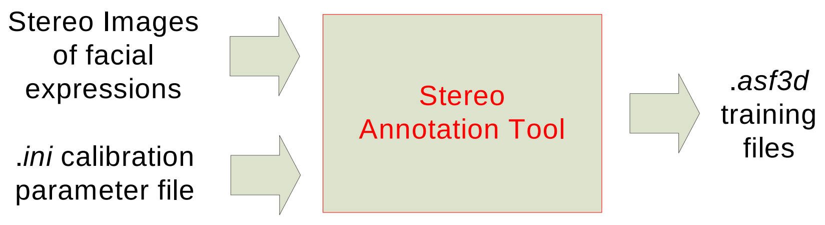

Stereo Images of facial expressions
.ini calibration parameter file
Stereo Annotation Tool
.asf3d training files

Figure 9.1: Context diagram of 3D shape data acquisition

An annotation tool has been developed in Matlab. The tool consists of a GUI interface with access to a directory of stereo image pairs and their calibration parameters. A point-and-click functionality allows the user to annotate feature points of choice in one of the stereo images. Subsequently, the corresponding points will automatically be found in the other stereo image and the 3D points calculated by reprojection and output to an *.asf3d* file.

In this project the objective is to build a sparse model that allows control of a virtual puppet face in order to convey emotions in a more or less abstract fashion. The assumption is, that the facial expressions embedding the emotions can be encoded, transmitted and decoded using relatively few data points. As the eye, mouth, and eyebrows are the most expressive parts of the buman face, these are the regions that must be modelled by this system.

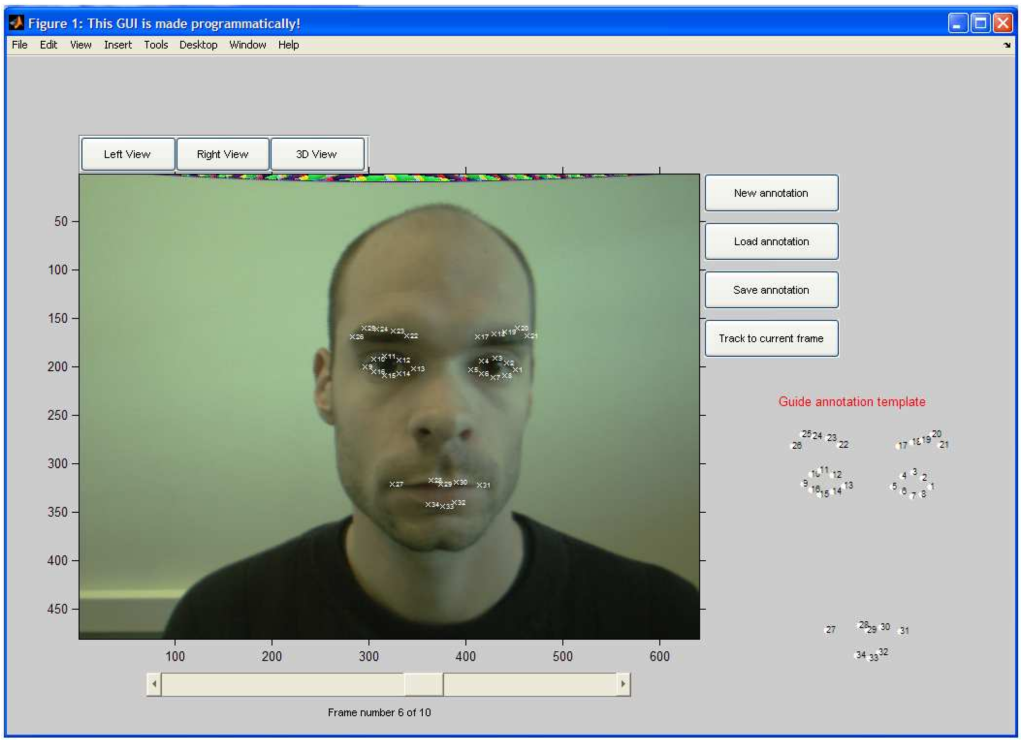


Figure 1: This GUI is made programmatically!
File Edit View Insert Tools Desktop Window Help
Left View
Right View
3D View
New annotation
Load annotation
Save annotation
Track to current frame
Guide annotation template
Frame number 6 of 10

Figure 9.2: Screenshot of annotation tool GUI.

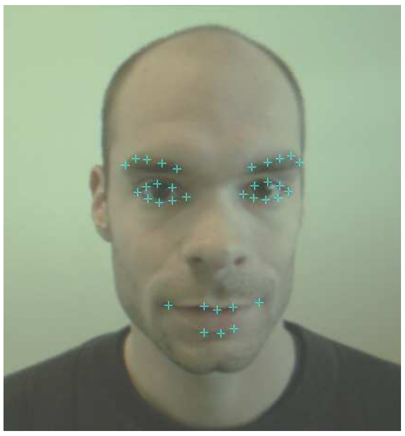


Figure 9.3: Image with feature points.

9.2 Feature point correspondance

Feature points are, as described in Section 3.4, points of high curvature. This property is helpful in stereo vision when one wishes to find conjugate points in a stereo image pair.

One approach to finding conjugate points is to cut out a template window around the feature point in the reference image (by convention chosen as the left image) and simply move the window along the points' epipolar line in the right image, looking for a best match by some similarity measure. Remember that, for horizontally aligned rectified images, the epipolar line of a point is the row of the point in the other image.

Another approach, that was found to work better than the above, was to perform the matching in the gradient domain instead of the grayscale intensity domain. One graycal image produces two gradient images, one for each image dimension. The best match is then the one that minimizes the dissimilarity, ϵ, over the compared image region, over both gradient images:

$$\epsilon = \sum_{W}(\mathbf{T}_{gx} - \mathbf{I}_{gx}) + \sum_{W}(\mathbf{T}_{gy} - \mathbf{I}_{gy}) \tag{9.1}$$

Here W is the template window. Matrices $\mathbf{T}_{gx}$, $\mathbf{T}_{gy}$ are the gradient images of the template cut out from the left image. The matrices $\mathbf{I}_{gx}$ and $\mathbf{I}_{gy}$ are gradient images of the entire right image where the conjugate point is searched.

An important question in the case of template matching is how large the template window should be. Using a larger window yields a more expensive matching procedure but at the same time it seems intuitive that the robustness of the matching increases with the window size. In Figure 9.4 it can be seen that this assumption is in general *not* true.

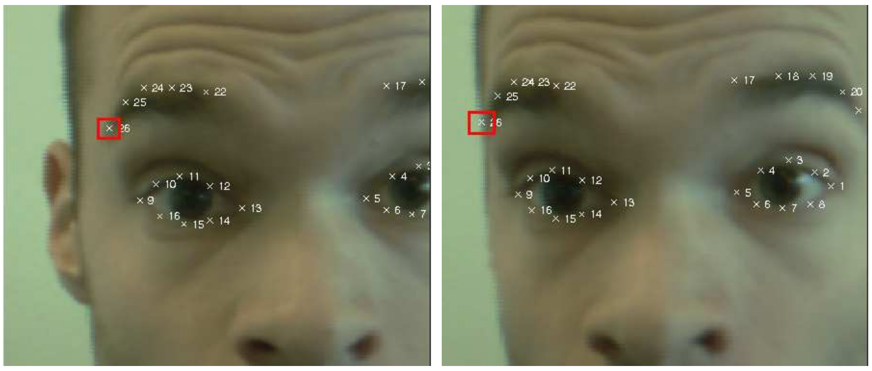


Figure 9.4: Two perspectives of the same face image. The area enclosed by the red square in the left image is used as a template to find the corresponding point in the right image. The problem is obvious.

The two images of the stereo pair show the face from two different perspectives, not very different, but still different. Therefore the distances between conjugate scene features can vary considerably between the two images. This is exemplified in Figure 9.4, where the template window is cut out around a landmark in the left reference image. In the right image, at the conjugate point, the template partially covers the image background. Therefore the landmark in the right image is not likely to be found.

Some trial-and-error experimentation with the template size, under considerations such as image resolution and camera-to-object distance, is therefore necessary. For the particular image material used in this project a template size of 21 x 21 pixels was found to give the best results.

However, as the matching is by no means perfect, a functionality has been added to the annotation tool that allows for manual point-and-drag correction of misplaced correspondance points.

After having obtained conjugate points in both stereo images the points' 3D position is calculated by reprojection (see Section 3.3.5). The annotation tool GUI visualizes these points in its *3D view*.

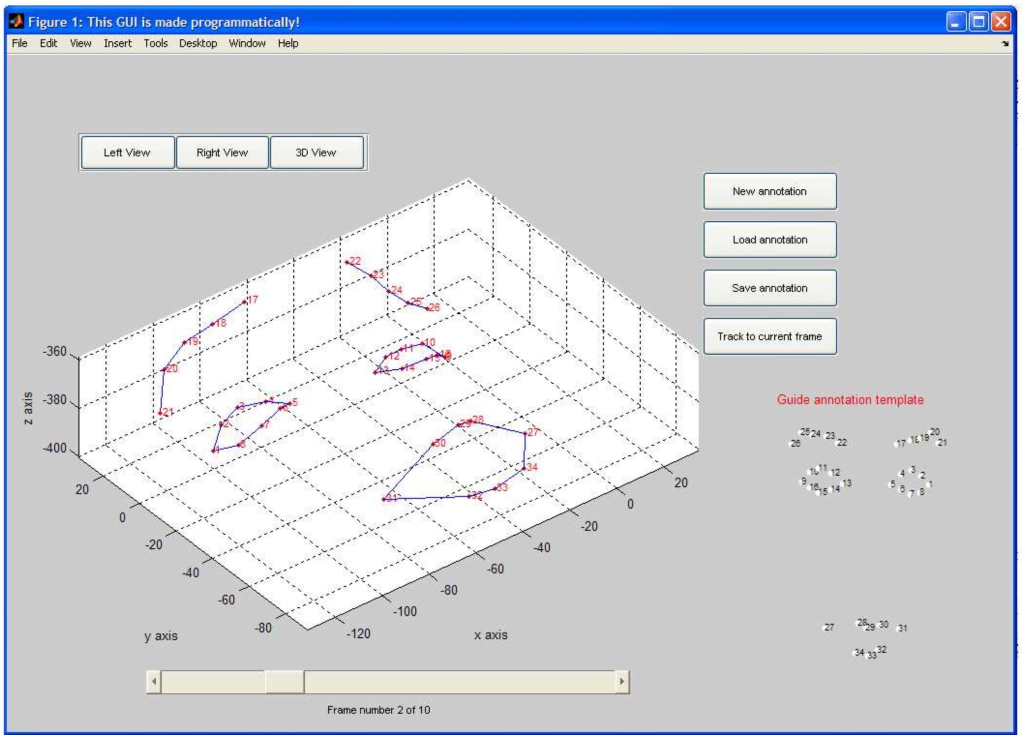


Figure 9.5: Screenshot of 3D view of annotation tool GUI.

Finally, the 3D points are written to an *.asf3f* file.

9.3 Assumption of Preserved Aspect Ratio after Projection

At this point an assumption is introduced, that will simplify the implementation of the 2D-to-3D AAM. The assumption concerns the training images, which is the left view subset of the stereo images on which annotations are performed.

The assumption is that the faces in the training images preserve their aspect ratios in the xy-plane after projection. That is in other words, that the face images are perfectly frontal and the optical axis passes the face's center of gravity.

The assumption is made to be able to express the 2D shape coordinates as a subset of the 3D shape coordinates, namely as the x- and y-components. The reason why we want the 2D coordinates readily available to the AAM model builder is that they define which part of the training image to use for the texture model.

The steps of obtaining a 3D shape that contains the annotated 2D shape in its xy-components are given below:

- Append an extra dimension of zeros to the annotated 2D shape of the left view training image, making it a flat *pseudo* 3D shape.
- align the *real* 3D shape to the pseudo 3D shape using Procrustes alignment.
- Copy the z-component of the aligned real 3D shape into the zero-column of the pseudo 3D shape.
- Normalize x- and y-components of the updated 3D shape according to left image width and height.
- Leave new z-component of new 3D shape un-normalized.

Note that the z-component is not immediately normalizable since its range is not defined.

However, the face images are *not* perfectly frontal and the aspect ratios are *not* preserved after projection. This can be seen in Figure 9.6, where the blue shape is the 2D shape found by annotating the left view of a pair of stereo images, while the red shape is the xy-component of the 3D shape found from the same pair of stereo images. The shapes have been aligned.

The question therefore remains, if the assumption of preserved aspect ratio in the projected faces is valid by some measure. Since face proportions are likely to vary substantially over a population, the assumption could be tested by calculating the likelihoods of the aspect ratios of the 2D shapes within a distribution of aspect ratios

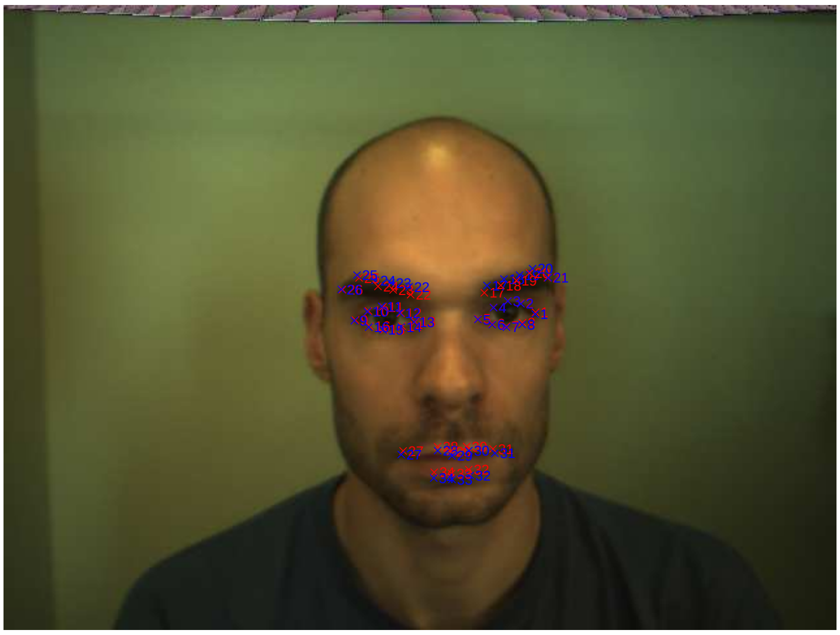


Figure 9.6: The figure shows an annotated left view stereo image. The annotated point are shown in blue color. The red color points are those of the 3D shape which was reprojected from the blue points (and their corresponding points in the right view image). The 3D shape has in the image been aligned to the 2D shape, so it can be seen how the aspect ratios of the two shapes differ slightly.

of 3D shapes. This test has been performed and the assumption has been validated as described in section 12.4.

Chapter 10

Sparse 2D-to-3D Active Appearance Models

10.1 Overview

The goal of the active appearance model in the context of this project is to synthesize sparse 3D geometry from 2D images. The approach described in this section develops the traditional 2D AAM method to deal with exactly this problem. By combining a statistical shape model from 3D data, and a corresponding statistical texture model from 2D textures, the appearance model embeds the relationship between 2D face images and their 3D geometry.

As a platform for exploring active appearance models, Denmarks Technical University (DTU) provides a free and open source 2D AAM implementation called the *AAM-API* [21]. The AAM-API is a very well-structured and well-documented object oriented application programming interface (API), that implements the AAM method for 2D shapes in more than 15,000 lines of C++ code. However, using it to model and synthesize 3D shapes is not an option. Therefore, it has constituted a major part of this thesis work to add 3D shape modelling functionality to the *AAM-API*.

The rest of this section describes how, based on the algorithmic implementations of the *AAM-API*, the *sparse 2D-to-3D AAM* method has been implemented.

10.2 Statistical Model Building

This subection deals with the construction of a sparse 2D-to-3D appearance model. The process takes a set of training images and their associated 3D shapes as input, and returns a parameterized statistical model, capable of synthesizing 3D geometry and 2D texture.

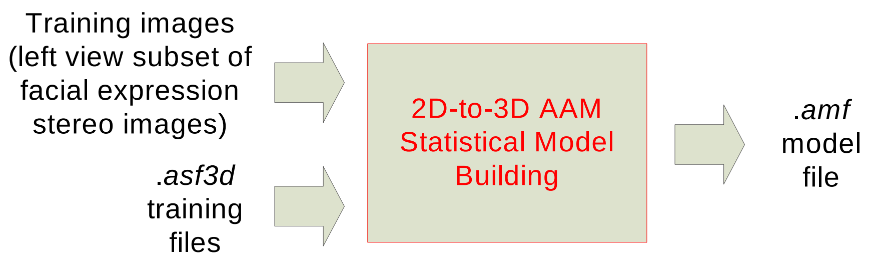


Figure 10.1: Context diagram of the 2D-to-3D AAM model building process.

A shape in 3D is mathetically described by simply adding a dimension to the 2D description.

$$\mathbf{x}_i = [x_1, x_2, \cdots, x_N, y_1, y_2, \cdots, y_N, , z_1, z_2, \cdots, z_N] \tag{10.1}$$

After aligning the training shapes by applying Procrustes alignment (Section 4.3) to 3D, a statistical model describing the shapes' 3D variations is built by principal component analysis (PCA).

The 2D-to-3D AAM model building procedure is very simular to the process described for 2D AAM in Section 4.4. However, a few subtle differences must be noted.

As explained in Section 9.3 the 3D training shapes arrive in a format where the x- and y-coordinates are coinciding with the landmark annotations of the training images, while the z-coordinates are those of the original 3D shape after alignment to the landmarks. The reason the effort was made to reach this format, is that projection of the 3D shape onto the face image is now simply a matter of removing the z-component from the shape.

This property comes in handy when retrieving the textures from the training images. The texture that is sampled from from a training image is the region covered by the 2D projection of the associated 3D training shape. A statistical texture model can then built from the sampled 2D textures exactly as described in Section 4.4.2.

Now, by combining the 3D shape model and the 2D texture model a compact representation, embedding the statistical relationsship between the texture of a mono-view face image and 3D face geometry, is obtained.

10.3 Parameter Prediction Optimization

As described in Section 4.5.1 a linear model is trained to predict changes in model parameters that will optimize the fitting of the model to an input image.

Since face rotations that occlude landmarks are illegal under AAM, the offline parameter prediction optimization trains the pose predictions model from translation, scaling and rotation in the xy-plane only. Actually, rotations in the xz- and yz-planes can be modelled to the degree that all feature points were visible in the training data. But since this thesis deals with facial expressions and not pose, this opporunity has not been considered.

Thus, the pose parameters are trained for displacements in the xy-plane only. The model parameters however, embedding the 3D shape and 2D texture models, are trained from displacements according to the parameters' variance in the model space, which is the Eigenspace constructed by the PCA and cannot be understood in terms of neither the 2D space of the textures nor the 3D space of the shapes.

10.4 Texture-to-Geometry Synthesis

The process of synthesizing 3D geometry from face images takes the parameter model as well as the optimization model described above, as its input. This information is read from the *.amf* file. For each frame in an input video, the the 2D-to-3D AAM optimally fits the model by adjusting the model parameters. In this section, an algorithmic description explaining this process is given.

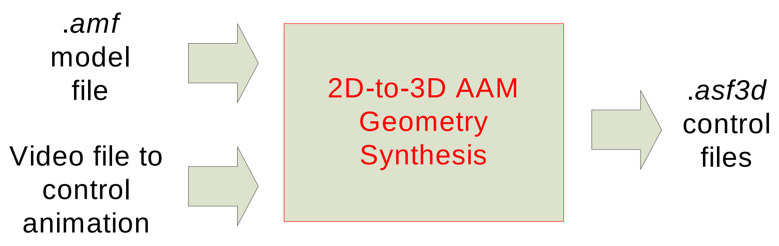

.amf model file
Video file to control animation
2D-to-3D AAM Geometry Synthesis
.asf3d control files

Figure 10.2: Context diagram of 3D shape synthesis process

The goal of the 2D-to-3D AAM optimization is, like that of the 2D AAM, to minimize the error between a texture sample from the input image and the texture instance generated from the appearance model. The texture instance is generated from the model by Eq. (4.15), together with a 3D shape instance, by Eq. (4.14), as shown in Section 4.4.3.

After projecting the 3D shape onto a 2D input frame, the frame is sampled in the region covered by the projection shape. This sample is compared to the texture instance generated from the model parameters. From the difference between the two textures, the model paramters can be adjusted to minimize this difference, by the help of the parameter prediction model.

The process was implemented by the following algorithm in the 2D-to-3D AAM:

1. Initialize the model within the reach of the parameter model's prediction range.
2. Generate the normalized texture and normalized 3D shape instances, $\mathbf{g}_m$ and $\mathbf{x}_{3D_m}$, from current model parameters
3. Project $\mathbf{x}_{3D_m}$ into $\mathbf{x}_{2D_m}$ on the xy-plane
4. Sample the image region covered by $\mathbf{x}_{2D_m}$ into $\mathbf{g}_{image}$
5. Normalize $\mathbf{g}_{image}$ into $\mathbf{g}_i$
6. Evaluate the error vector, $\delta\mathbf{g}_0 = \mathbf{g}_i - \mathbf{g}_m$
7. Evaluate the error, $E_0 = |\delta\mathbf{g}_0|$
8. Predict the displacements in pose parameters, $\delta\mathbf{t} = \mathbf{R}_t\delta\mathbf{g}_0$.
9. Predict the displacements in model parameters, $\delta\mathbf{c} = \mathbf{R}_c\delta\mathbf{g}_0$.
10. Set $i = 1$
11. Update model parameters, $\mathbf{c} = \mathbf{c} - k_i\delta\mathbf{c}$, where k is a damping/attenuation factor.
12. Transform the shape to invert the $\delta\mathbf{t}$ transformation
13. Repeat steps 3-6 to form a new error E_i
14. If $E_i > E_0$ set $i = i + 1$ and go to step 10
15. Save the resulting error $E = E_i$
16. If no improvement in E from the last iteration, declare convergence. Else go to step 3.

By this, the description of the sparse 2D-to-3D AAM method is concluded. In the emotion conveyance test in Section 12.3 it can be seen how well this method, in conjunction with the animation program presented in next chapter, lives up to its intentions of modeling facial expressions in 3D.

Chapter 11

The 3D Facial Animation Front End

11.1 Overview

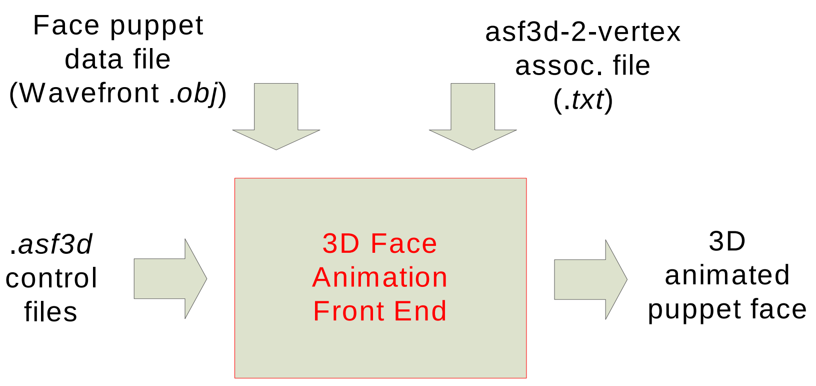

Face puppet data file (Wavefront *.obj*)
asf3d-2-vertex assoc. file (*.txt*)
.asf3d control files
3D Face Animation Front End
3D animated puppet face

Figure 11.1: Context diagram of the animation front end

This section describes how the motion capture framework *front end*, where 3D shapes, synthesized by the sparse 2D-to-3D AAM geometry synthesis, are employed in *digital puppetry*. The term *digital puppetry* is an analogy to real world puppetry, where a puppet is controlled by a human via a set of strings attached to expressive parts of the puppet. In this section the vocabulary of puppetry is adapted to help the reader visualize the concepts described.

- *Puppet* is used here to describe the 3D graphic face model to be animated by the captured data.

- *String* denotes an association between corresponding human and puppet face landmarks. The *set of strings* is thus the set of associations between *.asf3d* points and corresponding vertices of the puppet.
- *Controls* refer to the *.asf3d* files that specifies the 3D landmark positions of the human face over time.

The animation procedure takes as input a sequence of *.asf3d* control files, each specifying the 3D positions of the facial landmarks in one frame of a face video. Thus, from the control files the translations of the landmarks can be calculated on a per frame basis. The translation information retrieved is used to control the translation of the puppet vertices over time and thus animate the puppet. This process is explained in detail below.

11.2 The Puppet Face Model

The *test face puppet* used in this project is a polygon surface consisting of 6344 triangles. The test puppet is shown in Figure 11.2. It was determined by Parke in [17] that approximating the surface of a face with a polygonal skin containing approximately 250 polygons defined by about 400 vertices, is sufficient to achieve a realistic face.

Since efficiency is not in the scope of this project and that the animation methods applied are scalable to both higher and lower mesh resolution, no effort has been put into reducing the polygon count. The test puppet face was found and downloaded for free as a *Wavefront .obj* file from http://www.turbosquid.com.

The only modification to the original downloaded object has been the deletion of polygons between the upper and lower lip of the face, to enable opening of the mouth.

In face puppetry where the puppet is controlled by virtual strings consisting of associations between the human face landmarks and puppet face landmarks, it is natural to use a polygonal geometry. In polygon geometry direct associations can be established between human face landmarks and corresponding puppet vertices. This is opposed to e.g. parametric surfaces such as B-splines or NURBS [15].

As outlined in section Section 5.2, the topology of a polygonal face model should comply with certain rules depending on its usage.

The topology of the test puppet can be classified as an *arbitrary* mesh. However, the term *arbitrary* might seem somewhat inappropriate in this case: if the triangles are paired into *quads*, it is be seen how the polygonal structure is both symmetric

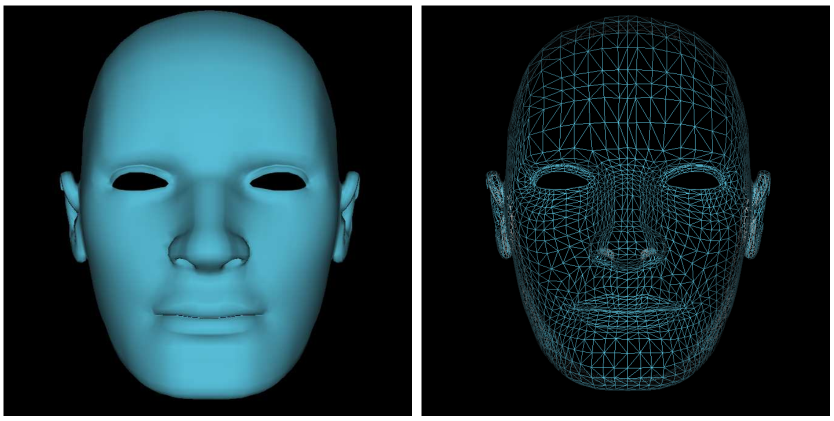


Figure 11.2: Test puppet face, with smooth shading and as wireframe model.

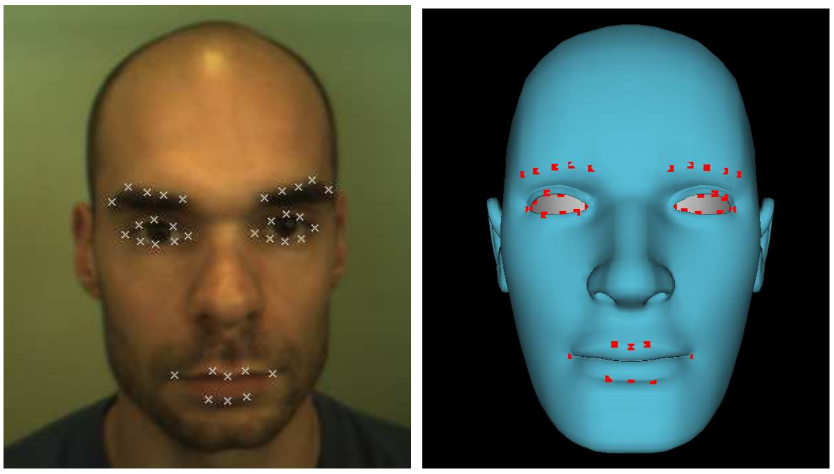


Figure 11.3: *Left*: Face image with annotated landmark points. *Right*: Face puppet with highlighted control points.

and contains *edge loops*. In Figure 5.1 the test puppet is shown composed of quads instead of triangles. In edge loops, polygons are organized into structures looping around face cavaties such as mouth and eyes. This structure supports expansion and subtration of the cavaties and thereby supports animation. It is also seen in Figure 5.1 how the areas of high curvature around e.g. the eyes are described by a higher number of polygons than low curvature areas as cheeks and forehead, to enable more detail.

The test puppet model does not embed any form of physically based muscle or skin model constraining the translation of its vertices during animation. In the following the animation scheme developed is described.

11.3 Attaching Strings to the Puppet

An animation scheme has been implemented in C++ using an OpenGL graphics library. It takes a set of *.asf3d* files, each file associated with a frame in the face recording used for motion capture, as input and produces an animation. The basic idea is to retrieve the point translation information from the set of *.asf3d* files and use it to translate the vertices of the puppet.

First, speaking in puppetry terms, *strings* must be attached to the puppet. This is done manually by annotating landmark points in the neutral expression puppet face corresponding to those originally annotated in the human face training images.

Marking a point in 3D space is not as trivial as in the 2D case. When a point is clicked in the viewport of a 3D scene, the point actually represents a *ray* stretching infinitely behind it. However, the vector describing this ray can be found, and the vertex nearest to the ray can be saved as the annotated landmark vertex. In puppetry terms, a string is now attached to this vertex. Note here that the ray will pass several polygons of the puppet. E.g. it will hit the marked polygon in the face, and as the ray continues through the model it will hit a polygon in the back of the head. In this case one has to choose the vertex of the nearest polygon to be the landmark point.

11.4 Controlling the Puppet

After setting up the set of strings, i.e. after establishing an association between *.asf3d* control points and puppet vertices, a weighting scheme is applied, see Figure 11.4. The weights determine how all the vertices that are not *landmark* vertices shall move relative to the landmark vertices. Again, to put it in puppet terminology, the weights determine the influence of each string on puppet surface points where no strings are attached.

For each vertex, v_i, in the puppet model, the four nearest landmark vertices, l_{ik}, are found, where $k \in \{1, 2, 3, 4\}$ indexes the four landmarks. Based on the four landmarks' euclidean distances from the vertex, $d(v_i, l_{ik})$, a normalized weight w_{ik} is associated with the each of four landmarks. The weights are found according to the *inverse distance weighting* scheme [19].

$$w_{ik} = \begin{cases} \frac{1}{d(v_i,l_{ik})} \cdot \sum_k d(v_i, l_{ik})^p, & \text{if } v_i \neq l_{ik} \\ 1, & \text{if } v_i = l_{ik} \end{cases} \tag{11.1}$$

The weights are to be applied during animation, to interpolate translation of vertices that are not landmark vertices.

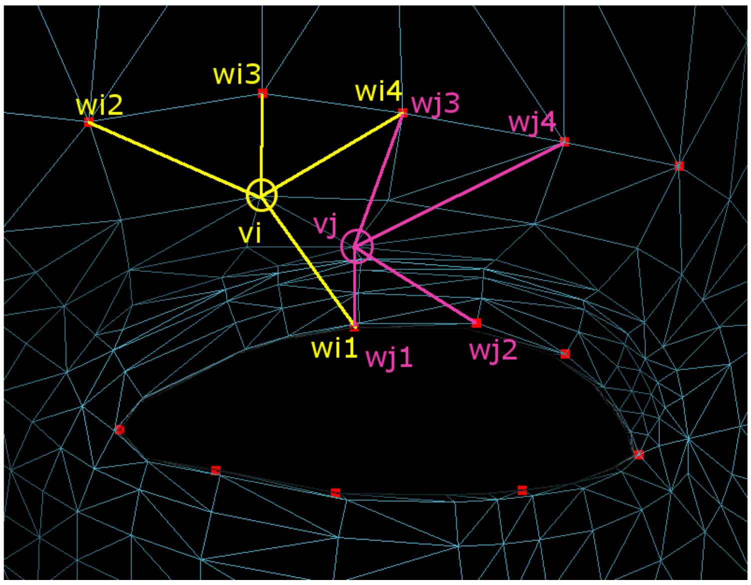


Figure 11.4: Weighting scheme. Each vertex is associated with its four closest landmarks. Each of the four landmarks is assigned a weight inverse proportional to its distance from the vertex.

Next, a neutral expression 3D shape, S_{base}, is chosen from the set of *.asf3d* shape files. This base shape is aligned in scale, translation and rotation with L_{base}, which is the shape formed by the neutral landmark vertices of the puppet, see Figure 11.5. The aligned shape is denoted the *base control shape*, C_{base}.

In Figure 11.5, the white control shape is aligned to the shape formed by the red test puppet landmarks (to avoid occlusion by the puppet the control shape is translated a bit in the image, along the z-axis, towards the camera). It is seen from the figure is how the aspect ratios of the two shapes differ.

As a last initialization step before the runtime animation can be started, the height, width and depth ratios, describing the size of L_{base} relative to C_{base}, are found. The ratios are denoted r_{width}, r_{height} and r_{depth}, and they ensure a proportionally correct vertex translations during animation.

Now everything is set up for the runtime animation algorithm, which runs over all *.asf3d* input files. As each file contains the shape information corresponding to the facial expression of one frame in a face video, these files should be processed at the frame rate of the video.

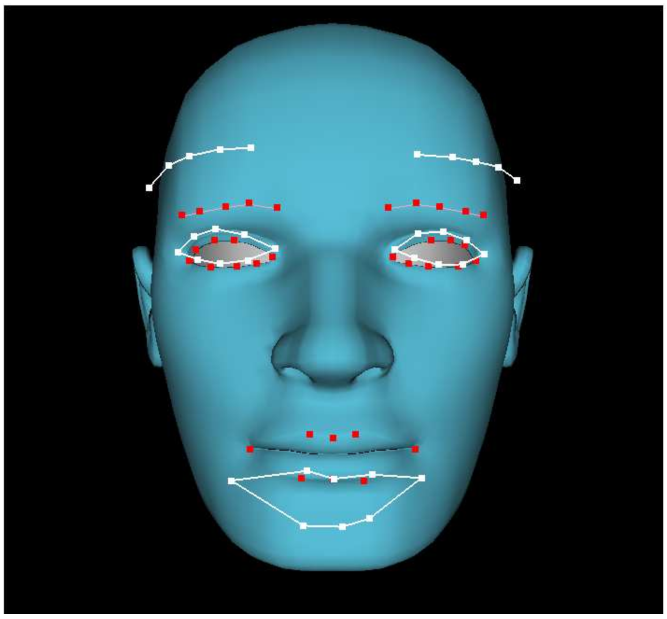


Figure 11.5: Puppet controls. The test puppet is shown in neutral expression with its landmark points, L_{base}, highlighted in red. The white shape is the base control shape, L_{base}, also in a neutral state. (Imagine a set of invisible strings going from the white to the red points).

11.5 The Runtime Animation Algorithm

The first step of the algorithm is to align the shape contained in the currently processed *.asf3d* file to the neutral expression landmarks of the puppet. This way, shape size variations are neutralized. Such variations would originate from the actor in the face video moving his/her head towards or away from the camera.

Next, the translations for the current shape are found relative to an aligned reference shape, above called C_{base}. The translations of the input shape are then used to calculate the translations of the face puppet landmark vertices, also relative to a reference shape, L_{base}. Knowing the translations of the landmark vertices, the translations of the remaining puppet vertices can be calculated by interpolation, applying the weights found earlier.

In pseudo code the algorithm looks like this:

- For all input 3D shapes from the face recording, do:
  - Align the current 3D shape, S, with the base puppet landmark shape, L_{base}, into the current control shape, C
  - For each point, c_j, in C, do:

        - Find the translation, $t(c_j)$, which is the euclidean distance from c_j to the j^{th} point in the base control shape, C_{base}.
        - Multiply the x-, y- and z-components of t_{c_j} with the size relation ratios, r_{width}, r_{height} and r_{depth}, respectively.
        - Insert t_{c_j} into the j^{th} column of the vector $\mathbf{t}_c$.
    - For each vertex, v_i, in the puppet head, do:
        - Retrieve v_i's four associated landmark vertices, $\mathbf{l}_i$, with weights $\mathbf{w}_i$.
        - Retrieve the translations of the four control shape points, $t_c(\mathbf{l}_i)$, from $\mathbf{t}_c$. Insert $t_c(\mathbf{l}_i)$ into vector $\hat{\mathbf{t}}_\mathbf{i}$
        - Interpolate translation of v_i: $t_{v_i} = \sum_{k=1}^{4}(\hat{\mathbf{t}}_\mathbf{i}(k) \cdot \mathbf{w}_i(k))$
        - Translate v_i according to t_{v_i}
- Wait, to align animation pace to frame rate of face recording

11.6 Special Case: Mouth Region

In the animation scheme described above the mouth region poses a problem since vertices on the upper lip are because of short distances associated with landmarks on the lower lip and vice versa. To fix this problem upper and lower lip vertices must be constrained to be associated with upper and lower lip landmarks, respectively. This is done by adding further initialization functionallity to be performed before the assignment of landmark points and weights to the vertices.

The first method is called *point-in-polygon* [14]. It is a method from graph theory that cleverly finds out if a vertex' xy-component is within the 2D polygon created by connecting the xy-components of the mouth landmarks.

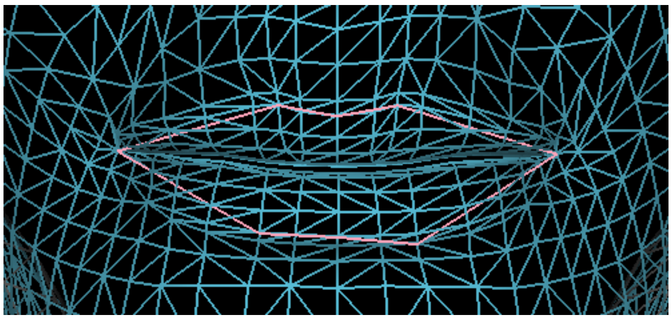


Figure 11.6: Mouth polygons and vertices. Pink lines are drawn between mouth landmarks.

Secondly, to determine if a vertex, that is classified to be within the mouth region, is within the upper or lower lip, a graph distance measuring function is applied. The method is *breadth first search* or simply *BFS* [12].

In this case, BFS is used to measure the shortest distance, along the puppet mesh edges, from a mouth vertex to each of the mouth landmarks. Since no edges connect the upper and lower lips in between the mouth corners, the distance along edges from, say, a vertex on the upper lip to a landmark on the lower lip is generally longer than the distance to an upper lip landmark. By comparing the sum of the vertex' distance to the upper lip landmarks with the sum of the vertex' distances to the lower lip landmarks, the vertex "lip association" can be determined.

The *nearest landmark* association and weight assignment of an upper lip vertex, can thus be limited to include upper lip landmarks and mouth corners.

11.7 Summary of 3D Facial Animation

The animation scheme described in this section has been implemented and results in smooth and recognizable facial expressions. Though the method is highly customized to the problem at hand, several parts, e.g. the weighting and vertex translation schemes are generally applicable in animation of polygonal face models.

Chapter 12

Tests

12.1 Overview

This chapter describes the tests performed in relation to this thesis. Two performance tests have been conducted. Each of them quantify to which degree two of the thesis objectives stated in Section 1.1 are fulfilled. A third test investigates if the assumption proposed in Section 9.3 is valid. The concerns of the tests are summarized here:

- Precision of the annotation tool
- Overall emotion conveyance quality
- The assumption of preserved aspect ratio

In the following the purpose, method, results and conclusion of each test are given.

12.2 Annotation Tool Precision

12.2.1 Purpose of Test

This test was performed to measure how well the 3D data obtained from the annotation tool reflects the annotated objects geometry.

12.2.2 Test Method

The precision of the annotation tool was measured by the following setup:

A stereo image was taken of a chessboard pattern wrapped around a right angle corner. The distance between the camera and the corner was approximately 40 cm at the top of the corner, which corresponds to the distance from camera to the human models in the facial expression footage for the face database. The chessboard pattern consisted of 7 x 9 square tiles each of 27 x 27 mm.

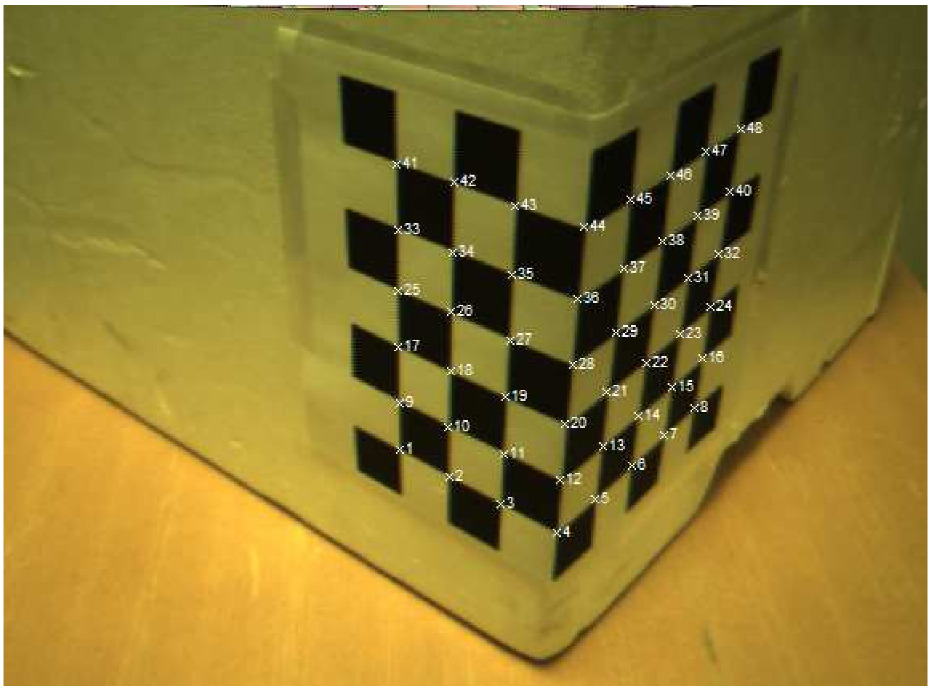


Figure 12.1: Chessboard wrapped around a right angle. Used for testing precision of annotation tool.

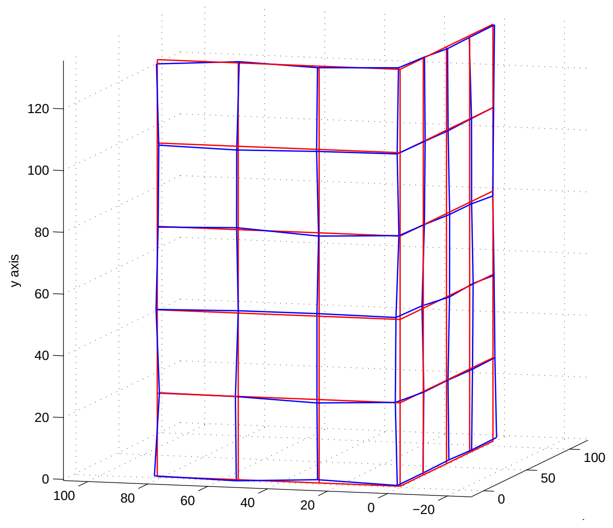
The chessboard corner was used as a ground truth object of the test. In addition, a virtual model of the chessboard corner was generated in Matlab.

The test was performed by 10 instances of manual annotations of 48 points on the chessboard corner, see Figure 12.1. Using the *Procrustes* alignment method, the datapoint set from the annotation was aligned to the virtual model w.r.t. translation and rotation, see Figure 12.2.

12.2.3 Test Results

The euclidean distances between the virtual shape and the 10 annotated shapes were measured in mm for each of the 48 points per shape. The sum of errors over each of the 10 annotated shapes were (in unit mm), the results are seen in Table 12.1

The average error of all 48 points in the shape over the 10 annotation instances was 58.66 mm. Per point that corresponds to an average error of 1.22 mm.



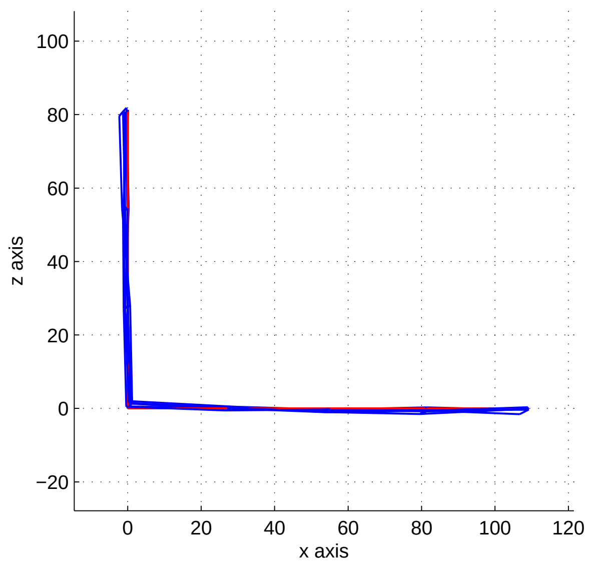


Figure 12.2: Matlab visualization of *chessboard corner* grid constructed from annotation (blue) compared to the ideal chessboard corner grid (red). Buttom image is top view.

12.2.4 Conclusion

The test shows that a point is on average 1.22 mm off its intended position. The source of the error is most likely imprecision on the side of the human doing the annotation.

In any case, for this project such an error is acceptable, taking the size of a human face into account. However, it should be noted, that the same level of accuracy

1	2	3	4	5	6	7	8	9	10
60.85	60.57	57.18	57.42	60.49	57.37	58.07	57.86	57.80	59.00

Table 12.1: Table of average sums of errors of all annotated points shown for the 10 test shapes, unit is mm.

cannot be expected when annotating a human face, due to much weaker landmark points, than those of a chessboard.

12.3 Overall Emotion Conveyance Quality

12.3.1 Purpose of Test

The purpose of this test is to document how well the motion capture framework developed in this project conveys emotions.

12.3.2 Test Method

Four groups of test images were created. Each of the image groups contained exactly one image of each of the six universal facial expression. The groups were *frontal face photos*, *near-profile face photos*, *frontal face puppet image*, and *near-profile face puppet image*. The four test image groups are shown in Figures 12.3, 12.4, 12.5 and 12.6.

10 test subjects where each presented for the test images, one image group at a time, in the sequence of presentation above. The images within each group were randomly permutated such that no correlation of facial expressions could be made between the groups in that aspect. For each image the test subjects were asked to tell which one of the six universal facial expressions the face on the test image was imposing. Lines connecting the eyebrow landmark vertices of the test puppet were added to the face puppet, to take advantage of the expressiveness of eyebrows.

The test subjects were one male of age 40-50, French national, and one male and 8 females, all age 20-30 and all Danish nationals.

12.3.3 Test Results

The results of the tests are shown in Tables 12.2, 12.3, 12.4, and 12.5.

Image/Emotion	Sadness	Fear	Happiness	Disgust	Anger	Surprise
Sadness	10	0	0	0	0	0
Fear	0	6	0	2	0	2
Happiness	0	0	10	0	0	0
Disgust	0	0	0	6	4	0
Anger	0	0	0	4	6	0
Surprise	0	0	1	0	0	9

Table 12.2: Frontal face images, test results.

Image/Emotion	Sadness	Fear	Happiness	Disgust	Anger	Surprise
Sadness	10	0	0	0	0	0
Fear	1	1	0	0	0	8
Happiness	0	0	10	0	0	0
Disgust	0	0	0	9	1	0
Anger	0	0	0	0	10	0
Surprise	0	0	1	0	0	9

Table 12.3: Face images from side, test results.

Image/Emotion	Sadness	Fear	Happiness	Disgust	Anger	Surprise
Sadness	9	0	0	0	0	1
Fear	6	4	0	0	0	0
Happiness	0	1	9	0	0	0
Disgust	0	0	0	4	5	1
Anger	0	0	0	0	5	5
Surprise	0	0	3	0	0	7

Table 12.4: Frontal puppet images, test results.

Image/Emotion	Sadness	Fear	Happiness	Disgust	Anger	Surprise
Sadness	9	0	0	1	0	0
Fear	9	0	0	0	0	1
Happiness	1	0	6	0	0	3
Disgust	2	0	0	1	7	0
Anger	0	0	3	2	3	2
Surprise	1	1	4	0	0	4

Table 12.5: Puppet images from side, test results.

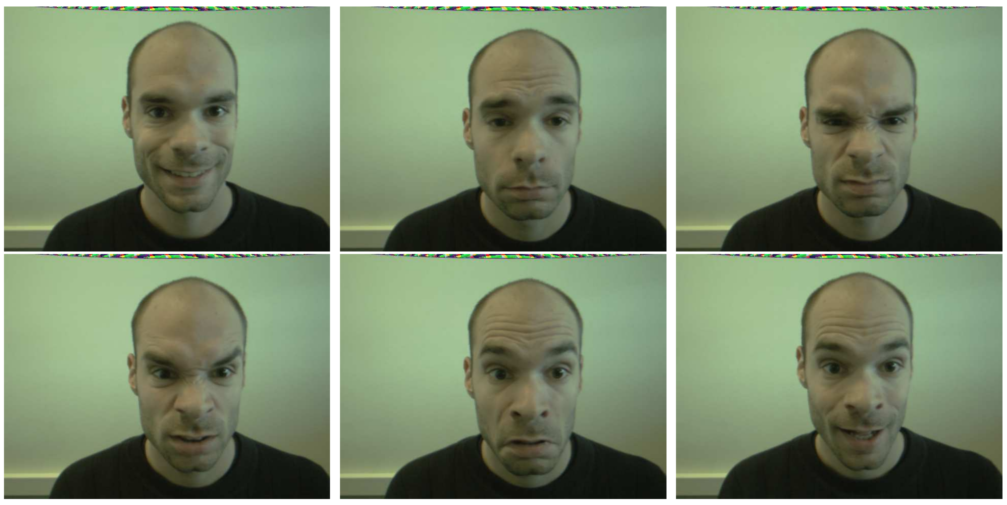


Figure 12.3: Face images, frontal view

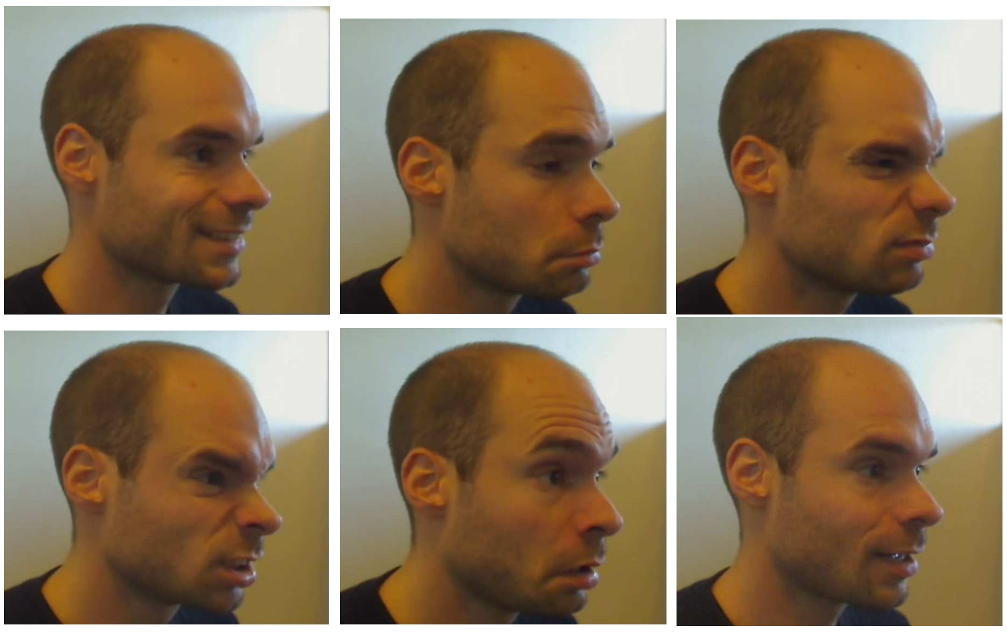


Figure 12.4: Face images, near-profile view

12.3.4 Conclusion

Discussion of Emotion Estimation from Photographies

It is seen in Table 12.2 and Table 12.3 that people generally recognize the emotions expressed in the face photos. However, some expressions, like *disgust* and *anger*, showed hard to distinguish in the frontal view, but easier in profile view. *Fear* was

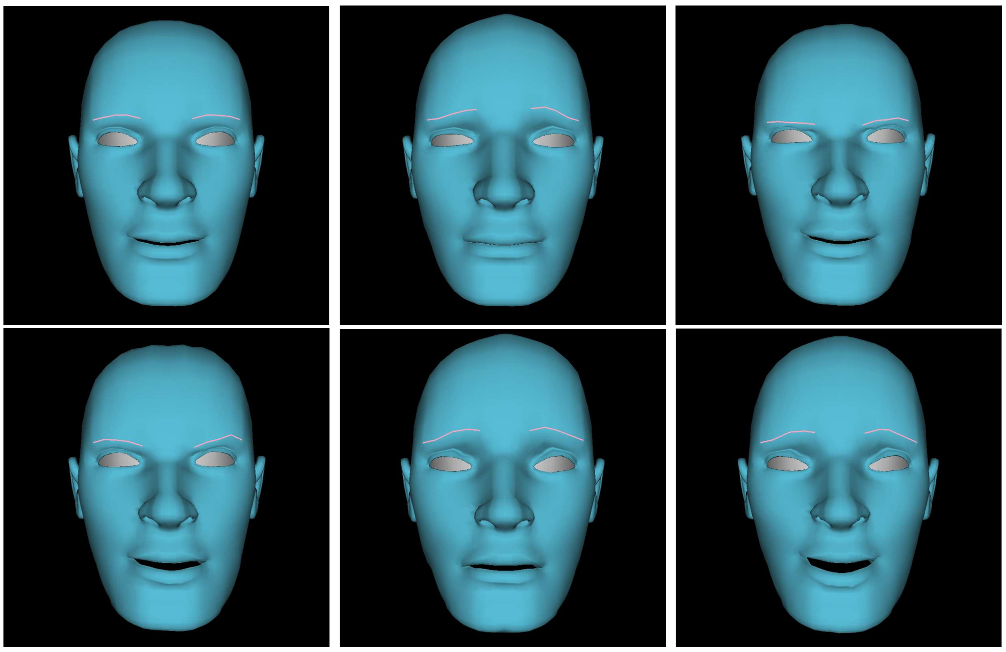


Figure 12.5: Puppet images, frontal view

also hard for the test subjects to recognize, especially from profile view, where it generally was confused with *surprise*. *Sadness* and *happiness* were recognized on both views by all test subjects, and *surprise* was recognized by all test subjects but one in both views.

Discussion of Emotion Estimation from Graphic Faces

In Table 12.4 and Table 12.5 the recognition result of the graphic facial expressions are seen. Again, *sadness* and *happiness* are the most recognizable expressions. The results are not as clear-cut as for the photographies, though. *Fear* is for example confused with *sadness* in most cases in both views.

However, it is seen that there is a certain degree of consistency to the confusion. In the frontal view *fear* is recognized by around half the test subjects as *fear*, while the other half all classifies it as *sadness*. The same type of consistent confusion applies to *disgust* and *anger*, and to *anger* and *surprise*. This can be interpreted as being due to the expressions resemblance. For example both *anger* and *surprise* are expressed by faces with mouth slightly open, and both *fear* and *sadness* expressions have raised the inner part of the eyebrows.

A possible source of confusion is that the emotions expressd by the actor, who is not trainied in this discipline, contains levels of other emotions in them. For example *Anger* contains *disgust*, and *surprise* contains *happiness* (in some cases), etc.

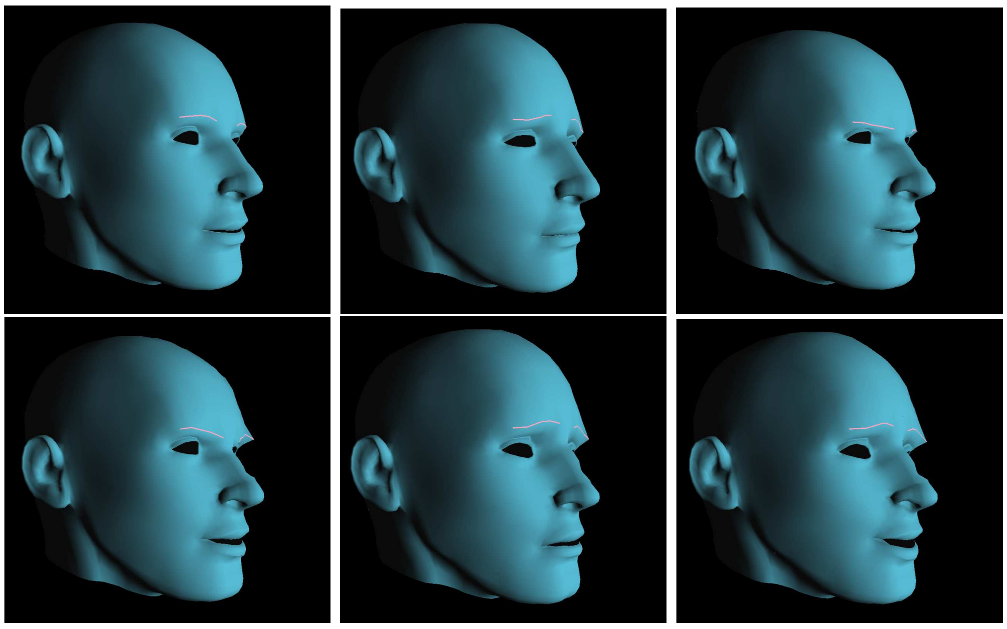


Figure 12.6: Puppet images, near-profile view

Summary of Discussion

Though the results of the test are far from equivocal, it can be concluded that emotions *are* conveyed by the graphic face, and in many cases the intended ones.

12.4 Assumption of Preserved Aspect Ratio

12.4.1 Purpose of Test

This test investigates whether or not the assumption of preserved aspect ratio proposed in Section 9.3 is a valid one.

12.4.2 Test Method

Two sets of shapes are in scope for this test: a set of 2D shapes of the annotations on the left view stereo image, and a corresponding set of 3D shapes, calculated by reprojections based on the 2D annotations. Since it is known that the left view stereo images are taken slightly from the left side, it is expected that the 2D shapes, due to perspective projection, are are narrower than the 3D shapes. Or to put in in terms of the aspect ratio: the 2D shapes are expected to have a higher height/width ratio. See Figure 12.7 for an example of the difference in aspect ratio.

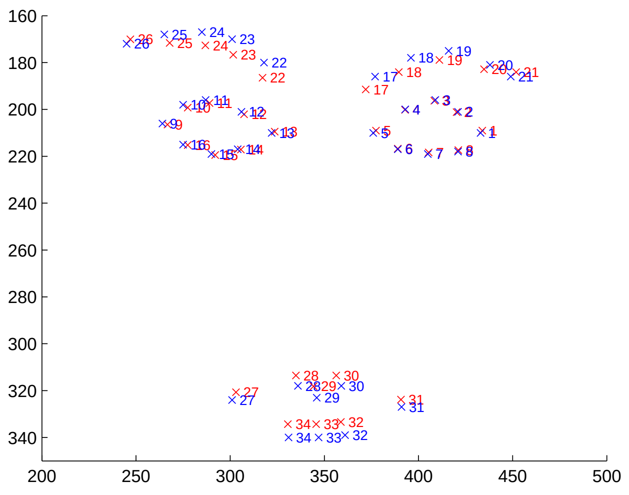


Figure 12.7: The figure shows a plot of a 2D shape (blue) and the xy component of an aligned corresponding 3D shape (red). It can be seen how the aspect ratios differ.

After aligning the 3D shapes to the 2D shapes using Procrustes alignment to maximize their span in the xy-plane, we can obtain two distributions of aspect ratios, one for each set.

The goal of the test is to see if the $height/width$ ratios, $\mathbf{r}_{2d}$, of the 2D shape annotations are "likely enough" to the distribution determined by the $height/width$ ratios, $\mathbf{r}_{3d}$, of the 3D shapes found by re-projection. The distance measure used is *Mahalanobis distance* D_M.

The term "likely enough" is very vaque. However, the properties of Mahalanobis distance provide a reference in this matter: For a univariate point, x_1, that is at a distance of *one* standard deviation from a distributions mean, the Mahalanobis distance is $D_M(x_1) = 1$, and for a univariate point, x_2, that is at a distance of *two* standard deviations form a distributions mean, the Mahalanobis distance is $D_M(x_1) = 4$. So for this test, a point in $\mathbf{r}_{2d}$ having a $D_M < 1$ will definitely be "likely enough".

12.4.3 Test Results

In Table 12.6 the Mahalanobis distances of the 22 2D shape ratios can be seen.

16 out of 22 ratios have a distance lower than 1. The average distance of 22 2D shape ratios was 1.0442, with a few outliers influencing this value substantially.

1	2	3	4	5	6	7	8	9	10	11
0.865	0.496	1.355	1.372	0.221	0.099	0.496	0.018	0.452	0.439	0.004
12	13	14	15	16	17	18	19	20	21	22
0.010	1.178	0.274	0.220	0.937	0.945	0.521	2.420	5.698	0.256	4.697

Table 12.6: Mahalanobis distances for the 22 2D shape ratios to the 3D shape distribution.

12.4.4 Conclusion

From the test it is seen that shape with aspect ratios simular to the 2D shape are likely to occur in the 3D hape distribution. In this light, it is decided that the assumption of preserved aspect ratio is valid.

Part III

Discussion

Chapter 13

Future Work

As a consequence of the modular structure of the motion capture framework described in this thesis, modifications, extentions and improvements can be applied at the level of the individual modules or to the framework as a whole. Following proposition to future work:

13.1 Data Acquisition

The framework is currently reliant on the *Videre Small Vision System* for stereo face images. It would improve the frameworks application as a DIY face motion capture framework to make it camera platform independent by adding a stereo calibration functionality.

13.2 Motion Capture

As it has been noted at several occasions in the thesis, AAM has no way of dealing with occlusion of landmarks. It would be of great interest and application to develop a method that would allow for larger degree of rotation, than can be accomplished under the no-occlusion constraint.

13.3 Animation

It would be very interesing to see how the sparse 2D-to-3D AAM method works in conjunction with a physically based muscle controlled animation model. In such a model local deformations propagate to the rest of the face according to the mucle model. Thus, a method could be devised that would allow a sparse set of control points to animate the face in considerable detail and constrains the vertex translations in the model to realistic facial expressions (or at least physically possible).

Chapter 14

Discussion

14.1 Summary of Main Contributions

The following sections summarize the main contributions of this thesis to the field of facial motion capture research.

14.2 Stereo Face Image Database

A small stereo face image database has been built. Besides the face images, the database includes a set of *.asf3d* 3D shape data files, and *.ini* files containing the stereo calibration parameters. Thus, the database can be readily applied for model training, or be extended with data of identical format.

14.3 Stereo Annotation Tool

A stereo annotation tool has been developed in Matlab. The tool retrieves 3D shape data wiht relativly high (considering the application domain) precision.

14.4 Sparse 2D-to-3D AAM

The sparse 2D-to-3D AAM method has been described and implemented. The method has shown capable of synthesing the 3D geometry of facial expressions from 2D face images.

14.5 3D Face Motion Capture Framework

A motion capture framework has been established to accomodate future exploration and research in 3D face motion capture. It can be argued that establishing a framework is nothing more than defining a set of interfaces for the framework's modules to communicate by. However, for a framework to be appealing for anyone to use, it must implement, at some level, the intended functionality and be able to succefully perform a set of tasks defined under the framework. For the framework described in this thesis these tasks are stereo image annotation, synthesis of 3D facial geometry from 2D images, and conveyance of emotions via an animation front end. In this sense the framework is complete.

14.6 Conclusion

The objectives of this thesis were: to discuss and summarize the process of facial motion capture,to describe a means of acquiring 3D face data, to describe in particular the application of the proposed sparse 2D-to-3D AAM method in facial motion capture, to show that facial expressions can be captured and conveyed by the method of sparse 2D-to-3D AAM. It has been argued in the previous sections that the objectives have been fulfilled. In doing so, this thesis constitutes a significant contribution to the research 3D facial motion capture and thereby the area of human-computer camera interfaces.

Bibliography

[1] Bruce G. Baumgart. Winged-edge polyhedron representation for computer vision. *National Computer Conference*, May 1975. [cited at p. 1]

[2] P. Bergeron and P. Lachapelle. Controlling facial expressions and body movements in the computer-generated animated short tony de peltrie. *SIGGRAPH 85 Advanced Computer Animation seminar notes*, 1985. [cited at p. 1]

[3] Volker Blanz and Thomas Vetter. A morphable model for the synthesis of 3d faces. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 187–194, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co. [cited at p. 2, 8]

[4] F. L. Bookstein. Landmark methods for forms without landmarks: Localizing group differences in outline shape. In *MMBIA '96: Proceedings of the 1996 Workshop on Mathematical Methods in Biomedical Image Analysis (MMBIA '96)*, page 279, Washington, DC, USA, 1996. IEEE Computer Society. [cited at p. 25]

[5] Chris Bregler. Motion capture technology for entertainment. *IEEE Signal Processing Magazine*, 24:156–160, November 2007. [cited at p. 8]

[6] T. F. Cootes and C. J. Taylor. Active appearance models. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 484–498. Springer, 1998. [cited at p. 2, 8]

[7] T.F. Cootes and C.J Taylor. Statistical models of appearance for computer vision, 2004. [cited at p. 2, 8, 23, 30]

[8] Charles Darwin. *Expression of the Emotions in Man and Animals*. John Murray, London, United Kingdom, 1872. [cited at p. 7]

[9] P. Ekman and W. Friesen. Facial action coding system: A technique for the measurement of facial movement. 1978. [cited at p. 7, 42]

[10] David A. Forsyth and Jean Ponce. *Computer Vision: A Modern Approach*. Pearson Education, 2003. [cited at p. 16, 20]

[11] Andrea Fusiello, Emanuele Trucco, and Alessandro Verri. A compact algorithm for rectification of stereo pairs. *Mach. Vision Appl.*, 12(1):16–22, 2000. [cited at p. 17]

[12] George F. Luger. *Artificial Intelligence: Structures and Strategies for Complex Problem Solving (Fifth Edition)*. Pearson Education, 2005. [cited at p. 65]

[13] T.E. Whalen M.D. Cordea, E.M. Pertiu. A 3d-anthropometric-muscle-based active appearance model. *VECIMS, IEEE Symposium*, pages 88–93, July 2004. [cited at p. 8]

[14] Joseph O'Rourke. *Computational Geometry in C (Second Edition)*. Cambridge Univeristy Press, 1998. [cited at p. 65]

[15] Frederic I. Parke and Keith Waters. *Computer Facial Animation*. A. K. Peters, Ltd., Natick, MA, USA, 1996. [cited at p. 33, 35, 60]

[16] Frederic Ira Parke. *A parametric model for human faces.* PhD thesis, 1974. [cited at p. 7]

[17] Frederick I. Parke. Computer generated animation of faces. In *ACM'72: Proceedings of the ACM annual conference*, pages 451–457, New York, NY, USA, 1972. ACM. [cited at p. 60]

[18] S.M. Platt. A system for computer simulation of the human face. Master's thesis, The Moore School, University of Pennsylvania, Philadelphia, 1980. [cited at p. 7]

[19] Donald Shepard. A two-dimensional interpolation function for irregularly-spaced data. In *Proceedings of the 1968 23rd ACM national conference*, pages 517–524, New York, NY, USA, 1968. ACM. [cited at p. 62]

[20] Jianbo Shi and Carlo Tomasi. Good features to track. pages 593–600, 1994. [cited at p. 21]

[21] M. B. Stegmann. Active appearance models: Theory, extensions and cases. Master's thesis, Informatics and Mathematical Modelling, Technical University of Denmark, DTU, Richard Petersens Plads, Building 321, DK-2800 Kgs. Lyngby, aug 2000. [cited at p. 2, 8, 26, 27, 29, 31, 40, 46, 55]

[22] Keith Waters. A muscle model for animation three-dimensional facial expression. *SIGGRAPH Comput. Graph.*, 21(4):17–24, 1987. [cited at p. 7, 35]

[23] Jing Xiao, Simon Baker, Iain Matthews, and Takeo Kanade. Real-time combined 2d+3d active appearance models. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages 535 – 542, June 2004. [cited at p. 8]