DIPLOMARBEIT

3D Active Appearance Models for Segmentation of Cardiac MRI Data

ausgeführt am Institut für Computergraphik und Algorithmen
der Technischen Universität Wien
in Kooperation mit dem
VRVis, Zentrum für Virtual Reality und Visualisierung

unter Anleitung von
Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Eduard Gröller
in Kooperation mit
Dr.techn. Jiří Hladůvka
Dipl.-Math. Dr.techn. Katja Bühler

durch
Sebastian Zambal
Matr. Nr.: 9826978
A - 3340 Waidhofen/Ybbs, Ybbsitzerstr. 44a

Wien, im August 2005

Abstract

Segmentation of volumetric medical data is extremely time-consuming if done manually. This is the reason why currently great efforts are being made to develop algorithms for automatic segmentation. Model based techniques represent one very promising approach. A model representing the object of interest is matched with unknown data. During the matching process the model's shape and additional properties are varied in order to iteratively improve the match. As soon as the model fits sufficiently well to the data, the properties of the model can be mapped to the data and so a segmentation is derived.

Recently the segmentation of cardiac magnetic resonance images (MRI) has been of great interest. In this work we outline some of the methods proposed to solve the problem of cardiac segmentation. We review Active Appearance Models (AAMs) which are a special type of deformable models. AAMs rule changes in shape and texture using statistical information obtained from a data base of representative examples. We describe the theory behind AAMs with special focus on 3D AAMs. These are applicable to volumetric medical image data. Our implementation of 3D AAMs is outlined and the results obtained for 3D segmentation of the left cardiac ventricle are presented.

Kurzfassung

Manuelle Segmentierung von volumetrischen medizinischen Bilddaten kann sich auf Grund des Umfanges der Daten sehr zeitaufwendig gestalten. Es gibt momentan große Anstrengungen, Algorithmen zu entwickeln, die diese Aufgabe weitgehend automatisch erfüllen. Ein sehr vielversprechender Ansatz ist die modellbasierte Segmentierung. Dabei wird ein Modell generiert, mit dessen Hilfe es möglich ist, verschiedene Instanzen des zu segmentierenden Objektes zu generieren. Eine Segmentierung von unbekannten Daten wird dann erzeugt indem die Differenz zwischen originalen Bilddaten und Modell minimiert wird. Sobald Modell und Daten hinreichend genau übereinstimmen, könnnen die Eigenschaften des Modells auf die Daten übertragen werden, woraus direkt eine Segmentierung abgeleitet werden kann.

Wir betrachten volumetrische Bilddaten des menschlichen Herzens, die durch Magnet-Resonanz-Tomographie (MRT) gewonnen wurden. Die vorliegende Diplomarbeit gibt zunächst einen kurzen Überblick über vorgeschlagene Methoden zur Segmentierung der Herzventrikel. Danach werden Active Appearance Models (AAMs) erläutert. Diese stellen eine spezielle Form von deformierbaren Modellen dar. Die Änderungen in Geometrie und Textur von AAMs werden mittels statistischer Methoden realisiert. Die notwendigen statistischen Daten werden dabei aus einer Datenbank von repräsentativen Datensätzen berechnet. Wir erläutern die Theorie hinter AAMs mit speziellem Fokus auf 3D AAMs. Diese können auf volumetrische Bilddaten angewendet werden. Weiters wird eine Implementierung von 3D AAMs umrissen. Abschließend werden die Resultate bezüglich Segmentierung des linken Herzventrikels präsentiert.

Acknowledgements

I would like to thank Professor Eduard Gröller for accepting me as his master student and reviewing my work.

This master thesis was carried out at the VRVis Center for Virtual Reality and Visualization in Vienna. I would like to thank the VRVis for supporting my work and providing me the necessary working facilities.

I thank Katja Bühler for giving me the chance to write my thesis at the VRVis. It is her who also supported me with important advice and valuable comments on my work.

I want to express my deep gratitude to Jiří Hladůvka. This work would not have been possible without him. He introduced me into the topic, supported me with literature, gave me important hints during many interesting discussions, and always encouraged me in my work.

André Neubauer helped me to prepare the medical data and Markus Hadwiger supported me with his excellent volume rendering software. I would like to thank them for their valuable contributions.

Finally I would like to express my gratitude to my girlfriend Heidi and all friends and relatives who have made my studies a good time. Especially I would like to thank my parents Christa and Walter Zambal who always did their best in supporting me.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Imaging techniques like magnetic resonance imaging (MRI), computer tomography (CT) and ultrasound (US) are gaining more and more importance in modern clinical diagnosis. However the interpretation and evaluation of images obtained with these techniques is not a trivial task. Also it is very time-consuming if done completely manually. This is especially the case for higher dimensional image data such as volumetric and/or time-dependent data. Ongoing research in the areas of computer vision and pattern recognition deals with the development of robust automatic and semiautomatic segmentation algorithms. Such algorithms are intended to assist medical doctors in clinical diagnosis by preprocessing and preparing the captured data for further analysis.

Approaches for segmentation based on region growing, thresholding or edge detection in most cases only lead to satisfying results when applied to qualitatively good images. With "qualitatively good" we mean images that do not contain too much noise and depicted objects are represented by more or less homogeneous regions. If this is the case structures in an image (e.g. edges) can be identified easily. In fact most medical images do not have such friendly properties. Noise, acquisition artifacts, complexity, and fuzziness of anatomical structures make it hard – even for the human observer – to correctly distinguish between different organs, blood vessels, tissues, etc. One great drawback of conventional segmentation methods is that they heavily rely on local image features. No information about the general appearance of segmented objects is used. Especially in the medical domain structures and appearances of anatomical objects are known very well. This makes it especially interesting to exploit prior knowledge for the design of robust algorithms for segmentation of medical images.

Model based segmentation denotes a class of very promising approaches that make use of prior knowledge in order to identify objects in images. The term "model" denotes an abstract description of a class of objects (for example hands, faces, etc.). In order to detect and segment an object, the attempt is made to match the model with a suitable region in the given image. The matching process tries to minimize the difference between model and image by changing parameters like position and appearance of the model. This type of identification of objects is often referred to as "analysis by synthesis".

A very simple model-like approach is to use a single representative example of a class of objects as model. For example, if a very representative image of a face is given, an algorithm could use this so called golden image for segmentation of unknown faces. However, this approach is relatively primitive and there exist better and more elaborated types of models.

The problem using a simple golden image is that it is not general and flexible enough. This is where the so-called "deformable models" come into play. As the name suggests these deformable models do not only represent an object of interest but also describe how its shape might deform. Of course the deformation has to be of such a kind that the deformed object still represents a valid instance of the considered class of objects. The ability to deform is exploited when the model is matched with unseen data. In order to achieve better matches, the model is deformed iteratively.

This work focusses on Active Appearance Models (AAMs) which rule their deformation using statistical methods. In the following we give a short historical survey of AAMs.

1.1 Historical Background of AAMs

In the early beginning of model based approaches the idea of aligning objects by changing their position [Besl and McKay, 1992] was of great interest. This problem was a fruitful basis for the development of more elaborated approaches. As an enhancement of the simple alignment, deformable models came up. These models cannot only change their position in order to fit some other instance but may also deform to some extent in order to achieve better matches. Such deformable models use different methods to govern deformations. So-called articulated models only allow connected sliding and rotation of their rigid parts. Active blobs [Sclaroff and Isidoro, 1998; Sclaroff and Isidoro, 2003] deform according to piecewise affine warping. Models based on mechanical Finite Element Methods (FEM) base their deformations on physical properties such as stiffness and elasticity.

A type of deformable models sensitive to local image features are active contours or snakes [Kass *et al.*, 1987; Leymarie and Levine, 1993] and balloons [Cohen, 1991]. Snakes and balloons represent curves that deform according to so-called internal and external forces. The internal forces keep the curve smooth while the external forces pull the curve towards local image features such as edges.

In this work we focus on Active Appearance Models (AAMs). These deform according to statistical features computed by analysis of a representative training set. In order to derive deformations the standard AAM makes use of a prediction scheme based on linear regression. This introduces prior knowledge not only about the appearance of the model but also about how to match the model to unknown data.

AAMs have been introduced under this name first by Cootes and Taylor in 1998 [Cootes *et al.*, 1998a]. A large amount of literature about AAMs has been published since then [Cootes and Taylor, 2000; Cootes and Taylor, 2001b;

Cootes and Taylor, 2001a; Cootes *et al.*, 1999; Matthews and Baker, 2004; Stegmann *et al.*, 2003; Taylor *et al.*, 2000]. AAMs are a direct enhancement of Active Shape Models (ASMs) [Cootes *et al.*, 1995; Dickens *et al.*, 2002]. The idea behind both AAMs and ASMs is to use a deformable shape model that can be matched with unseen data using texture difference between image and model. This difference serves as a criterion for how good a current match is and how it can be improved in the following iteration. ASMs only use textures along scan lines crossing the edges of the model's shape. AAMs use the complete texture from the area or volume where the model overlaps with the data. This makes AAMs very robust when fitting to unseen data. On the other hand the often large amount of texture also introduces problems with respect to computational efficiency.

The basis for building an AAM is a training set of representative examples of the object of interest. To create a model, variations between individual training data sets are analyzed statistically. Since images in the training set basically only include gray values, geometric information is not directly available. This geometric information has to be added by identifying significant points – so-called landmark points – in the examples included in the training set. Such an annotation is often obtained by letting a human expert observer define several landmark points. For the human face, for example, the corners of the eyes or mouth are good landmark points.

The human face has always been a very popular object for experimenting with AAMs [Edwards *et al.*, 1998; Stegmann and Larsen, 2003; Gross *et al.*, 2004]. One reason for this probably is that humans are highly sensitive to even small changes in face appearances. This makes it interesting for us to examine how good computers are able to deal with faces. Indeed many optimizations of AAMs for use with faces have been proposed. Some of them deal with the problem that faces usually are not given in perfect front view but are turned to the side. In such a case the original 2D AAMs cannot reproduce suitable instances. On the other hand a full 3D approach would be too expensive. As a solution several authors propose variations of AAMs which combine 2D and 3D concepts [Cootes *et al.*, 2000; Xiao *et al.*, 2004].

Recently a lot of extensions for AAMs have been proposed. While the original AAMs work with gray value images in 2D, different strategies for adaptations to higher dimensions have been suggested. In theory such adaptations can be done straight forward. Practically there are some critical aspects. Different methods have been proposed to overcome the problems in higher dimensions. Applications include time-continuous (2D+time) segmentation of image sequences [Lelieveldt *et al.*, 2001; Edwards *et al.*, 1998], real-time combined 2D+3D AAMs [Xiao *et al.*, 2004], and bi-temporal 3D AAMs [Stegmann and Pedersen, 2005]. Normally an increase in dimensionality inevitably causes a rapid increase of data. Especially the number of texture samples mounts significantly for higher dimensional data. Large data is the reason why methods for compression of texture data using wavelets [Wolstenholme and Taylor, 1999; Stegmann *et al.*, 2004] or wedgelets [Darkner *et al.*, 2004] recently have been proposed.

Although we do not further consider this type of application we mention that AAMs also have been used for object tracking [Mittrapiyanuruk *et al.*, 2004;

Stegmann, 2001b]. Many interesting refinements of AAMs come from this application domain.

1.2 Problem Statement

Evaluation of a patient's heart function is a medical task that benefits very much from modern non-invasive imaging techniques. With such techniques it is possible to capture volumetric image data of a complete heart cycle. The manual investigation of such 4D data is a tedious task. Furthermore, a manual investigation does not directly allow the extraction of global information such as the exact volume of blood pumped around. Automatic and semiautomatic segmentation methods allow a better visualization of the data as well as the extraction of meaningful higher-level information.

Both Magnetic Resonance Imaging (MRI) and Computer Tomography (CT) have been used for the acquisition of cardiac data. In general CT scanners generate images of much higher quality than MRI. The drawback of CT is that X-rays are used, which are considered much more dangerous for the patient than the magnetic fields used in MRI.

The relatively bad quality of cardiac MRI data makes the development of robust segmentation algorithms a challenge. The goal of this thesis is to show that 3D AAMs are highly suitable for segmentation of volumetric cardiac data. The time-dimension will not be considered. We want to show that even for qualitatively bad MRI data 3D AAMs can successfully be used to segment the left cardiac ventricle with high accuracy.

1.3 Thesis Overview

We describe properties of 3D AAMs how to build them, and how to match them to unseen data. We verify our implementation with a set of 32 volumetric MRI images of the left cardiac ventricle (the left chamber of the heart).

Before we go too much into detail on AAMs we review the problem of cardiac segmentation in general in chapter 2. We outline the techniques that have been proposed so far. In chapter 3 we review the process of building an AAM in 3D. We explain the theoretical aspects and discuss special difficulties coming up in 3D. Further we outline some extensions of the AAM model formulation. Chapter 4 contains a discussion on how a 3D AAM is matched with given volumetric data. This can be done by exploiting prior knowledge about the relation of model parameters and texture differences. In chapter 5 we discuss a few important issues concerning an implementation of 3D AAMs. The results obtained with our implementation are presented in chapter 6. Finally we give a conclusion and discuss future work in chapter 7.

1.4 Mathematical Notation

In order to make formulas easier to read, we follow some rules in mathematical notation. Scalar values are written using cursive letters (e.g.: x, D). Matrices are denoted by bold capital letters (e.g. $\mathbf{M}$). Vectors are column vectors and are denoted by bold lower case letters (e.g. $\boldsymbol{v}$). A matrix whose columns are individual vectors (e.g. $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n$) is denoted by the same but capital letter (e.g. $\mathbf{X}$). Data given as a set of vectors transformed to zero-mean is marked with a hat (e.g. $\hat{\boldsymbol{x}}$).

The L_1-norm of some vector $\boldsymbol{x}$ is the sum of absolute values of its elements and is denoted by single vertical lines:

$$|\boldsymbol{x}| = \sum_i |x_i|. \tag{1.1}$$

The L_2-norm of a vector $\boldsymbol{x}$ is the sum of squares of the vector's elements. The L_2-norm is denoted by double vertical lines:

$$\|\boldsymbol{x}\| = \sqrt{\sum_i x_i^2}. \tag{1.2}$$

The following listing associates frequently used terms with their meaning:

$\mathbf{I}$	the identity matrix
$\boldsymbol{\phi}_i$	the i-th eigenvector
$\boldsymbol{\Phi}$	a matrix whose columns are eigenvectors ($\boldsymbol{\Phi} = [\boldsymbol{\phi}_1 \vert \boldsymbol{\phi}_2 \vert \ldots \vert \boldsymbol{\phi}_k]$)
λ_i	the i-th eigenvalue
$\boldsymbol{\Lambda}$	diagonal matrix whose diagonal elements are eigenvalues ($\boldsymbol{\Lambda} = \operatorname{diag}(\lambda_1, \lambda_2, \ldots \lambda_k)$)
p_s	parameter of a shape model
p_g	parameter of a texture model
p_c	parameter of a combined model
k	number of eigenvectors considered
N_x	number of shape points
N_s	number of shape features
N_g	number of texture features
N_t	number of training sets
α	mean of gray values
β	variance of gray values
tr	trace of a matrix
d_M	Mahalanobis distance metric
τ	a transformation optimally aligning two shapes
φ	level set function
E	expectation value
R_k	regression matrix calculated using k eigenvectors
$\boldsymbol{\delta}_x$	deformation of vector $\boldsymbol{x}$

Chapter 2

Segmentation of the Left Cardiac Ventricle

Segmentation of cardiac data has recently been of great interest. In this chapter we give an overview over this specific application domain. A short anatomical overview is presented in which we outline the general structure of the considered data. We review recently proposed approaches for the problem of cardiac segmentation.

2.1 The Left Cardiac Ventricle

For a better understanding of the problem domain we give a short anatomical overview of our object of interest, the left cardiac ventricle. Cardiac ventricles are the parts of the heart from which blood is pumped around the body. The right ventricle receives deoxygenated blood and pumps it towards the lungs. The left ventricle receives the blood enriched with oxygen and pumps it to the rest of the body. The muscle surrounding the left ventricle is much thicker than the one of the right ventricle.

Figure 2.1 illustrates the overall anatomy of the human heart. Note the left cardiac ventricle (on the right in the figure) which will be of main interest for our further discussion. The figure also roughly shows the position of the volume data that usually is captured for medical diagnosis. The volume of interest resides in between the two black lines which illustrate the bordering planes. The left upper plane lies at the base of the ventricle. The right lower plane lies at the ventricle's apex.

Important clinical properties of the observed data for example are thickening and motion/contraction of the heart muscle over time and the volume of blood pumped around (ejection fraction). The American Heart Association (AHA) suggests a visualization of the measured properties in form of a "bull-eye" view [Cerqueira *et al.*, 2002]. Measurements from three main parts of the left ventricle are shown in rings around the center of the bull-eye. The inner ring represents the apical part of

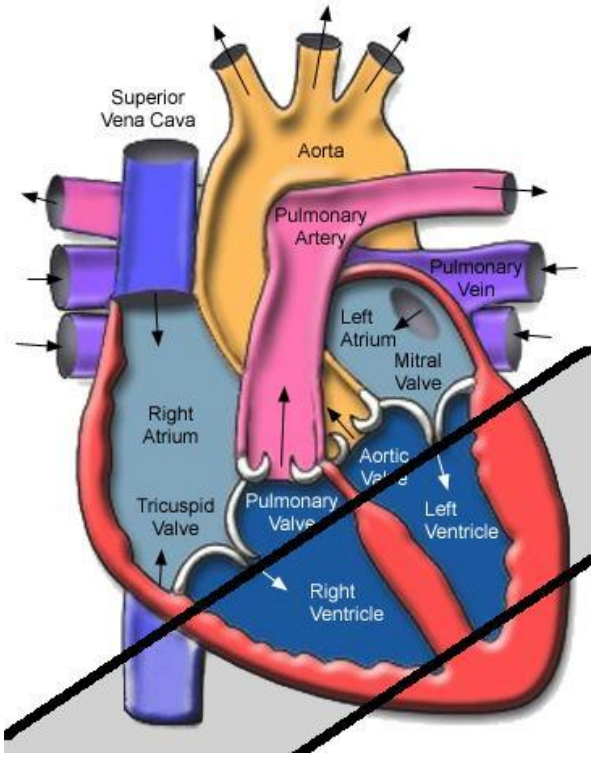


Figure 2.1: Anatomy of the human heart. The two black lines in the figure delimit the volume captured with imaging systems. Figure from [Texas Heart Institute, 2005].

the heart. The farther a ring is away from the center, the closer to the base is the associated region of the heart. The AHA also recommends a detailed naming scheme for the individual parts of the left ventricle. Figure 2.2 shows a bull-eye view with the according naming scheme of individual regions.

The bull-eye view is a good example of how MRI data can be prepared in order to supply medical doctors with important meaningful high-level information. Such prepared information cannot directly be extracted from the MRI data. It can only be derived from a well-done segmentation of this data. This is where 3D AAMs come into play. Manual segmentation of 3D MRI data takes medical doctors quite a lot of time. Highly automated segmentation methods such as AAMs can help to save a lot of time in every-day medical work.

So far the anatomical background. We now take a closer look at the great variety of techniques which have been proposed for the segmentation of cardiac ventricles.

2.2 Snakes

The concept of snakes or active contours originally has been introduced by Kass *et al.* (1987). It is a method that has been applied to many different tasks related to segmentation and tracking. The idea behind snakes is to initially place a curve

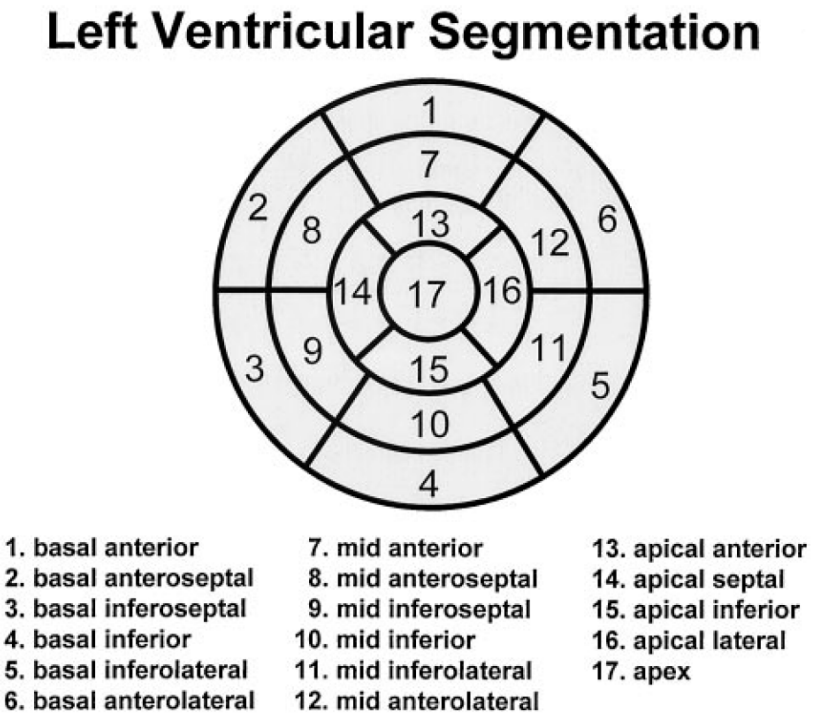


Figure 2.2: The bull-eye view as recommended by the American Heart Association (AHA). Figure from [Cerqueira *et al.*, 2002].

close to some edge in a given image. Then the curve or snake deforms according to internal and external forces so that it optimally approaches the edge.

The standard method is to define a snake as a parametric curve $s(t)$ where t is a parameter ranging from 0 to 1. The energy of a snake is defined as

$$E_s = \int_0^1 E_{int}(s(t)) + E_{ext}(s(t))dt \tag{2.1}$$

where E_{int} and E_{ext} denote the internal and the external energies influencing the snake. The internal energy is a term which makes the snake resistant to bending and stretching. E_{int} is small if the curve is smooth and not stretched. The external energy relates to image features close to the curve. Usually the curve is supposed to be attracted by edges. This is why E_{ext} often is selected to correspond to the image gradient:

$$E_{ext} = -\|\nabla Img\|^2. \tag{2.2}$$

The external energy in this case becomes smaller and smaller as the curve approaches an edge in the image.

A typical application based on snakes allows the user to manually draw an initial rough guess of an object's contour. This manually drawn curve is considered to be the initial snake. Iteratively the energy term E_s is then minimized, which deforms the snake. In other words the snake propagates through the image and approaches edges while remaining sufficiently smooth. Of course for the relative importance

of external and internal energies a tradeoff has to be found which depends on the considered application.

Note that equation 2.1 is a continuous formulation of snakes because of the integral it includes. To ease an implementation of snakes often the discretized version [Lobregt and Viergever, 1995] is used. The continuous snake is replaced by points connected by line segments. The energy function then is only calculated for the given points and the integral in equation 2.1 becomes a discrete sum.

Several authors have proposed slightly differing strategies in using snakes for cardiac segmentation [Appleton, 2003; Mikic *et al.*, 1998; Pluempitiwiriyawej *et al.*, 2005]. All the proposed methods differ in the type of energy functions that are applied to the snakes. Pluempitiwiriyawej *et al.* (2005) introduce a special external force. Not the plain image gradient is used in the external energy term. Instead statistical properties of the image data are analyzed. The intention behind this is to make snakes more robust to noisy data. In the next section we review a similar method.

2.3 A Skeleton-Based Segmentation Technique

Recently a technique has been proposed that performs segmentation of 4D cardiac data by building a skeleton of the left ventricle [Neubauer and Wegenkittl, 2003]. The first step in the segmentation process has to be carried out manually. For a single slice the center line of the myocardium of the left ventricle has to be drawn.

The gray values and gradients close to the center line are calculated. This information is used to characterize the appearance of the heart muscle (myocardium) which is to be segmented. Using this precalculated information, the points defining the center line are propagated towards the outer border (epicardium) as well as towards the inner border (endocardium) of the left ventricle. The propagation stops when the distribution of gray values under the current position of the curve varies significantly from the originally calculated appearance. The positions of points representing the propagated curve constitute the skeleton for the final segmentation. The original, manually drawn centerline can now be improved by centering it relative to the calculated inner and outer contours. Once this improved centerline is determined it can be propagated to other slices of the considered time step. In this way the complete volume can be segmented. Also the propagation from the current volume to the volume of the previous or next time step is possible.

Finally each voxel has to be classified in order to make the segmentation complete. This is achieved by calculating a weighted path from individual voxels to the nearest points of the skeleton. The length of the path is calculated using the Euclidean distance and the values of the voxels that are traversed. If the length is below some threshold the voxel is assumed to belong to the left ventricle.

The outlined method is typical for many approaches using an extended version of snakes. Note that snakes do not make extensive use of prior knowledge. For the above skeleton-based technique only the initial center line encodes some prior information about the ventricle's overall shape. The user has to provide this prior

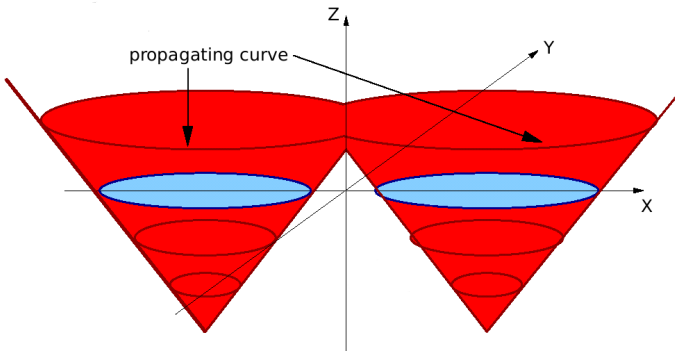


Figure 2.3: Curves as zero level sets. Figure from [Sethian, 1999].

information by manually drawing the center line. One goal of research currently is to reduce such interaction.

2.4 Level Set Methods

A problem of the traditional formulation of snakes as parametric curves is that topological changes cannot be handled very well. Also the propagation of the curve can be complicated with an explicit formulation of the snake. This is why often level set representations are used in order to better deal with propagating curves.

Level sets were first introduced by Osher and Sethian (1988). Following this approach curves are represented implicitly. A level set function φ is used that for each point $\boldsymbol{x}$ evaluates to the minimal Euclidean distance between $\boldsymbol{x}$ and the curve s. Usually the modelled curve is closed and each point can be considered inside or outside. The sign of φ then depends on whether the considered point $\boldsymbol{x}$ lies inside the curve (s_{in}) or outside (s_{out}). φ is equal to zero where a point lies upon s:

$$\varphi(\boldsymbol{x}) = \begin{cases} 0, & \boldsymbol{x} \in s \\ -\|\boldsymbol{x}, s\|, & \boldsymbol{x} \in s_{in} \\ +\|\boldsymbol{x}, s\|, & \boldsymbol{x} \in s_{out} \end{cases} \tag{2.3}$$

The level set representation introduces an additional dimension. For example a curve in 2D can be regarded as the cut of a 3D surface. Figure 2.3 illustrates this principle. The red cones represent the level set function. The two blue ellipses show the 2D curve for $z = 0$.

The level set function is updated in order to propagate the curve. To find out the current shape at a specific point in time the level set function is cut at $\varphi = 0$. Thus the propagation of the curve is replaced by a dynamic change of the level set function. The motion of the level set function is governed by the level set equation:

$$\varphi_t + F\|\nabla\varphi\| = 0 \tag{2.4}$$

where F denotes a speed function and φ_t is the initial state of the level set function. Note that equation 2.4 formulates an initial value problem. Figure 2.3 illustrates

the propagation of a 2D curve by moving the cut through a 3D surface. Note how the topological change is performed implicitly when the two curves meet. For a more detailed discussion of level set methods we refer to more specialized literature [Sethian, 1999; Osher and Sethian, 1988].

Level set methods have also been used for the segmentation of cardiac data [Rousson *et al.*, 2004; Paragios *et al.*, 2002]. Also a segmentation of brain ventricles with level sets has been reported [Rousson *et al.*, 2004].

The considered medical data often includes strong irregularities and so the proposed applications all need a good initial guess in order to converge correctly. Similarly to level sets snakes often suffer from the bad quality of data and easily diverge. This is why currently many researchers try to incorporate more prior knowledge about the shape of analyzed anatomical structures into their level set methods.

2.5 Using Prior Knowledge

All the previously discussed approaches using snakes, level sets or similar techniques have a great weakness in common: only little prior knowledge is used. The general appearance of the object of interest is not considered. This is why recently techniques have gained popularity that extensively make use of existing knowledge of the analyzed anatomical objects.

2.5.1 Atlas-based methods

The idea behind atlas based methods is to use a data base of segmented data sets in order to segment new data sets. An appropriate atlas is selected from the data base and matched with the unseen data. This matching or registration can either be achieved in a rigid or non-rigid way. Rigid registration performs only a transformation while non-rigid registration additionally deforms the atlas image in order to increase matching accuracy.

2.5.2 Statistical Models

Statistical models are an extension of atlas-based methods. Not only a single atlas is used for matching but multiple atlases are combined to create new instances. This has two main advantages. First, the atlas or model can be deformed in a plausible way. This is the case because the statistical shape variances of an object can be calculated. Second, many different appearances of texture can be modelled, which increases the model's matching capabilities.

Chapter 3

Active Appearance Models

A very popular type of statistical models are Active Appearance Models (AAMs) on which we focus in this thesis. Recently AAMs have also become very popular for segmentation of cardiac data [Bosch *et al.*, 2002; Breen *et al.*, 2002; Mitchell *et al.*, 2001; Mitchell *et al.*, 2002; Montagnat and Delingette, 2005; Stegmann, 2001a]. The elegant mathematical formulation of AAMs and their robustness make them very suitable for cardiac segmentation.

The process of building a computerized model frequently is equated with making the computer learn possible appearances of an object. The basis for such a learning process is some ground truth, the training set. The idea of teaching the computer general knowledge has made the field of deformable models interesting for both, researchers in computer graphics and researchers in the areas of cognitive and neurological sciences [Vetter and Poggio, 1997]. To understand the model based approaches from the cognitive point of view is worthwhile. Researchers working in the field of cognitive science nicely formulated the demands on cognitive visual systems [Riesenhuber and Poggio, 2000]:

> A vision system needs to generalize across huge variations in the appearance of an object such as a face, due for instance to viewpoint, illumination or occlusions. At the same time, the system needs to maintain specificity.

The "vision system" described in this thesis – the classical Active Appearance Model – tries to explain the above mentioned variations in terms of statistics. It has recently been shown that such a statistical approach is very robust and stable. Before we describe the actual model building process we first introduce statistical principles which are the basis for AAMs.

3.1 Statistical Analysis of Large Data

Statistical analysis is used to describe important properties of large data with a small set of meaningful values. This is exactly what we need for representing a model. We want to model the appearance of large data with only few parameters.

The large data we have to handle in the context of this thesis mainly is image information which in the simplest case is given in form of bitmap images. For example human faces can very well be represented using such bitmap images. However humans talking about faces use information that is more abstract than pixel colors of bitmap images. We rather use information like “long/short nose”, “large/small eyes” etc. This allows us to describe variances in appearance of an object with compact and highly semantic pieces of information. For AAMs statistical methods analogously are the key to determine the necessary meaningful high-level information.

For the moment we do not care about the exact structure of data we want to analyze statistically. This will be discussed in detail later. We only state that each example in a training set can be represented by an n-dimensional vector (for example such a data vector could contain any low-level information like gray-level intensities of pixels, etc.).

With the abstraction from object instances to simple vectors in mind one can start to think about the actual statistical analysis. A simple statistical approach is to calculate the element-wise arithmetical mean value of a set of high-dimensional vectors. However the mean value alone is not very flexible and this is why often statistical variances are considered additionally. As long as the considered data can be assumed to follow a Gaussian distribution the description of the data using mean values and variances is a good choice. This is the case because the complete structure of a Gaussian distribution is completely determined by mean values and variances.

AAMs are intended to model variations. This is the case why we concentrate mainly on variances. Statistical methods for analyzing variances are also referred to as second order methods. One very popular second order method is Principal Component Analysis (PCA) which we discuss below.

3.1.1 Principal Component Analysis - Overview

For many applications Principal Component Analysis (PCA) is the standard approach to find out the data’s inherent structure. PCA is also used in the AAM building process.

The extraction of statistical features with PCA can be regarded as a projection of high-dimensional data vectors into a space of much lower dimensionality. PCA is also referred to as Karhunen-Loève transform or Hotelling transform [Hyvärinen, 1999].

As its name suggests PCA tries to determine the principal components of some given data which describe the largest amount of the data’s variation. Figure 3.1 illustrates the principal components of a set of points in 2D. The first principal component (PC 1) points into the direction of the largest variance in the point-cloud. The second principal component (PC 2) is perpendicular to the first one and defines the direction of the second largest variance. In 2D only two principal components are defined. Note that two important assumptions are made. First the

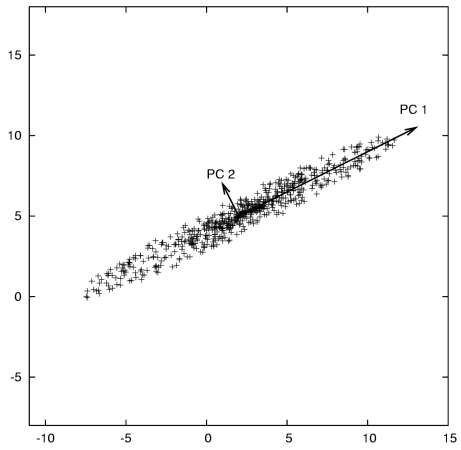


Figure 3.1: Principal Components of data in 2D.

data is assumed to be linear and second it is assumed that the principal components are mutually orthogonal.

Having another look at figure 3.1 one can easily see that both principal components PC 1 and PC 2 form a new basis for the depicted points. Each original point $\boldsymbol{x}_i$ can be calculated as the sum of the mean plus a linear combination of PC 1 and PC 2:

$$\boldsymbol{x}_i = \overline{\boldsymbol{x}} + \sum_{i=1}^{k} \boldsymbol{\phi}_i p_i \tag{3.1}$$

where k is the number of principal components and the vectors $\boldsymbol{\phi}_i$ are the principal components. The mean vector $\overline{\boldsymbol{x}}$ is the arithmetic mean of the components of the given data vectors:

$$\overline{\boldsymbol{x}} = \frac{1}{N} \sum_{i=1}^{N} \boldsymbol{x}_i \tag{3.2}$$

with N being the number of data vectors $\boldsymbol{x}_i$. Usually the principal components are sorted. The first principal component explains the largest amount of variance, the last principal component explains the smallest amount of variance. In order to reduce dimensionality the last few principal components are simply ignored and left out. This is legitimate since small variations in data can often be considered to reflect noise. In figure 3.1 for example PC 2 could be omitted as basis vector. This would reduce the dimensionality of the data to 1. The linear combination could only describe points lying on a straight line through the mean pointing into the direction of PC 1. As long as PC 2 is small enough this means that only very little information is lost because of this dimensionality reduction.

3.1.2 Recursive Definition of PCA

Geometrically seen the mean vector introduces a shift of the coordinate system. This is not very exciting and the more important concept of PCA is variance. In

order to focus on variance the mean vector $\overline{\boldsymbol{x}}$ simply is subtracted from the original data vectors $\boldsymbol{x}_i$: $\hat{\boldsymbol{x}}_i = \boldsymbol{x}_i - \overline{\boldsymbol{x}}$. Then only the remaining zero-mean data vectors $\hat{\boldsymbol{x}}_i$ are considered.

The length of vectors representing the principal components is not important. The only interesting information they contain are the directions of principal components. Therefore we follow the convention to use unit vectors to represent principal components: $\|\boldsymbol{\phi}_i\| = 1$.

Intuitively principal components can be defined recursively. The first principal component is calculated as the unit vector $\boldsymbol{\phi}_1$ which aligns with the direction of the maximum variance that is observed in the data:

$$E\{(\boldsymbol{\phi}_1^T \hat{\boldsymbol{x}})^2\} \rightarrow max \tag{3.3}$$

where $E\{f(\hat{\boldsymbol{x}})\}$ denotes the expectation value of $f(\hat{\boldsymbol{x}}) = (\boldsymbol{\phi}_1^T \hat{\boldsymbol{x}})^2$. The expectation value is defined

$$E\{f(\hat{\boldsymbol{x}})\} = \sum_i^N f(\hat{\boldsymbol{x}}_i) P(\hat{\boldsymbol{x}}_i) \tag{3.4}$$

with $P(\hat{\boldsymbol{x}}_i)$ being the probability of vector $\hat{\boldsymbol{x}}_i$. This probability is set to 1 for all data sets: $P(\hat{\boldsymbol{x}}_i) = 1$ for all $1 \leq i \leq N$.

The following principal components are calculated in a similar way. Having computed the direction of the first principal component this direction can be removed from the data by projecting the data to the according perpendicular hyperplane through the origin. Note that the assumption is made that all principal components are orthogonal. This means that all other principal components have to reside within the mentioned hyper-plane. Every following principal component then is determined within the projected data. The k-th principal component is defined as the unit vector $\boldsymbol{\phi}_k$:

$$E\{(\boldsymbol{\phi}_k^T (\hat{\boldsymbol{x}} - \sum_{i=1}^{k-1} \boldsymbol{\phi}_i \boldsymbol{\phi}_i^T \hat{\boldsymbol{x}}))^2\} \rightarrow \max. \tag{3.5}$$

3.1.3 PCA and Eigenanalysis

The above definition of principal components is intuitive but can hardly be used for calculation. To calculate PCA therefore another approach is used which is based on the analysis of the covariance matrix.

The elements of the covariance matrix are the covariances of each pair of elements of the original data vectors:

$$\mathbf{S}_x = \frac{1}{N-1} \sum_{i=1}^{N} (\boldsymbol{x}_i - \overline{\boldsymbol{x}})(\boldsymbol{x}_i - \overline{\boldsymbol{x}})^T = \frac{1}{N-1} \hat{\mathbf{X}} \hat{\mathbf{X}}^T \tag{3.6}$$

where N denotes the number of original data vectors.

We have already mentioned that PCA performs a change of the basis. Formally we can write this down as a simple multiplication of the original data matrix transformed to zero-mean $\hat{\mathbf{X}}$ by some matrix $\mathbf{P}$:

$$\hat{\mathbf{Y}} = \mathbf{P}\hat{\mathbf{X}}. \tag{3.7}$$

The column vectors of $\hat{\mathbf{X}}$ are the zero-mean data vectors $\hat{\boldsymbol{x}}_i$ and the rows of $\mathbf{P}$ form the new basis vectors. Matrix $\hat{\mathbf{Y}}$ represents the data in $\hat{\mathbf{X}}$ after the change of the basis. We can now derive the covariance matrix $\mathbf{S}_y$ of matrix $\hat{\mathbf{Y}}$:

$$\begin{aligned}
\mathbf{S}_y &= \frac{1}{N-1}\hat{\mathbf{Y}}\hat{\mathbf{Y}}^T && (3.8)\\
&= \frac{1}{N-1}(\mathbf{P}\hat{\mathbf{X}})(\mathbf{P}\hat{\mathbf{X}})^T && (3.9)\\
&= \frac{1}{N-1}\mathbf{P}\hat{\mathbf{X}}\hat{\mathbf{X}}^T\mathbf{P}^T && (3.10)\\
&= \frac{1}{N-1}\mathbf{P}(\hat{\mathbf{X}}\hat{\mathbf{X}}^T)\mathbf{P}^T && (3.11)\\
&= \mathbf{P}\mathbf{S}_x\mathbf{P}^T. && (3.12)
\end{aligned}$$

The goal of PCA is to derive a representation where the variance along individual principal components is maximized. This maximization of variance along the principal components means that the variance of individual components increases while the covariance between different components decreases. The "optimal" covariance matrix for an "optimal" representation would thus be a diagonal matrix with all off-diagonal elements set to zero. This is the point where eigenvectors come into play. The eigenvectors $\boldsymbol{\phi}_i$ and eigenvalues λ_i of a symmetric matrix $\mathbf{A}$ fulfill the following equation:

$$\mathbf{A}\boldsymbol{\phi}_i = \lambda_i\boldsymbol{\phi}_i \qquad \forall i = 1 \ldots N \tag{3.13}$$

where i is the index of eigenvalues and eigenvectors. A matrix $\boldsymbol{\Phi}$ with columns that are individual eigenvectors is introduced. The corresponding eigenvalues can be written as a diagonal matrix $\boldsymbol{\Lambda} = \operatorname{diag}(\lambda_1, \lambda_2, \ldots, \lambda_N)$. Equation 3.13 can so be reformulated:

$$\mathbf{A}\boldsymbol{\Phi} = \boldsymbol{\Lambda}\boldsymbol{\Phi} \tag{3.14}$$

Matrix multiplication with a diagonal matrix is commutative. Furthermore $\boldsymbol{\Phi}$ is orthonormal [1] and thus its transpose is equal to its inverse. Considering this we can derive the following equation:

$$\mathbf{A} = \boldsymbol{\Phi}\boldsymbol{\Lambda}\boldsymbol{\Phi}^T. \tag{3.15}$$

Equation 3.15 can be used to extend the formulation of covariance matrix $\mathbf{S}_y$ from equation 3.12. Replacing $\mathbf{S}_x$ by its eigenvectors and eigenvalues (using equation 3.15) we get:

$$\begin{aligned}
\mathbf{S}_y &= \mathbf{P}\mathbf{S}_x\mathbf{P}^T && (3.16)\\
&= \mathbf{P}(\boldsymbol{\Phi}_x\boldsymbol{\Lambda}_x\boldsymbol{\Phi}_x^T)\mathbf{P}^T && (3.17)\\
&= (\mathbf{P}\boldsymbol{\Phi}_x)\boldsymbol{\Lambda}_x(\boldsymbol{\Phi}_x^T\mathbf{P}^T) && (3.18)
\end{aligned}$$

[1] Matrix $\boldsymbol{\Phi}$ is orthonormal because we agreed on a normalized representation of eigenvectors. Furthermore the eigenvectors are mutually orthogonal.

If we finally choose $\mathbf{P}$ to be equal to $\mathbf{\Phi}_x^T$, we can even simplify this to:

$$\mathbf{S}_y = \mathbf{\Lambda}_x \tag{3.19}$$

In other words, if we choose the eigenvectors of the covariance matrix $\mathbf{S}_x$ as a new basis for the data representation, $\mathbf{S}_y$ is diagonalized. This means that the desired principal components are the eigenvectors contained in matrix $\mathbf{\Phi}_x$.

The eigenvector that corresponds to the largest eigenvalue represents the largest variance and thus marks the first principal component. A sorting of all eigenvectors with respect to decreasing corresponding eigenvalues leads to the correct sequence of principal components. The influence on data representation decreases as the eigenvalues decrease. Removing the last few principal components is equal to ignoring the smallest variances. Especially if eigenvalues are equal to zero this means that the corresponding eigenvector describes zero variance and can be removed without any loss of information. In general the maximum number of eigenvectors with eigenvalues not equal to zero is $min(n, N) - 1$, where n is the dimensionality of the data vectors and N the number of data vectors.

Using PCA on high dimensional data imposes the question which criteria should be used to determine how many eigenvectors can be removed without simplifying the data "too much". A simple method is to argue that the complete variation in the data relates to the sum of all eigenvalues $V = \sum \lambda_i$. To maintain a reproducibility of at least a fraction r of the original training set, the least important eigenvectors with an index greater than k can be removed as long as the following equation holds:

$$\sum_{i=1}^{k} \lambda_i \geq rV. \tag{3.20}$$

In the following we summarize the complete process of calculating PCA. Given some data in form of a matrix $\mathbf{X}$ whose columns are the vectors $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N$, we calculate the mean $\overline{\boldsymbol{x}}$ and the covariance matrix $\mathbf{S}_x$. By determining the eigenvectors $\boldsymbol{\phi}_1 \boldsymbol{\phi}_2 \ldots \boldsymbol{\phi}_k$ and eigenvalues $\lambda_1, \lambda_2 \ldots \lambda_k$ from $\mathbf{S_x}$ we get the principal components. Keeping the k most important eigenvectors and removing all others we obtain the final compact representation in form of a linear combination of eigenvectors (equation 3.1). This equation can be written using a matrix whose columns are the eigenvectors $\boldsymbol{\phi}_i$:

$$\boldsymbol{x} = \overline{\boldsymbol{x}} + \mathbf{\Phi}_x \boldsymbol{p} \tag{3.21}$$

where $\boldsymbol{p}$ is the vector including the values p_i as elements. Together with the eigenvalues λ_i equation 3.21 represents the result of PCA applied to some data matrix $\mathbf{X}$.

Note that equation 3.21 not only allows to approximately reproduce the original data. It also makes it possible to create new data being an interpolation of the original one. Each instance one can create is defined by the values p_i. However the newly created data is only similar to the original instances if the absolute values of p_i do not become too large (which means that they differ too much from the mean and thus also from all the original data). Therefore usually the values of all the p_i are restricted to an interval around zero correlating with the magnitude of λ_i. For

more information on how to limit the p_i to reasonable intervals we refer to section 3.5.

3.1.4 Using the Transposed Covariance Matrix

When building models the data vectors representing instances of the training set usually are very large while the number of vectors is relatively small. This means that the dimensionality n (the number of elements in the original data vectors) is much higher than the size of the training set N_t: $n \gg N_t$. In this case the covariance matrix $\mathbf{S}_x$ calculated in equation 3.6 gets a size of $n \times n$. As a consequence the calculation of eigenvectors and eigenvalues computationally gets very costly. However there is a way to avoid this high computational effort [Cootes and Taylor, 2000; Stegmann, 2000].

One can calculate eigenvectors $\mathbf{\Phi}'$ and eigenvalues $\mathbf{\Lambda}'$ of a (different) matrix $\mathbf{S}'$:

$$\mathbf{S}' = \frac{1}{N-1}\hat{\mathbf{X}}^T\hat{\mathbf{X}}. \tag{3.22}$$

This differs from equation 3.6 in that the multiplication of $\hat{\mathbf{X}}$ with its transpose is permuted. We can further derive:

$$\begin{aligned}
\mathbf{S}'\mathbf{\Phi}' &= \mathbf{\Lambda}'\mathbf{\Phi}' && (3.23)\\
\frac{1}{N-1}\hat{\mathbf{X}}^T\hat{\mathbf{X}}\mathbf{\Phi}' &= \mathbf{\Lambda}'\mathbf{\Phi}' && (3.24)\\
\frac{1}{N-1}\hat{\mathbf{X}}\hat{\mathbf{X}}^T\hat{\mathbf{X}}\mathbf{\Phi}' &= \mathbf{\Lambda}'\hat{\mathbf{X}}\mathbf{\Phi}' && (3.25)\\
\mathbf{S}_x\hat{\mathbf{X}}\mathbf{\Phi}' &= \mathbf{\Lambda}'\hat{\mathbf{X}}\mathbf{\Phi}'. && (3.26)
\end{aligned}$$

The last equation expresses that $\hat{\mathbf{X}}\mathbf{\Phi}'$ is a matrix of eigenvectors of $\mathbf{S}_x$ and $\mathbf{\Lambda}'$ is the diagonal matrix of according eigenvalues. Instead of calculating the eigenvectors and eigenvalues of the large matrix $\mathbf{S}_x$ we can calculate them for the much smaller matrix $\mathbf{S}'$. To achieve the correct eigenvectors we only have to multiply the calculated matrix $\mathbf{\Phi}'$ by $\hat{\mathbf{X}}$.

3.2 Shape Model

The first type of information included in an AAM normally is referred to as shape. Before we discuss the actual statistical analysis of shape we consider some general issues about shape and how shape information is obtained. We also take a look at how the shape of a complete training set can be normalized. For an introduction to statistical shape models we also refer to Stegmann and Gomez (2002).

3.2.1 What is Shape?

In everyday speech the term *shape* is used to refer to the outline or contour of an object. In computer vision *shape* usually refers to geometrical properties in a more

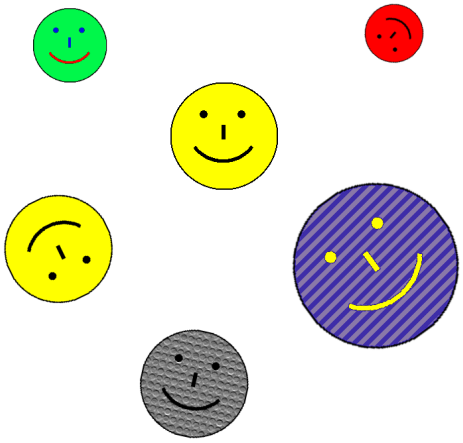


Figure 3.2: Invariance of shape. All smileys share the same shape independent of rotation, scaling, translation and texture.

general way. With the purposes and applications of computer vision in mind, shape can be regarded as [HyperDictionary, 2005]:

> "the spatial arrangement of something as distinct from its substance."

In the field of AAMs the mentioned substance distinct from shape is nothing else but the actual gray values or pixel colors of an image.

Very important for the concept of shape is also that shape has to be independent of rotation, scaling and translation. The shape of something remains the same no matter whether it appears rotated, scaled or translated. Figure 3.2 illustrates this principle.

In human language the shape of an object is often described by comparing it with another object, a geometrical figure or giving it attributes like "thin", "angular", "round", etc. Such descriptions are relatively vague and can of course not be understood by a computer. To be able to treat shape with an exact algorithm, we need a more mathematical method.

In computer vision and pattern recognition the concept of landmarks was introduced to effectively represent shapes. Identifying the shape of an object then means to identify distinctive points – the landmark points – in the geometrical appearance of this object. The shape of a human face, for example, could be described by identifying the corners of the mouth or the eyes as landmark points. This process of placing landmark points is called annotation. By carrying out the annotation for an image of some object, we make explicit the shown object's geometry or shape information. Indeed this annotation information is the shape information that can be further processed by an algorithm running on a computer.

3.2.2 Obtaining Landmark Points

A crucial problem is how to correctly identify landmarks in a given image. The simplest approach is to let a human expert manually create the annotation. However there are several arguments against this solution. The most obvious of these arguments is that it takes a lot of time to annotate large data. This makes it very impractical and can be costly when the annotation has to be carried out by a well trained expert as it is the case for clinical applications. Also the human expert might make mistakes or the manual annotations can be slightly inaccurate. In this thesis we concentrate on AAMs in 3D. Since volume data already is not easy to visualize it is even harder to find good landmark points manually in 3D.

In practice there exist relatively few really significant and unique points that can be identified manually as exact landmark points. To increase the number of landmarks, additional points are introduced using an interpolation scheme or local image features. A general classification of landmark points with respect to how they are placed is given below.

- **Anatomical landmarks** correspond between organisms in some biologically meaningful way. Such points are usually identified by a human expert.
- **Junctions** are points where different clearly distinguished boundaries meet.
- **Mathematical landmarks** are located using some mathematical property, i.e. high curvature or an extremum.
- **Pseudo landmarks** are (usually evenly spaced) points upon an object's surface or outline between existing landmarks.

Figure 3.3 illustrates the different types of landmark points. It has to be stated that the identification of landmark points in 3D is much harder than in 2D. For example T-junctions which are often relatively easy to determine in 2D often appear much more diffuse in 3D. The number of landmark points which are well-defined often is relatively low in 3D medical data. Therefore intermediate landmark points are used to a large extent, which in general is not so good for the quality of the model.

AAMs use statistical methods and learn appearance not only from a single golden image but from a whole training set. To provide the algorithm with the complete shape information that is necessary, landmarks have to be defined for each example in the training set. Besides making a manual annotation even more complex this poses the problem of correspondence between annotations of different examples. Any landmark point in one training example has to define exactly the corresponding salient point as in all other training examples. This seems to be no problem if suitable distinctive points exist in the training data. However, especially for volumetric medical data this is rarely the case.

Currently intensive research is done to develop new methods to find good corresponding landmark points automatically [Davies *et al.*, 2001; Davies *et al.*, 2002; Taylor *et al.*, 1999; Kotcheff and Taylor, 1998; Bookstein, 1997; Brett and Taylor, 1998]. Most of these methods start with some not so good annotation and try to

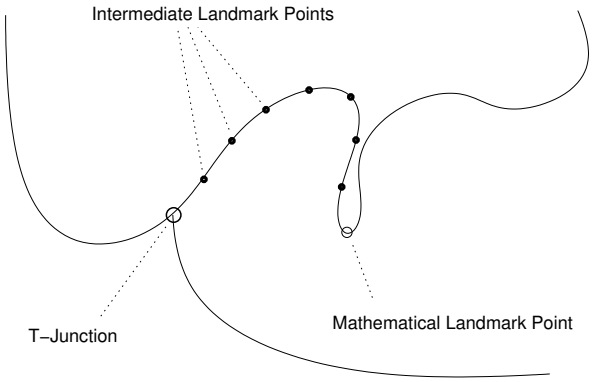


Figure 3.3: The different types of landmark points.

increase correspondence between correlating points of different data instances by displacing the initial landmark points slightly. A very promising approach [Davies *et al.*, 2001] uses the description length of a model as a criterion for its compactness and quality. The description length is a concept from information theory. It is roughly spoken the amount of information needed to transmit all information making up the model. The idea is that good, corresponding landmark points lead to a small description length. The problem of finding good landmark points can then be seen as an optimization problem for which the description length has to be minimized by changing the coordinates of landmark points.

A critical aspect concerning volumetric medical data is that artifacts due to motion of the patients frequently appear. This is the case when individual slices are captured one after the other with some time gap in between. In order to get good landmarks some kind of motion compensation has to be performed. For cardiac data, artifacts are often the result of respiratory motion and so the slices are displaced in a similar way. The question is whether it could make sense to omit motion compensation so that the model built from the according data also "learns" the most common displacements that occur. However in general one wants to keep out such motion distortions from the landmarking information. We will return to the problem of motion artifacts in chapter 6.

3.2.3 Aligning two Shapes in 3D

The first step in analysis of shapes is to optimally align the considered shapes in order to filter out differences due to translation, scale and rotation. In this context we consider shapes to be sets of points each shape consisting of the same number of N_x points. In this section we discuss alignment of pairs of shapes. In the next section we show how the presented technique to align two shapes is used for the alignment of multiple shapes.

Although the following methods in principle are applicable to data of any dimension, we stick to the 3D case. A single shape in our notation is thus a $3 \times N_x$ matrix. We denote the two matrices representing the two shapes which are to be

aligned $\mathbf{X}$ and $\mathbf{Y}$. Note that both shapes include the same number of points and thus $N_x = N_y$.

In matrix notation we can now write down the sum of squared Euclidean distances between pairs of correlating points:

$$d(\mathbf{X}, \mathbf{Y}) = \sum_{i=1}^{N_x} \|\boldsymbol{x}_i - \boldsymbol{y}_i\|^2. \tag{3.27}$$

In the above equation d describes the difference between the two shapes $\mathbf{X}$ and $\mathbf{Y}$ and is also referred to as Procrustes distance of two shapes. The goal of aligning two shapes can be identified as minimizing d with respect to translation, scale and rotation. We can thus formulate the problem as a minimization problem

$$\sum_{i=1}^{N_x} \|\boldsymbol{x}_i - T(\boldsymbol{y}_i)\|^2 \rightarrow \min \tag{3.28}$$

where T refers to a transformation consisting of translation, scaling and rotation.

It can be shown [Horn, 1987] that the optimum translation vector is equal to the difference vector between the centroids $\boldsymbol{\mu}_x$ and $\boldsymbol{\mu}_y$ of the two shapes. The centroids are the mean vectors of all column vectors of the considered matrix:

$$\boldsymbol{\mu}_x = \frac{1}{N_x} \sum_{i=1}^{N_x} \boldsymbol{x}_i, \qquad \boldsymbol{\mu}_y = \frac{1}{N_y} \sum_{i=1}^{N_y} \boldsymbol{y}_i. \tag{3.29}$$

So the first step in aligning $\mathbf{X}$ and $\mathbf{Y}$ is to align their centroids $\boldsymbol{\mu}_x$ and $\boldsymbol{\mu}_y$. This can be achieved by shifting the shapes such that their centroids align with the origin of the coordinate system.

The shift to the origin of the coordinate system is the same as replacing $\mathbf{X}$ and $\mathbf{Y}$ with the zero-mean matrices $\hat{\mathbf{X}}$ and $\hat{\mathbf{Y}}$. For finding the optimum scale and rotation we thus consider only this zero-mean data.

The optimum scaling factor s that has to be applied to $\mathbf{Y}$ such that both shapes become the same size can be shown to be [Horn, 1987]:

$$s = \sqrt{\frac{\sum_{i=1}^{N_x} \|\hat{\boldsymbol{x}}_i\|^2}{\sum_{i=1}^{N_y} \|\hat{\boldsymbol{y}}_i\|^2}}. \tag{3.30}$$

Finally the optimum rotation has to be found. For this we consider the data from which translation and scale are already filtered out. We represent this data with two matrices $\mathbf{X}'$ consisting of the column vectors

$$\boldsymbol{x}'_i = \hat{\boldsymbol{x}}_i \tag{3.31}$$

and $\mathbf{Y}'$ consisting of the column vectors

$$\boldsymbol{y}'_i = s\hat{\boldsymbol{y}}_i. \tag{3.32}$$

Representing rotation by matrix $\mathbf{R}$ the following equation can be introduced

$$\mathbf{E} = \mathbf{R}\mathbf{Y}' - \mathbf{X}' \tag{3.33}$$

where $\mathbf{E}$ is the difference after rotation of $\mathbf{Y}'$. The trace of $\mathbf{E}\mathbf{E}^T$ is equal to the Procrustes distance d as introduced in equation 3.27:

$$\mathrm{tr}(\mathbf{E}\mathbf{E}^T) = d(\mathbf{X}', \mathbf{R}\mathbf{Y}'). \tag{3.34}$$

Finding the optimal rotation is a matter of minimizing the above trace:

$$\mathrm{tr}(\mathbf{E}\mathbf{E}^T) \rightarrow \min . \tag{3.35}$$

Note that so far we have not made that $\mathbf{R}$ really is a rotation matrix. This is done by claiming that $\mathbf{R}$ is orthonormal or

$$\mathbf{R}\mathbf{R}^T = \mathbf{R}^T\mathbf{R} = \mathbf{I} \tag{3.36}$$

where $\mathbf{I}$ is the identity matrix.

Using the method of Lagrange multipliers [Edwards, 1994] the minimization problem stated in equation 3.35 with the side condition stated in equation 3.36 can be solved analytically. Using Singular Value Decomposition (SVD) of matrix $\mathbf{Y}'\mathbf{X}' = \mathbf{U}\mathbf{D}\mathbf{V}^T$ it can be shown that $\mathbf{U}\mathbf{V}^T$ is the rotation matrix optimally aligning $\mathbf{Y}'$ to $\mathbf{X}'$. For more details about the mathematical background we refer to Schoenemann (1966).

In order to avoid the side condition stated in equation 3.36 an alternative approach can be used to find the optimal rotation. Using unit quaternions to represent rotations the explicit claim that the transformation matrix has to be orthonormal can be omitted [Coutsias *et al.*, 2004; Horn, 1987].

We resume that there exist analytical methods to calculate the transformation for optimally aligning a shape $\mathbf{Y}$ to another shape $\mathbf{X}$. In the following discussion we denote such a transformation $\tau(\mathbf{X}, \mathbf{Y})$.

3.2.4 Aligning Multiple Shapes

In order to build a statistical shape model, first all shapes $\mathbf{X}_i$ of a training set have to be aligned. We outline an iterative algorithm to perform this task [Besl and McKay, 1992]. The basic idea is to iteratively align all shapes to some mean shape.

At the beginning any shape is chosen to be the initial estimation of the mean shape $\overline{\mathbf{X}}$. All the remaining shapes are then aligned with the initial mean shape. This can be done by following the approach from above for alignment of two shapes.

After the first alignment a new mean shape $\overline{\mathbf{X}}$ is calculated as the arithmetic mean of corresponding coordinates of all shapes:

$$\overline{\mathbf{X}} = \frac{1}{N_t} \sum_{i=1}^{N_t} \mathbf{X}_i. \tag{3.37}$$

where N_t denotes the number of shapes $\mathbf{X}_i$ in the training set. Again all shapes are aligned to this new mean shape. One result of the alignment is that the mean shape changes. Therefore a new mean shape has to be calculated in each iteration according to equation 3.37. The process of alignment and calculation of a new mean shape is iterated until the distance between the current mean shape and the mean shape from the last iteration falls below some threshold.

$\overline{\mathbf{X}}$ as defined by equation 3.37 is also referred to as Procrustes mean shape [Stegmann, 2000; Stegmann and Gomez, 2002]. The new $\overline{\mathbf{X}}$ only is a valid shape instance if the considered shapes are already more or less aligned. This is why for the first alignment any shape is chosen to be the mean shape. This results in a good initial alignment after the first iteration. The following iterations then produce plausible Procrustes mean shapes.

We can now formulate the complete iterative closest point (ICP) algorithm to align all shapes $\mathbf{X}_i$. In the following we denote transformations as t. Accordingly $t(\mathbf{X})$ is the point set $\mathbf{X}$ transformed by t. A concatenation of two transformations is written $t_c = t_a \cdot t_b$. The application of t_c is then equal to an application of t_a followed by an application of t_b: $t_c(\mathbf{X}) = t_b(t_a(\mathbf{X}))$. The identity transformation is denoted t_{id}.

For each shape a transformation t_i is kept that represents the complete transformation of the original shape's position. In algorithm 1 again d denotes the Procrustes distance of two shapes and d_0 denotes a threshold for the termination criterion. When the algorithm terminates, t_i stores the complete transformation for shape $\mathbf{X}_i$. Applying each transformation q_i to its associated shape $\mathbf{X}_i$ aligns all shapes.

1: **for** each $i = 1 \ldots N_t$ **do**
2: $t_i \leftarrow t_{id}$
3: **end for**
4: Select any shape $\mathbf{X}_i$ to be the initial mean shape $\overline{\mathbf{X}}_{new} \leftarrow \mathbf{X}_i$
5: **repeat**
6: $\overline{\mathbf{X}} \leftarrow \overline{\mathbf{X}}_{new}$
7: **for** each shape $i = 1 \ldots N_t$ **do**
8: Calculate the optimal registration $t^* = \tau(\overline{\mathbf{X}}, t_i(\mathbf{X}_i))$
9: Append the current registration update t^* to the shape's transformation: $t_i \leftarrow t_i \cdot t^*$
10: **end for**
11: Recalculate the Procrustes mean shape $\overline{\mathbf{X}}_{new}$ from all $t_i(\mathbf{X}_i)$
12: **until** $d(\overline{\mathbf{X}}, \overline{\mathbf{X}}_{new}) < d_0$

Algorithm 1: Iterative closest point (ICP) alignment of multiple shapes.

3.2.5 Shape Model Formulation

We have discussed alignment of shapes and principal component analysis of data in general. Building a shape model is not much more than combining both methods

appropriately. The input of the whole data analysis is a set of shapes with corresponding points. In the terminology of AAMs a so called point distribution model (PDM) is derived from these shapes. This PDM is nothing else but a compact description of possible shapes similar to the ones in the training set.

After having solved the problem of aligning individual shapes the actual statistical analysis can be carried out in order to build a pure shape model. In the following we will represent shapes not as matrices as we have done previously. From now on we represent shapes as vectors $\boldsymbol{s}$. A shape vector includes all coordinates constituting a shape. It does not matter in which order the coordinates are included in a shape vector or of which dimensionality the landmark points are. The only thing that has to be taken care of is that there is a 1-to-1 correspondence between the i-th coordinate of one shape and the i-th coordinate of another shape. We use the convention that a shape vector is represented by $\boldsymbol{s} = (x_1, y_1, z_1, x_2, y_2, z_2, \ldots, x_{N_x}, y_{N_x}, z_{N_x})^T$ where x_i, y_i, z_i are the coordinates of the i-th landmark point. The abstraction we use is to regard each shape as a simple vector of features where each feature is one coordinate of a landmark point. We consider all shapes already being aligned by an ICP algorithm as described above.

The next step is to compile the individual (aligned) shapes $\boldsymbol{s}_i$ of the training set to a matrix $\mathbf{S}$ where $\boldsymbol{s}_i$ is the i-th column vector. The last step in building the shape model then is to calculate the principal component analysis (PCA) of matrix $\mathbf{S}$. Similarly to equation 3.21 we get the model representation of the shape data:

$$\boldsymbol{s} = \overline{\boldsymbol{s}} + \boldsymbol{\Phi}_{s,k}\boldsymbol{p}_s. \tag{3.38}$$

Together with the eigenvalues λ_i equation 3.38 constitutes a shape model. The elements $p_{s,i}$ of $\boldsymbol{p}_s$ are called the modes of the shape model. k is the number of eigenvectors used in the model. The greater k, the more of the shape variation included in the training set can be expressed by the model.

Let N_s be the number of shape features and N_t the number of training sets. In 3D and for N_x shape points there are $N_s = 3N_x$ shape features. Equation 3.38 tells us how a shape instance can be calculated from a vector of modes $\boldsymbol{p}_s$. It is an important property of PCA to reduce dimensionality. This means that in general $\boldsymbol{p}_s$ includes fewer elements than $\boldsymbol{s}$. Especially if there are many more shape features than there are elements in the training set ($N_s \gg N_t$), the dimensionality is reduced reasonably. In fact the number of modes (denoted k) cannot be greater than the number of training sets minus 1 ($k \leq N_t - 1$). Then k also is much smaller than N_s ($N_s \gg N_t > k$).

In a realistic setting we used 1500 shape features and 30 training sets. This means a vector including a maximum of 29 modes is needed to represent all shapes in the training set each one including 1500 elements. Note that the model allows us to control variances in large data vectors with a relatively small number of modes.

3.3 Texture Model

Similarly to the shape model, a texture model can be formulated. The data analyzed for building a texture model is usually represented in form of simple gray

values. Although there exist approaches using more complex color information [Stegmann and Larsen, 2003], we will consider only simple gray valued textures in the following discussion. We do this without restrictions since the medical MRI data considered here does not include any higher bandwidth color information.

3.3.1 Obtaining Texture

The first step in building a texture model is to find out which gray values are important. This has to be done since the complete image data usually does not only include gray values of the object for which we want to build the model. Also texture information resides in the image which comes from things which are distinct from our object of interest. Cardiac MRI data, for example, also shows parts of other organs, not only of the heart. For building a model of the left ventricle one only wants to use the according texture.

However, spatial information already has to be available in form of annotations for the shape model we have already outlined. Obviously this geometrical information is a perfect basis for deciding which gray values we should use in the texture model. The standard solution is to calculate a Delaunay triangulation of the landmark points and to use all gray values lying inside the resulting simplices (triangles in 2D, tetrahedra in 3D, ...). In other words to obtain the texture used for model building we run through all samples of the grid representing the image data and store the according gray values.

An important property of the texture model is that spatial variation is filtered out completely. This is achieved by placing a regular grid over the mean shape. Only the nodes of the grid residing inside the mean shape are taken. The mean shape is now transformed such that it becomes equal to the considered data set's shape. The grid nodes introduced in the mean shape space are deformed accordingly. The texture can then be resampled at exactly the positions where the transformed grid nodes reside.

The remaining question is how to deform the grid nodes according to the shape points. The simplest solution is to use a Delaunay triangulation of the mean shape as mentioned above. This subdivides the model into simplices. Linearity is assumed for the deformation of each simplex. This means that the barycentric coordinates of a point inside a simplex are constant under the linear deformation of the shape.

The described piecewise linear warping is relatively easy to implement and is thus the standard solution for warping AAMs. However it can lead to some problems. The kinks that appear at the edges of the warped simplices make smooth contours appear not smooth under the deformation. As an improvement thin plate splines [Bookstein, 1991] could be used for warping, which in general generate smoother deformations. Another problem that is not solved using thin plate splines are foldings of space as depicted in figure 3.4.

Recently the use of diffeomorphic deformations for statistical shape models has been discussed, which avoids the mentioned folding [Cootes *et al.*, 2004]. However, such more elaborated deformation strategies increase the computational effort

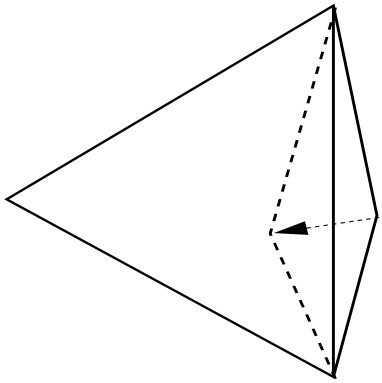


Figure 3.4: Overlapping due to folding. A typical fold as it can appear when applying an affine warp. The corner of the (2D) triangle on the right is transformed into the (2D) triangle on the left. The result is that both triangles overlap.

especially when used for higher dimensional data. Since usually only very small deformations are considered, the mentioned folds and kinks are often ignored in AAM implementations.

In spite of the mentioned problems connected to affine warping we follow this approach in the present work. In the mean shape space the barycentric coordinates of regularly spaced grid nodes are determined. If the texture of a given data set has to be extracted the data set's shape is taken and is made the new basis for the previously calculated barycentric coordinates. In other words we do not change the barycentric coordinates but their basis points. This leads to the correct Cartesian coordinates of sample nodes in the considered data set's shape space. At these transformed nodes the volume data is resampled and stored in the data set's texture vector. This proceeding is illustrated in figure 3.5.

Note that with the above procedure for each data set exactly the same number of texture samples is calculated. Also the correlation of individual texture samples is maintained between different data sets. For example a texture sample that resides at the inner border of the left ventricle in one data set also resides at the corresponding position for another data set thanks to warping of the resampling grid.

So far we have assumed that all the texture is taken from evenly spaced samples lying in the triangles of a Delaunay triangulation of the mean shape. This is a very simple method to obtain texture samples. The great drawback is that a Delaunay triangulation covers the complete convex hull of all points for which it is built. In other words if the mean shape of an object is not convex the Delaunay triangulation includes both the interior and exterior simplices.

For a 2D shape of a human hand, for example, the Delaunay triangulation also creates triangles in between the fingers. For the texture model this means that texture of the background is included. Such texture from outside the actual object usually is not wanted. Simplices lying outside the mean shape have to be identified and removed before texture is resampled. In the simplest case this can be done manually.

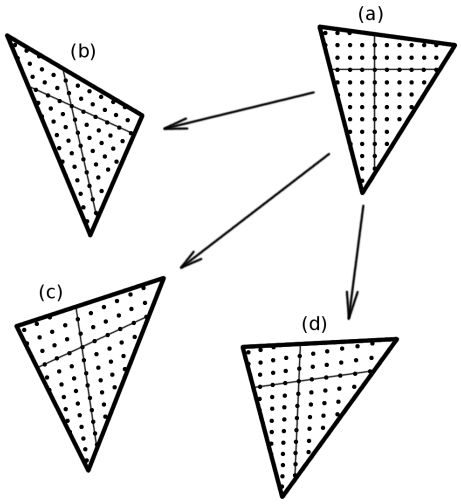


Figure 3.5: Sampling in shape-normalized space. The mask for resampling texture is determined in mean shape space (a). Then this mask is warped to the individual shapes of the examples in the training set (b), (c) and (d). After this the texture values are resampled in the warped nodes.

3.3.2 Intensity Normalization

Usually there are variations of gray value intensities included in most image data. These variations can be considered acquisition artifacts. Since the goal is to build models not including such unwanted information, these variations have to be filtered out. In the following we assume that for each data set a vector of texture values has already been extracted.

A simple method is to subtract the mean value $\overline{g}_j$ of all intensity values in the texture vector belonging to data set j. Assuming that there are N_g samples $g_{j,1}, g_{j,2}, \ldots, g_{j,N_g}$ concatenated to a single texture vector $\boldsymbol{g}_j$, the according zero-mean texture samples $\hat{g}_{j,i}$ can be calculated by

$$\hat{g}_{j,i} = g_{j,i} - \overline{g}_j, \quad \overline{g}_j = \frac{1}{N_g} \sum_{i=1}^{N_g} g_{j,i} \tag{3.39}$$

where $\overline{g}_j$ denotes the average gray value intensity of a single data set j. Furthermore the average mean value over all data sets can be calculated

$$\overline{g} = \frac{1}{N_t} \sum_{j=0}^{N_t} \overline{g}_j, \tag{3.40}$$

with N_t being the number of data sets.

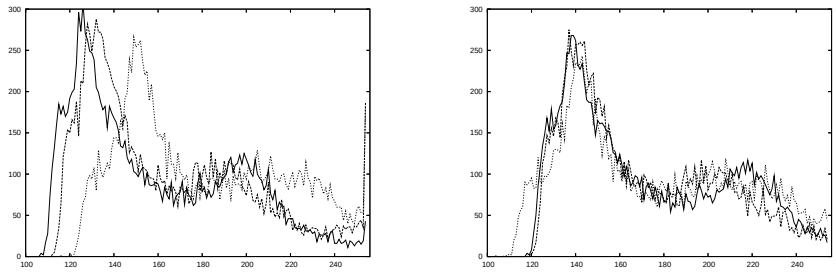


Figure 3.6: Normalization of intensities of a given texture vector. The figure on the left shows histograms of distributions of texture intensities of three images. On the right both, mean and standard deviation, are unified. This makes the distributions of intensities for all three data sets fit together much better. The intention of texture normalization is to filter out global intensity variations before a model is built.

Next we would like to unify the different variances in the training set. This can be done by first calculating the average variance $\overline{\alpha}$ of all data sets:

$$\overline{\alpha} = \frac{1}{N_t} \sum_{j=0}^{N_t} \alpha_j, \quad \alpha_j = \frac{1}{N_g - 1} \sum_{i=1}^{N_g} (\hat{g}_{j,i})^2. \tag{3.41}$$

Here α_j denotes the variance of the j-th data set and N_t again is the number of data sets. Finally the normalization of an individual texture vector is achieved by dividing each intensity value shifted to zero mean by the data set's standard deviation $\sqrt{\alpha_j}$, multiplying it by the average standard deviation $\sqrt{\overline{\alpha}}$ and shifting it to the average mean $\overline{g}$:

$$g'_{j,i} = \frac{g_{j,i} - \overline{g}_j}{\sqrt{\alpha_j}} \sqrt{\overline{\alpha}} + \overline{g} \tag{3.42}$$

where $g'_{j,i}$ is the normalized i-th sample of the j-th texture vector. The result of such a normalization is illustrated in figure 3.6.

When taking a look at slices of volumetric medical data sets one can see that there are great differences in intensities of adjacent slices. This texture-related problem is similar to the geometry-related problem of motion dependent displacement of slices. The sometimes considerable differences of intensities between slices induce additional unwanted variations that one wants to keep out of the model. One approach to solve this problem is to normalize intensities of individual slices. For example, intensity distributions of individual slices could all be shifted to the same mean and stretched to the same variance as described above. Note that for this strategy all voxel intensities of a complete slice are considered. This makes a big difference to the normalization described above where only texture from inside the model is considered.

Removing inter-slice variations in the data can be regarded as a preprocessing of the data. If such a preprocessing is introduced the fact that also background is considered for the normalization has to be kept in mind. It makes the strategy very

problematic if the objects in the background, which are not included in the model, vary strongly from data set to data set. Then unwanted variations from outside the model disturb the model's texture. For many medical data sets however this problem can be neglected since the surroundings of organs usually do not vary so much.

Intensity-adjustment of slices actually is some kind of preprocessing of the original data. It takes place before any real model building step is carried out. Refer to chapter 6 for more details on texture preparation for our cardiac ventricle data sets.

3.3.3 Texture Model Formulation

Once texture values are resampled and normalized the texture model itself can be built. This is done in a very similar way to building the shape model. The texture vector extracted from data set j is represented as a vector $\boldsymbol{g}_j$ of normalized gray values. All texture information can be represented by a matrix $\mathbf{G}$ whose column vectors are denoted $\boldsymbol{g}_j$. Finally the PCA of $\mathbf{G}$ can be calculated resulting in the texture model:

$$\boldsymbol{g} = \overline{\boldsymbol{g}} + \boldsymbol{\Phi}_{g,k}\boldsymbol{p}_g. \tag{3.43}$$

As for the shape model we can now represent large textures $\boldsymbol{g}$ with the k parameters or modes $p_{g,i}$ included in vector $\boldsymbol{p}_g$.

3.3.4 Border AAMs

In general, only texture from inside the shape model is taken for the texture model. Sometimes, however, it can also be useful to include texture samples from outside the model shape. This is done because the outer border of the shape often corresponds to an edge in the image data. Taking only texture from inside the model means that the model has only information about texture from one side of the edge. Thus the model has no information about the edge itself. To include the edge information in the model, a thin border around the shape is considered to include additional texture samples in the model.

This approach is also known as "Border AAMs" [Stegmann, 2000]. It is especially interesting for medical applications since the variances of adjacent organs correlate. This justifies the inclusion of surrounding texture. This approach avoids the shrinking of the model away from the object's outer border when matching the model to unseen data.

3.4 Combined Model

The last step in building a complete AAM is to merge the shape model given by equation 3.38 and the texture model represented by equation 3.43 into a combined model. The combined model then describes the complete statistical variations in both shape and texture.

The first step in building the combined model is to simply concatenate the vectors representing shape and texture:

$$\boldsymbol{x} = \begin{bmatrix} \boldsymbol{s} \\ \boldsymbol{g} \end{bmatrix}. \tag{3.44}$$

The above concatenation is problematic because shape features which represent coordinates and texture features which represent gray values are put together. The magnitude of both types of data in general varies and the following statistical analysis of the combined data leads to bad results. This is why a weight matrix $\mathbf{W}_s$ for model parameters of the shape model is introduced. The shape model is reformulated,

$$\boldsymbol{s} = \overline{\boldsymbol{s}} + \boldsymbol{\Phi}_s \boldsymbol{p}_s = \overline{\boldsymbol{s}} + \boldsymbol{\Phi}_s \mathbf{W}_s^{-1} \underbrace{\boldsymbol{p}_s \mathbf{W}_s}_{\boldsymbol{p}'_s} = \overline{\boldsymbol{s}} + \boldsymbol{\Phi}_s \mathbf{W}_s^{-1} \boldsymbol{p}'_s \tag{3.45}$$

where $\boldsymbol{p}'_s$ denotes the scaled model parameters. In the following the scaled shape model parameters are considered instead of the original shape model parameters.

An open question is how the weight matrix $\mathbf{W}_s$ is determined. A common method is to calculate the fraction of the sum of eigenvalues $\lambda_{g,i}$ of the texture model and the sum of eigenvalues $\lambda_{s,i}$ of the shape model:

$$r = \frac{\lambda_g}{\lambda_s}, \quad \lambda_s = \sum \lambda_{s,i}, \quad \lambda_g = \sum \lambda_{g,i}. \tag{3.46}$$

The weight matrix $\mathbf{W}_s$ for the shape model parameters is now chosen to be the diagonal matrix whose elements are equal to r:

$$\mathbf{W}_s = \begin{bmatrix} r & & 0 \\ & \ddots & \\ 0 & & r \end{bmatrix}. \tag{3.47}$$

With the weighted version of the shape model parameters we are now able to formulate an initial version of the combined model:

$$\boldsymbol{x} = \begin{bmatrix} \boldsymbol{s} \\ \boldsymbol{g} \end{bmatrix} = \begin{bmatrix} \overline{\boldsymbol{s}} \\ \overline{\boldsymbol{g}} \end{bmatrix} + \begin{bmatrix} \boldsymbol{\Phi}_s \mathbf{W}_s^{-1} & 0 \\ 0 & \boldsymbol{\Phi}_g \end{bmatrix} \begin{bmatrix} \boldsymbol{p}'_s \\ \boldsymbol{p}_g \end{bmatrix}. \tag{3.48}$$

Using equations 3.38 and 3.43 a model instance can be built according to some given parameters. For building the combined model it is necessary to invert this calculation. Namely, the parameters have to be determined given the model instances. The equation of the shape model can be inverted according to the following derivation:

$$\begin{aligned} \boldsymbol{s}_i &= \overline{\boldsymbol{s}} + \boldsymbol{\Phi}_s \boldsymbol{p}_{s,i} \\ \boldsymbol{s}_i - \overline{\boldsymbol{s}} &= \boldsymbol{\Phi}_s \boldsymbol{p}_{s,i} \\ \boldsymbol{\Phi}_s^T (\boldsymbol{s}_i - \overline{\boldsymbol{s}}) &= \boldsymbol{p}_{s,i} \end{aligned} \tag{3.49}$$

where $\boldsymbol{s}_i$ denotes the shape and $\boldsymbol{p}_{s,i}$ the parameters of data set i. Note that $\boldsymbol{\Phi}_s^{-1} = \boldsymbol{\Phi}_s^T$ since the columns of $\boldsymbol{\Phi}_s$ are orthonormal eigenvectors. For the weighted shape model parameters we get:

$$\mathbf{W}_s \boldsymbol{\Phi}_s^T (\boldsymbol{s}_i - \overline{\boldsymbol{s}}) = \boldsymbol{p}'_{s,i}. \tag{3.50}$$

Analogously an inversion of the texture model gives:

$$\boldsymbol{\Phi}_g^T(\boldsymbol{g}_i - \overline{\boldsymbol{g}}) = \boldsymbol{p}_{g,i}. \tag{3.51}$$

Using the two previous formulas one can calculate the corresponding vector of model parameters for any training data set i:

$$\boldsymbol{p}_i = \begin{bmatrix} \boldsymbol{p}'_{s,i} \\ \boldsymbol{p}_{g,i} \end{bmatrix}. \tag{3.52}$$

Each of these vectors contains both the parameters for the shape model and the parameters for the texture model. Each $\boldsymbol{p}_i$ encodes shape and texture – the complete appearance – of one example in the training set. All vectors $\boldsymbol{p}_i$ are assembled into matrix $\mathbf{P}$ as column vectors. We can now apply PCA to $\mathbf{P}$ in order to get a model-like representation of its column vectors:

$$\boldsymbol{p} = \begin{bmatrix} \boldsymbol{p}'_s \\ \boldsymbol{p}_g \end{bmatrix} = \overline{\boldsymbol{p}} + \boldsymbol{\Phi}_p \boldsymbol{p}_c. \tag{3.53}$$

The trick is now to replace vector $(\boldsymbol{p}'_s|\boldsymbol{p}_g)^T$ in equation 3.48 with the right part of equation 3.53:

$$\boldsymbol{x} = \begin{bmatrix} \boldsymbol{s} \\ \boldsymbol{g} \end{bmatrix} = \begin{bmatrix} \overline{\boldsymbol{s}} \\ \overline{\boldsymbol{g}} \end{bmatrix} + \begin{bmatrix} \boldsymbol{\Phi}_s \mathbf{W}_s^{-1} & 0 \\ 0 & \boldsymbol{\Phi}_g \end{bmatrix} (\overline{\boldsymbol{p}} + \boldsymbol{\Phi}_p \boldsymbol{p}_c). \tag{3.54}$$

The parameters of the combined model are represented by vector $\boldsymbol{p}_c$. The elements of this vector do not correspond to shape or texture values separately but influence both. Equation 3.54 however is not the final formulation of the combined model. Since the model parameters of shape and texture have values evenly distributed around zero we can assume that $\overline{\boldsymbol{p}}$, the mean of parameters in the models of shape and texture models is the null-vector: $\overline{\boldsymbol{p}} = (0, 0, \ldots, 0)^T$. This simplifies equation 3.54 to

$$\boldsymbol{x} = \begin{bmatrix} \boldsymbol{s} \\ \boldsymbol{g} \end{bmatrix} = \begin{bmatrix} \overline{\boldsymbol{s}} \\ \overline{\boldsymbol{g}} \end{bmatrix} + \underbrace{\begin{bmatrix} \boldsymbol{\Phi}_s \mathbf{W}_s^{-1} & 0 \\ 0 & \boldsymbol{\Phi}_g \end{bmatrix} \boldsymbol{\Phi}_p}_{\mathbf{C}} \boldsymbol{p}_c \tag{3.55}$$

or simply:

$$\boldsymbol{c} = \overline{\boldsymbol{c}} + \mathbf{C}\boldsymbol{p}_c \tag{3.56}$$

3.5 Valid Range for Model Parameters

Equation 3.56 is the final representation of model instances. Each of the elements in $\boldsymbol{p}_c$ is a parameter or mode of the model and usually is considered to have a value of $\pm 3\sigma_i$ where σ is the standard deviation. We know that the eigenvalues λ_i calculated in the PCA of $\mathbf{P}$ are the maximum likelihood estimates for the variances of the combined model's parameters. Accordingly the standard deviation for each model parameter is the square root of its associated eigenvalue: $\sigma_i = \sqrt{\lambda_i}$.

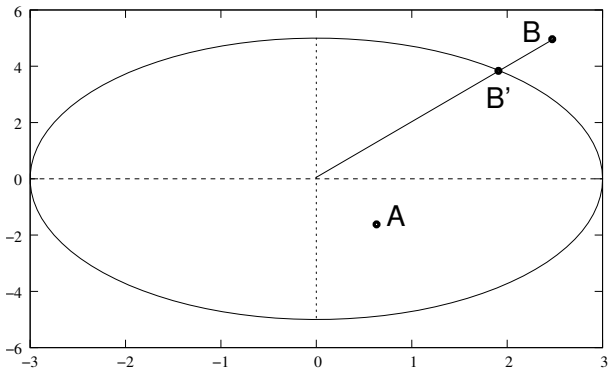


Figure 3.7: The hyper-ellipse defining the valid model instance. Point A denotes a valid model instance. B represents an invalid instance which can be corrected to B' which is the closest valid instance to B.

By restricting the individual model parameters to three times their standard deviation the specificity of the model can be assured. In literature this is described as the usual restriction so that all derived model instances are likely to represent valid instances of the modelled object [Cootes and Taylor, 2000; Stegmann, 2000]. Further it can be stated that model instances whose parameters all differ a lot from zero are rather implausible. Thus often an even stronger restriction is made to the model parameters. The Mahalanobis distance between the mean (where all modes are zero) and some model instance is calculated. If this distance exceeds some threshold the model instance is declared invalid [Cootes and Taylor, 2000; Stegmann, 2000]. The Mahalanobis distance is frequently used in statistics to measure the distance of two points $\boldsymbol{a}$ and $\boldsymbol{b}$. It is defined as

$$d_M(\boldsymbol{a}, \boldsymbol{b}) = \sqrt{(\boldsymbol{a} - \boldsymbol{b})^T \mathbf{S}^{-1} (\boldsymbol{a} - \boldsymbol{b})} \tag{3.57}$$

where $\mathbf{S}$ denotes the covariance matrix. Here we only want to calculate the distance of some instance to the mean. Since the mean has all parameters set to zero we can consider $\boldsymbol{b} = 0$. Further the covariance matrix calculated by the PCA is diagonalized and includes the eigenvalues as its diagonal elements. Thus equation 3.57 can be rewritten as:

$$d_M(\boldsymbol{a}, 0) = \sqrt{\boldsymbol{a}^T \boldsymbol{\Lambda}^{-1} \boldsymbol{a}}. \tag{3.58}$$

In other words the criterion for a model instance to be valid is

$$\sqrt{\boldsymbol{p}_c^T \boldsymbol{\Lambda}^{-1} \boldsymbol{p}_c} = \sqrt{\sum_{i=0}^{N} \frac{\boldsymbol{p}_{ci}^2}{\sigma_i^2}} \leq D_m \tag{3.59}$$

with N being the number of eigenvalues not equal to zero and D_m a threshold. In order to conform with the criterion that a parameter value should not exceed $\pm 3\sqrt{\lambda_i}$ the threshold D_m is chosen to be 3. If only one single parameter is not equal to zero the validity criterion becomes

$$\frac{\sqrt{\boldsymbol{p}_{ci}^2}}{\sigma_i} \leq 3. \tag{3.60}$$

If a set of parameters is not valid,

$$\sqrt{\sum_{i=0}^{N} \frac{\boldsymbol{p}_{ci}^2}{\sigma_i^2}} = F > D_m, \tag{3.61}$$

a correction has to be done to the model parameters in order to make the model plausible again. This can be achieved by scaling each individual parameter by D_m/F:

$$\boldsymbol{p}_{c,i} \leftarrow \boldsymbol{p}_{c,i} \cdot \frac{D_m}{F} \tag{3.62}$$

The space delimited by the Mahalanobis distance is a hyper-ellipse. The valid model instances only lie inside this ellipse. This is illustrated in figure 3.7.

3.6 Extended Model Formulations

The standard AAMs assume that each shape instance can be modelled as a simple linear combination of a basis of shapes and textures as found in the training set. In this section we will take a closer look at model formulation and outline some recently proposed techniques that try to extend model representation and make it more robust.

We remind the reader of the main problem of efficient model representation: The model has to be general enough to be able to represent all possible instances and at the same time it has to be specific enough to avoid invalid instances. In the following we present some approaches which try to resolve this antagonism of generality and specifity.

3.6.1 Non-Linearity and Non-Continuity

One critical aspect of AAMs is that linearity of shape and texture is assumed. The question is what will happen if this assumption fails and the real object variances are not linear. Figure 3.8 shows possible distributions of model parameters for highly non-linear training sets. Such non-linearities for example appear if the shape of an object is bending over different instances in the training set.

Another case where the assumption of linearity fails is if there is some feature that can have two or more distinct values where an interpolation in between does not make sense. For example imagine a training set for a face model including images of people some wearing glasses and some wearing no glasses. The standard AAM would then be able to create instances with “a little bit” glasses. This does obviously not make much sense and so we can state that these instances should be identified invalid and should be avoided during the model matching process. Although a lot of literature has been published about non-continuous and non-linear modelling [Roweis and Saul, 2000; Tenenbaum *et al.*, 2000] we shall here only review some of the proposed strategies.

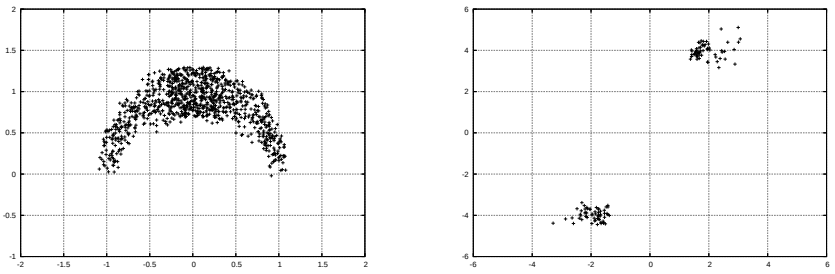


Figure 3.8: This figure shows two examples where the linearity assumption is not appropriate: non-linearity (left) and non-continuity (right) in the model parameters of a training set.

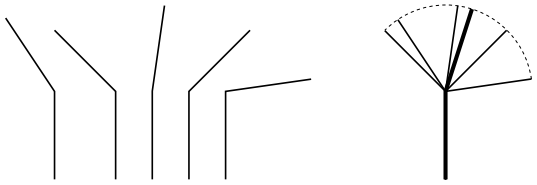


Figure 3.9: A shape that is strongly bending over examples in the training set

3.6.2 Projection into Tangent Space

There are several techniques dealing with the problem of non-linearity. One approach [Cootes and Taylor, 1999; Cootes and Taylor, 2000; Stegmann and Gomez, 2002] uses a projection into tangent shape space of individual training shapes. Imagine a training set where strong bending is observed in the given examples. Figure 3.9 shows an example for such a shape that strongly bends over the training set. This variation can be explained by a single non-linear parameter. However, the use of PCA in such a case leads to two linear model parameters. The modelling of non-linear variations by multiple linear ones is typical of PCA. Not only the increase of the model's dimensionality is a problem. Also the specificity of the model suffers. In the following we discuss projection of shapes into tangent space. The intention is to transform non-linear variations into linear ones.

Often non-linearities are caused by the alignment of shapes. In section 3.2.3 we have discussed alignment of shapes. For this the shapes are scaled in their size such that the Procrustes distance is minimized. This scaling is not always optimal with respect to linearity of the positions of shape points.

In figure 3.10 the two arrows illustrate the positions of two shape points. The bold arrow represents the point's position in the mean shape and the other arrow illustrates the corresponding point of some specific training example. Pivoting of the shape point is indicated by the depicted arc. The figure illustrates that a projection of the point to the tangent of its mean shape transforms the non-linear variation into

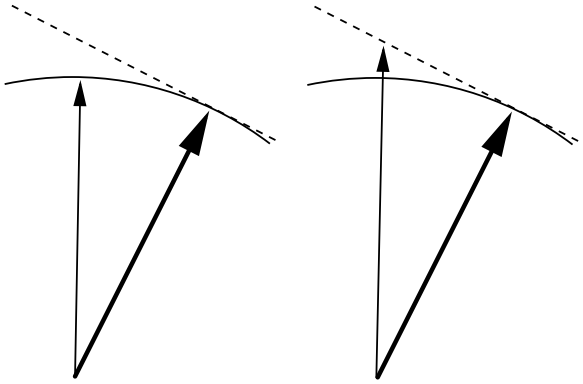


Figure 3.10: A shape which varies from the mean by pivoting (left) is transformed to tangent shape space (right) by scaling. The bold arrow represents the mean shape. The transformed shape is represented by the thin arrow [Stegmann and Gomez, 2002].

a linear one. A projection into tangent shape space can be accomplished by scaling individual shapes $\boldsymbol{s}_i$ by the factor $1/(\boldsymbol{s}_i \cdot \overline{\boldsymbol{s}})$.

However in the given example a relatively simple non-linearity appears. The method described above only works if the non-linearities approximately follow circular arcs around the centroid of shapes. For real-world shapes the situation is more complicated.

3.6.3 Using Polar Coordinates

To better handle non-linearities also the use of polar coordinates for individual points has been proposed [Heap and Hogg, 1996]. Similar to the use of tangent space in chapter 3.6.2 the assumption is made that the appearing non-linearities are approximately circular. The main idea of the hybrid representation of landmark points is that parts of a model can pivot relative to other parts. For example in a hand model the fingers might rotate relative to the palm. A hybrid model can be formulated that uses Cartesian coordinates for points lying upon the palm. Points upon the fingers are modelled using polar coordinates. The PCA is carried out on the mixture of Cartesian and polar coordinates. The model finally is capable of modelling bending of a finger with a linear change of the angle of the according polar coordinates.

The lamp depicted in figure 3.11 also is an object which can very well be described using the mentioned hybrid approach. In order to describe possible deformations of this lamp as indicated in the figure, point 5 for example has to be represented by polar coordinates. Point 4, which has to be given in Cartesian coordinates, serves as the origin for the polar coordinates of point 5. However point 4 is not the only reference point needed. Another point is needed because the angle in the polar coordinates has to be given relative to some straight line. Such a reference line has to pass through point 4. A second point has to be found to completely specify the line's orientation. Two properties of the second point are claimed:

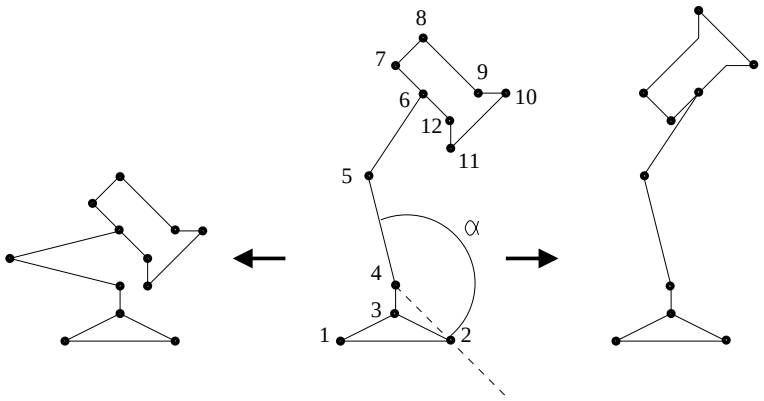


Figure 3.11: The shape of a lamp with pivoting parts [Heap and Hogg, 1996].

- The point has to be far away from the origin
- The point should move as less as possible relative to the origin.

In the above example point 2 would be feasible. Finally the position of point 5 can be determined completely by specifying its distance to point 4 and the angle between line $\overline{42}$ and line $\overline{45}$.

The outlined representation has the advantage that a hierarchy of point groups can be established. The points in the lowest layer have to be given in Cartesian coordinates. Point groups in a higher layer always have the possibility to rotate around a point from a lower layer. In this way it is possible that point 5 pivots around point 4, 6 pivots around 5 and 7, 8, 9, etc. pivot around 6.

For a small shape it is possible to manually determine which points shall be represented by which type of coordinates. For polar coordinates the corresponding reference points can also be manually determined. Anyway, for more complex shapes this method is too time-consuming. Two algorithms for automated assignment of the representation have been proposed [Heap and Hogg, 1996]. The first one allows only pivoting relative to existing points in the shape. The second one adds points to the shape as parts of the shape seem to pivot around this point.

To our knowledge the outlined hybrid method has not yet been extended to 3D. In principle this could be done straight forward. However, one has to consider that rotations in 3D are much more difficult to describe than in 2D because the moving points describe a sphere.

The hybrid method using Cartesian and polar coordinates is useful for objects whose parts are rigid and rotate relative to each other. However for medical data sets this usually is not the case. Non-linearities appearing in medical data appear rather due to more complicated bending combined with stretching and the like. We suggest that the outlined hybrid approach is not adequate to be applied in the medical domain.

3.6.4 Mixture Models

The previously described methods all use very specific approaches to handle very specific non-linearities in shape. The idea is to detect or assume a special kind of non-linearity in the model. Specific techniques are used to convert the non-linear relations into linear relations.

The Mixture Model approach [Cootes and Taylor, 1999] tries to increase specifity of a model without losing generality where it is needed. In the standard AAM approach the original shapes are projected into a lower-dimensional space by applying PCA. Then the domain of plausible shapes is identified as a hyper-ellipse around the mean shape defined by the Mahalanobis distance. For Mixture Models also the projection into a lower dimensional space is carried out. The difference is that the hyper-ellipse is not used to identify valid instances. Instead Mixture Models make use of Gaussian Kernels in order to describe the space of valid model instances.

The crucial question is to define what a “plausible shape” actually is. It must be stated that whether a shape is plausible or not often allows no strict answer. A shape may be “more plausible” or “not so plausible”. This makes it necessary to specify plausibility of shape in terms of probabilities. This is something only roughly considered in the standard AAM. Only global variances measured in the data set are considered.

Mixture models refine the treatment of statistical variances. The only information about plausible shapes that can be learned directly is that the training shapes can be assumed to be plausible. Of course this is not enough since there exist other plausible shapes similar to the ones in the training set. The simplest way to increase the domain of plausible shapes is to put Gaussian kernels around the training examples. This is done already in the space of reduced dimensionality as determined by PCA. The great advantage of this method is that the domain of valid model instances is described directly and this has the nice property that a lot of non-linearities or non-continuities in shape or appearance in general can be handled.

An important remaining question is how we can make sure that only valid instances are created during the model search. In case of the standard AAMs a truncation of parameter values is applied in order to project the invalid shape to the next valid shape (see figure 3.7). For Mixture Models in principle something similar is done. Suppose the plausibility for a specific shape becomes very small during search. Then this shape should somehow “flow” towards a close plausible state.

Cootes and Taylor (1999) suggest to move implausible shapes towards more plausible shapes using a gradient ascent approach. To make clear how this works in practice take a look at the training set of synthetic shapes in figure 3.12. Each shape consists of 28 points (there are three points lying on each line segment).

Figure 3.13 (left) shows an instance created by the model with the points disturbed by noise. Projecting the instance into the reduced space of parameters results in a shape as depicted in 3.13 (middle). This shape is already more plausible since the noise is reduced. However the size of the triangle still seems to be too large compared to the square. To solve this problem the model instance represented by a point in the parameter space is moved iteratively towards a point with higher “plausibil-

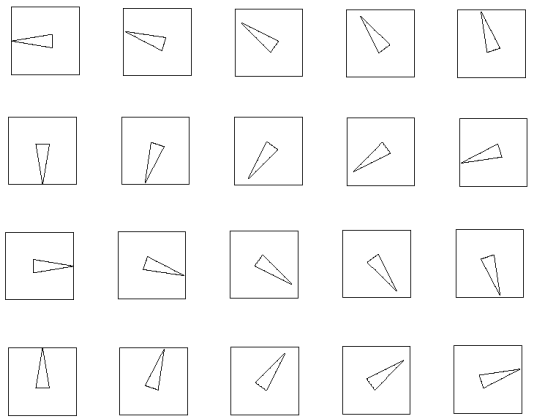


Figure 3.12: A synthetic training set including strong non-linearities. Figure from [Cootes and Taylor, 1999].

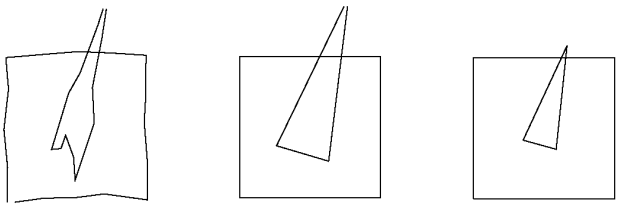


Figure 3.13: Increasing plausibility for generated shapes. Figure from [Cootes and Taylor, 1999].

ity". This means the Gaussian mixture function is ascended towards a maximum. The result of this can be seen in figure 3.13 (right).

3.6.5 Independent Component Analysis for AAMs

Principal Component Analysis which is used for standard AAMs makes the assumption that the individual principal components are orthogonal. This assumption allows an elegant calculation of principal components by analysis of the covariance matrix. However, recently another method has gained much interest. The so called Independent Component Analysis (ICA) [Hyvärinen, 1999; Üzümcü *et al.*, 2003] is a powerful alternative to PCA. In the following we discuss the main ideas behind ICA and show how it can be used for AAMs.

ICA versus PCA

We have already mentioned that the use of PCA is legitimate as long as the analyzed data follows a Gaussian distribution. This is the case because PCA mainly uses variances to describe the distribution of the data. However, if no Gaussian distribution

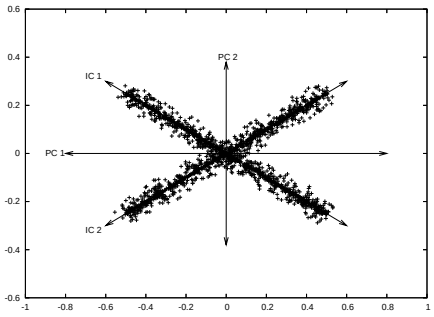


Figure 3.14: PCA and ICA on non-Gaussian data.

can be assumed the use of PCA is problematic. For example the point-cloud in figure 3.14 obviously includes two main directions. PCA performed on this data leads to principal components PC 1 and PC 2 which do not align at all with the two main directions in the data. This is the motivation for the use of higher-order techniques. For the example in figure 3.14 it would be nice to calculate the two independent components IC 1 and IC 2. For the depicted data IC 1 and IC 2 would definitely describe more meaningful and interesting properties.

In general the goal of reductions of dimensionality is to extract meaningful information from some high-dimensional data. As with PCA this is performed by some projection of the high-dimensional data to a low-dimensional space. In the simplest case the complex data is projected to a simple one-dimensional curve. The question is how to determine an optimal projection in order to derive highly meaningful low-dimensional information. Projection pursuit is a technique developed in statistics for finding "interesting" projections of multidimensional data. It has been argued [Huber, 1985] that projections which show a Gaussian distribution are the least interesting ones and that the most interesting projections are those which show the least Gaussian distribution. For example in figure 3.15 the first principal component of PCA is horizontal and a projection to this principal component shows a Gaussian distribution. However this does not contribute to the fact that the data forms two separate clusters. Applying a projection to the vertical axis, this information could be maintained. This is another case where PCA fails to extract an important feature.

PCA only considers variances which contribute to correlations in the data. The principal components are chosen such that correlations of different variables are minimized. For ICA not the correlations of different variables but statistical dependences are claimed to be minimized. Statistical independence is a much stronger requirement than uncorrelatedness. The latter is stated

$$E\{y_i y_j\} - E\{y_i\}E\{y_j\} = 0, \forall i \neq j \tag{3.63}$$

where E is the expectation value and y is the considered random vector. Independence on the other hand is stated by

$$E\{g_1(y_i)g_2(y_j)\} - E\{g_1(y_i)\}E\{g_2(y_j)\} = 0, \forall i \neq j \tag{3.64}$$

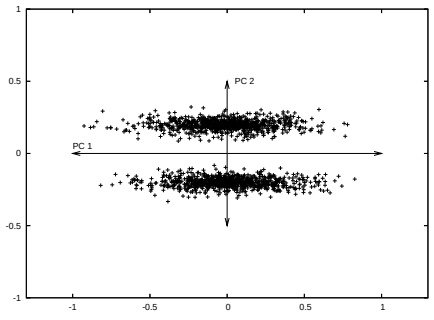


Figure 3.15: PCA on clustered non-Gaussian data.

for any functions g_1 and g_2. This is the case if the density function can be factorized:

$$f(y_1, \ldots, y_m) = f_1(y_1) f_2(y_2) \ldots f_m(y_m). \tag{3.65}$$

Definitions of linear independent component analysis

Although there exist also non-linear approaches to linear component analysis we shall in the following only consider the linear case. Also we consider the data to be mean-centered. This is no real restriction since every data set can be mean-centered by subtraction of the mean vector. According to Hyvärinen (1999) there exist three main definitions for ICA.

Definition 1: ICA of a random vector y consists of finding a linear transform $\boldsymbol{s} = \mathbf{W}\boldsymbol{y}$ so that the components s_i are as independent as possible, in the sense of maximizing some function $F(s_1, \ldots s_m)$ that measures independence.

This first definition is very general. The linear transform mentioned reminds very much of PCA where also a linear transformation is performed. The great difference is that for ICA no orthogonality of the independent components is claimed. Instead a cost function measuring independence is introduced. A maximization algorithm maximizing this function can be used in order to determine the independent components.

Definition 2: ICA of a random vector $\boldsymbol{y}$ consists of estimating the following generative model for the data:

$$\boldsymbol{y} = \mathbf{A}\boldsymbol{s} + \boldsymbol{n} \tag{3.66}$$

where the components s_i in the vector $\boldsymbol{s} = (s_1, \ldots, s_n)^T$ are assumed independent. Matrix $\mathbf{A}$ is a constant "mixing" matrix and $\boldsymbol{n}$ is a vector representing noise.

Definition 2 represents ICA as the problem of finding latent variables in the model. The problem with this definition is that the noise vector $\boldsymbol{n}$ makes it hard to handle. This is why in most cases the third definition of ICA is used.

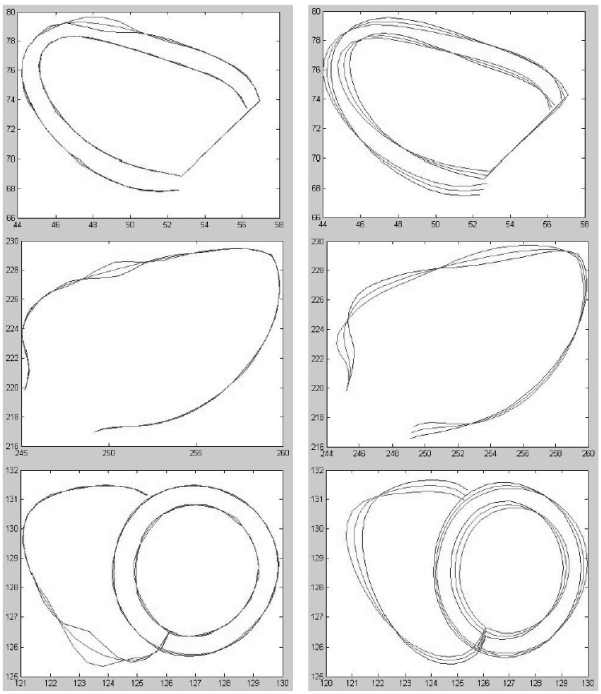


Figure 3.16: Comparison of variation in ICA (left) and PCA (right). The shapes are cross sections of the heart ventricles. Figure from [Üzümcü *et al.*, 2003].

Definition 3: ICA of a random vector $\boldsymbol{y}$ means estimating the following model generating $\boldsymbol{y}$:

$$\boldsymbol{y} = \mathbf{A}\boldsymbol{s} \tag{3.67}$$

where $\mathbf{A}$ and $\boldsymbol{s}$ are the same as in definition 2.

Definition 3 is a simplification of definition 2. For many applications this definition is sufficient. Definitions 1 and 3 become equivalent if certain measures of independence are used in definition 1 and $\mathbf{W} = \mathbf{A}^T$.

To determine the independent components usually some kind of objective- (or contrast-) function is defined which leads to the independent components as it is minimized or maximized. Of course the exact statistical properties of ICA depend on the objective function. A lot of different approaches to ICA and thus a great variety of objective functions have been proposed. For more information refer to further literature [Hyvärinen, 1999; Bingham and Hyvärinen, 2000].

ICA and AAMs

Recently ICA has been applied to AAMs [Üzümcü *et al.*, 2003]. Using ICA has been compared to the standard AAM approach using PCA. It has been shown that ICA obviously has a big advantage compared to PCA. ICA seems to detect local variations in the training data. This is not the case for PCA where modes of variations in appearance are global and usually affect the complete texture and shape of an object. Figure 3.16 illustrates variations for both PCA and ICA.

Chapter 4

Model Matching

A statistical model as described in the previous chapter allows one to describe a given volumetric object and its statistical variances in a very compact way. Only few parameters describe the variances in the object's appearance. The assumption is made that the model can form all – or at least most – possible instances of the modelled object of interest.

In this chapter we discuss how the model can be used to imitate and thus analyze *unseen* data. The idea is to first superimpose the model over a given image (or volume respectively). Then the model parameters are changed until the model is as close to the target image as possible. If the difference between model and target becomes sufficiently small it can be stated that the model's appearance very well reflects the appearance of the target object. In other words the image data's shape, texture, orientation, position and average intensity are known.

4.1 Parameters for Matching

Before we start to discuss methods for fast optimization we extend the description of the model a little. So far actual scaling, rotation and translation has not been part of the actual model. Positional information has been filtered out intentionally. When the model is matched to unseen data this information has to be added again. To match the model the correct modes of variation have to be found and the model also has to be scaled, rotated, and translated.

Analogously the information filtered out for the texture model has to be considered for matching. Since model texture is normalized with respect to mean and variance of intensities these two values also have to be added explicitly to the model description for the matching process.

The vector of model parameters is enlarged by the elements which describe the model's pose and global deviations of mean and variances of gray values. Pose can be represented using a quaternion written as a vector $\boldsymbol{q}$ with four elements [Shoemake, 1985]. This quaternion defines rotation and scale. Additionally a translation vector $\boldsymbol{t}$ with three elements is used to represent the translation.

Texture intensity variations are represented by some mean β and variance α. All the mentioned components are appended to the vector of core model parameters $\boldsymbol{p}_c$ representing the main modes of variation calculated by PCA. An enlarged vector of parameters $\boldsymbol{p}$ is obtained: $\boldsymbol{p} = (\boldsymbol{p}_c|\boldsymbol{q}|\boldsymbol{t}|\alpha|\beta)^T$. In order to optimize a model match the right values for the elements of vector $\boldsymbol{p}$ have to be found.

4.2 Matching as Minimization

As mentioned before the model matching process tries to minimize the difference between some model and a target. For this one needs some concrete measurement for the difference between model and target. The root mean square of the sum of differences of individual gray values can very well be used to estimate to which extent model and target fit together:

$$d(model(\boldsymbol{p}), target) = \sqrt{\sum_{i}^{N_g} (g_{model,i} - g_{target,i})^2}. \tag{4.1}$$

where $g_{model,i}$ and $g_{target,i}$ are the i-th gray values of model instance and target respectively. N_g is the number of texture values contained in the model. Once such a difference is defined, matching the model can be regarded as a minimization of the difference between unseen data and model. The idea is to incrementally update the model parameters in order to improve an initial guess until no better matching can be achieved or some other termination criterion becomes true.

4.3 AAM Search

At a first glance it seems to make sense to apply any general optimization technique to solve the matching problem. A general strategy to do this higher dimensional non-linear optimization problem should be sufficient to do the job. Possible options include the Nelder Simplex, Powell's Method or randomized approaches like Simulated Annealing and Genetic Algorithms. All these methods work fine when solving optimization problems for general instances. For matching a specific model, however, we assume that the unseen data to which we match the model represents the same kind of object. In other words, the problem instances for which we have to carry out optimization are inherently very similar. The idea is to learn the search paths which have to be similar due to the similarity of the actual problem instances. In literature this method is usually referred to as AAM search [Cootes *et al.*, 1998a].

The important question is what should be understood by "learning the search path". Imagine a face model is put over an image of a real face. We assume further that the model is slightly translated relatively to the image. We notice that the texture sampled under the shape of the model tells us something about how the model has to be translated so that the model texture and the image texture fit together better. In other words the difference of the current model texture and the image texture

encodes information about how the parameters of the model have to be updated in order to achieve a better matching. Formally we state that there is a function f describing the correlation between the vector of texture differences $\boldsymbol{\delta}_g$ and the vector of model parameter updates $\boldsymbol{\delta}_p$:

$$\boldsymbol{\delta}_p = f(\boldsymbol{\delta}_g). \tag{4.2}$$

For the training data sets one can obtain deviations in the parameters $\boldsymbol{\delta}_p$ and the according deviations in the gray values $\boldsymbol{\delta}_g$. The crucial point now is to derive a function that does not only reproduce this data but also interpolates unknown data accordingly. This problem – finding a function that sufficiently estimates a given set of observations – is generally known as regression. The points in figure 4.1 illustrate observations of two variables along X- and Y-axis. The distribution of points suggests that both variables correlate. The straight line represents the linear regression which determines the simplified correlation of both variables. Knowing this regression line one can estimate X knowing Y and vice versa. In other words linear regression gives us the possibility to predict the value of one variable by measuring its correlated "twin".

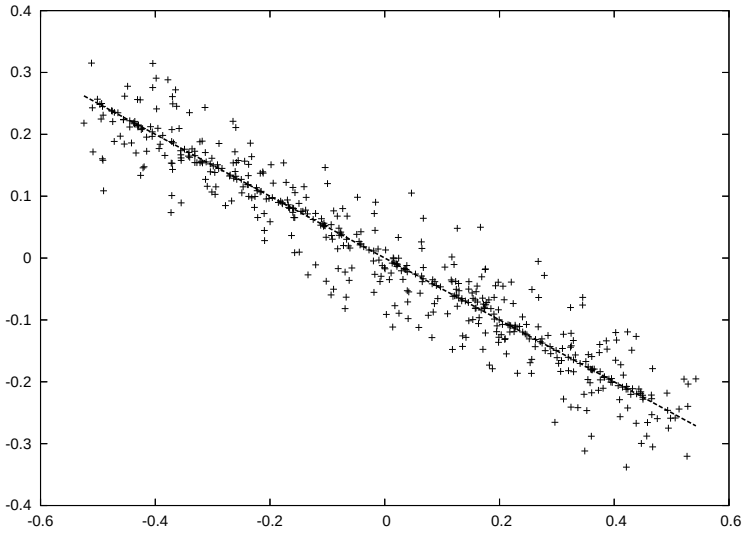


Figure 4.1: The principle of linear regression. The observed data is approximated by a straight line.

For the moment we assume that some function f as mentioned above could be deduced in order to estimate the correlation of texture and parameter changes. We will see later how this can be done using multivariate linear regression. Once a regression is known the iterative AAM Search (algorithm 2) can be applied.

Usually it is assumed that the initial position of the model is close to the object which is to be found in the image. Thus a relatively good initialization has to be found. This can, for example, be achieved by manually placing the model.

1: Place a model instance with mean appearance onto the unseen data
2: **repeat**
3: Determine the texture difference $\boldsymbol{\delta}_g$
4: Calculate the update of parameters:
$\boldsymbol{\delta}_p \leftarrow f(\boldsymbol{\delta}_g)$
5: Update the parameters of the model instance accordingly: $\boldsymbol{p} \leftarrow \boldsymbol{p} + \boldsymbol{\delta}_p$
6: Calculate the difference between model and image:
$E \leftarrow d(model(\boldsymbol{p}), target)$.
7: **until** E is smaller than some threshold

Algorithm 2: The standard AAM search [Cootes *et al.*, 1998a].

4.4 Multivariate Linear Regression

For the discussion of AAMs so far we have often assumed linearity. We proceed in this fashion and assume that the correlation of texture difference and model parameter update is locally linear. Then f in 4.2 can be represented by a simple matrix multiplication and we obtain:

$$\boldsymbol{\delta}_p = \mathbf{R}\boldsymbol{\delta}_g. \tag{4.3}$$

The remaining problem is to find an appropriate regression matrix $\mathbf{R}$.

The idea of the standard AAM approach is to estimate $\mathbf{R}$ in a precalculation step. The parameters of a model instance are changed and the according differences in texture are measured. Based on this information it should be possible to calculate matrix $\mathbf{R}$. We represent the texture differences measured each as a column vector of a matrix $\boldsymbol{\Delta}_g$. The corresponding parameter differences are the column vectors of a second matrix $\boldsymbol{\Delta}_p$. This extends equation 4.3 to

$$\boldsymbol{\Delta}_p = \mathbf{R}\boldsymbol{\Delta}_g. \tag{4.4}$$

The i-th column vector of model parameter differences in $\boldsymbol{\Delta}_p$ corresponds to the i-th column vector of texture differences in $\boldsymbol{\Delta}_g$. Algorithm 3 is used to determine both matrices $\boldsymbol{\Delta}_p$ and $\boldsymbol{\Delta}_g$.

In the following we denote the number of texture samples N_g and the number of parameters involved N_p. The parameter changes with according texture changes are calculated N_{exp} times. In other words $\boldsymbol{\Delta}_p$ has dimensions $N_p \times N_{exp}$, matrix $\mathbf{R}$ has dimensions $N_p \times N_g$ and matrix $\boldsymbol{\Delta}_g$ has dimensions $N_g \times N_{exp}$. Once we have obtained $\boldsymbol{\Delta}_p$ and $\boldsymbol{\Delta}_g$, we can use equation 4.4 to calculate $\mathbf{R}$.

We determine eigenvectors $\boldsymbol{\Phi}$ and eigenvalues $\boldsymbol{\Lambda}$ for matrix $\boldsymbol{\Delta}_g^T\boldsymbol{\Delta}_g$:

$$\boldsymbol{\Delta}_g^T\boldsymbol{\Delta}_g\boldsymbol{\Phi} = \boldsymbol{\Phi}\boldsymbol{\Lambda}. \tag{4.5}$$

Since the matrix of eigenvectors is orthonormal, $\boldsymbol{\Phi}$ multiplied by its transpose is the identity matrix $\mathbf{I}$:

$$\boldsymbol{\Phi}^T\boldsymbol{\Phi} = \mathbf{I}. \tag{4.6}$$

Equations 4.5 and 4.6 directly lead to:

$$\boldsymbol{\Phi}^T = \boldsymbol{\Lambda}^{-1}\boldsymbol{\Phi}^T\boldsymbol{\Delta}_g^T\boldsymbol{\Delta}_g. \tag{4.7}$$

1: **for** each training data set $i = 1 \dots n$ **do**
2: Create a model instance with the parameters set to the according parameters for the i-th training data set.
3: **for** each parameter $j = 1 \dots k$ considered in the matching process **do**
4: **for** $c \in \{\pm c_1, \pm c_2, \dots\}$ **do**
5: Create the vector $\boldsymbol{\delta}_p$ consisting of zeroes except for the j-th element which is set to the current displacement c: $\boldsymbol{\delta}_p \leftarrow (0, \dots, 0, c, 0, \dots, 0)^T$.
6: Append $\boldsymbol{\delta}_p$ to matrix $\boldsymbol{\Delta}_p$.
7: Create a model instance according to the new parameters and determine the difference in texture $\boldsymbol{\delta}_g$.
8: Append $\boldsymbol{\delta}_g$ to matrix $\boldsymbol{\Delta}_g$.
9: **end for**
10: **end for**
11: **end for**

Algorithm 3: Building matrices $\boldsymbol{\Delta}_p$ and $\boldsymbol{\Delta}_g$ for linear regression.

Remember that usually the number of texture samples in AAMs is very large compared to the number of training sets. This means that the column vectors of $\boldsymbol{\Delta}_g$ are very much larger than its row vectors. Our goal is to project the large texture vectors to a lower dimensional space. In the last chapter we have used PCA to reduce dimensionality. For the linear regression we will also use the principal components to project $\boldsymbol{\Delta}_g$ to lower dimensional space. In equation 4.5 we have introduced the eigen decomposition of $\boldsymbol{\Delta}_g^T \boldsymbol{\Delta}_g$. It can be shown that $\boldsymbol{\Delta}_g$ multiplied by the matrix of eigenvectors $\boldsymbol{\Phi}$ is the matrix $\boldsymbol{\Phi}_{pc}$ whose columns are the principal components of $\boldsymbol{\Delta}_g$:

$$\boldsymbol{\Delta}_g \boldsymbol{\Phi} = \boldsymbol{\Phi}_{pc}. \tag{4.8}$$

This directly leads to

$$\boldsymbol{\Phi}_{pc}^T \boldsymbol{\Delta}_g = \boldsymbol{\Phi}^T. \tag{4.9}$$

Equation 4.9 states that $\boldsymbol{\Phi}_{pc}^T$ projects $\boldsymbol{\Delta}_g$ to lower dimensional space. We can thus replace the original matrix $\boldsymbol{\Delta}_g$ by ϕ^T for the calculation of the regression. The right side in equation 4.4 is replaced by $\mathbf{R}'\boldsymbol{\Phi}^T$:

$$\boldsymbol{\Delta}_p = \mathbf{R}'\boldsymbol{\Phi}^T \tag{4.10}$$

where $\mathbf{R}'$ represents the correlation between the parameter differences $\boldsymbol{\Delta}_p$ and the texture differences after projection $\boldsymbol{\Phi}^T$.

Further the left side of equation 4.10 is replaced by the right side of 4.4:

$$\mathbf{R}\boldsymbol{\Delta}_g = \mathbf{R}'\boldsymbol{\Phi}^T \tag{4.11}$$

Now the left side of equation 4.7 is used to replace $\boldsymbol{\Phi}$ in equation 4.11 and $\boldsymbol{\Delta}_g$ is removed from both sides of the equation. This leads to:

$$\mathbf{R} = \mathbf{R}'\boldsymbol{\Lambda}^{-1}\boldsymbol{\Phi}^T\boldsymbol{\Delta}_g^T. \tag{4.12}$$

According to equation 4.10 $\mathbf{R}'$ can be expressed as:

$$\mathbf{R}' = \boldsymbol{\Delta}_p\boldsymbol{\Phi}. \tag{4.13}$$

Together equation 4.12 and equation 4.13 lead to

$$\mathbf{R} = \mathbf{\Delta}_p \mathbf{\Phi} \mathbf{\Lambda}^{-1} \mathbf{\Phi}^T \mathbf{\Delta}_g^T. \tag{4.14}$$

Equation 4.14 allows us to finally calculate matrix $\mathbf{R}$ as claimed by the linear correlation formulated in equation 4.2. This matrix represents correlations of texture and parameter differences and is the key to efficient model matching. It allows the design of very fast gradient-based optimization algorithms for matching a model to unseen data. In algorithm 4 we resume the two necessary steps for calculating the multivariate linear regression.

1: Calculate the difference matrices $\mathbf{\Delta}_p$ and $\mathbf{\Delta}_g$ using algorithm 3
2: Calculate the regression matrix using equation 4.14

Algorithm 4: Calculating multivariate linear regression for AAM search.

4.5 Parameter Displacements for Regression

In algorithm 3 the model is put over the original training data sets. The model parameters are displaced by amounts $\pm c_1, \pm c_2, ...$ and the differences in the texture vectors are determined. The remaining question is how many c_i should be taken and which values they should have.

Cootes and Taylor (2000) suggest to use ± 0.5 standard deviation (over the training set) of the core model parameters, about 10% for scaling, the equivalent of 3 pixels translation and about 10% change in texture normalization parameters. This suggestion is made for a 2D face model.

Mitchell *et al.* (2002) use between ± 1.5 standard deviation for the core model parameters, up to $3-5$ voxels for translation and 10% for the remaining parameters. The authors use these values for a volumetric model of the left cardiac ventricle.

4.6 Elastic AAMs

The great advantage of AAMs is their compact description of deformation. Only a few parameters explain the most important deformations of the object of interest. These most important deformations are learned from the training set through statistical analysis. The assumption is made that unseen data can be approximated quite well with the known deformations. Anyway, this is not always the case. A representative training set increases the possibility of the model to match unseen data sufficiently. However there is still no guarantee that for some instances matching the model leads to sufficient results.

Taylor and Cootes (2000) propose a method to explicitly extend the model's elasticity by allowing additional small deformations. The idea is to use standard AAM search to find an initial solution. Then additional deformations are carried out on the shape of the model in order to improve the fit.

4.6.1 Local AAMs

To introduce local deformations the standard representation of the model as given in equation 3.55 is extended. A deformation vector $\boldsymbol{\delta}_x$ that affects only shape is added:

$$\boldsymbol{c} = \begin{bmatrix} \boldsymbol{s} \\ \boldsymbol{g} \end{bmatrix} = \begin{bmatrix} \overline{\boldsymbol{s}} \\ \overline{\boldsymbol{g}} \end{bmatrix} + \begin{bmatrix} \boldsymbol{\Phi}_s \mathbf{W}_s & 0 \\ 0 & \boldsymbol{\Phi}_g \end{bmatrix} \boldsymbol{\Phi}_p \boldsymbol{p} + \begin{bmatrix} \boldsymbol{\delta}_x \\ 0 \end{bmatrix}. \tag{4.15}$$

The remaining question is how $\boldsymbol{\delta}_x$ can be calculated so that matching is improved.

It can be shown that the overall idea of the AAM search can also be formulated in a local context. The standard AAM search sets up a correlation between deformations and texture differences. A similar strategy is used for local deformations. The main difference is that a deformation is not considered to affect the whole shape but only a single point of it. Analogous to the standard AAM search the interrelation of displacement of individual shape points and changes in texture can be learned. With this precalculated information local deformations can be predicted.

The displacements of individual points should only depend on their surrounding texture. Texture far away from a landmark point should not influence the prediction of the point's displacement. Local AAMs only deal with local texture. In the following we discuss how locality of texture can be introduced.

For each mean-shape landmark point $\overline{\boldsymbol{x}}_i$ a (sparse) matrix $\mathbf{W}_i$ is defined that contains information about which texture samples are supposed to influence the point's displacement. Imagine the displacement of the i-th point is governed by N_i texture samples stored in a vector $\boldsymbol{g}_i$. The complete texture information is stored in vector $\boldsymbol{g}$ which contains all N_g texture samples. Necessarily N_g is much larger than N_i: $N_g \gg N_i$. Matrix $\mathbf{W}_i$ extracts the local samples from the vector of all samples and thus maps $\boldsymbol{g}$ to $\boldsymbol{g}_i$:

$$\boldsymbol{g}_i = \mathbf{W}_i \boldsymbol{g}. \tag{4.16}$$

More precisely $\mathbf{W}_i$ is a $N_i \times N_g$ matrix with only one element not equal to zero per row. The non-zero elements of $\mathbf{W}$ can be chosen such that they form some kind of kernel. For example a Gaussian kernel smoothly increases the importance of samples as they lie closer to the considered mean-shape landmark point.

In a preprocessing step the matrices $\mathbf{W}_i$ have to be determined. For each shape point the closest samples have to be found. For this some kind of space subdivision such as an octree or a simple grid can be used. We stress that the distance between a texture sample and a point is determined in the mean shape reference frame.

To train local AAMs, individual shape points are displaced randomly and the changes of surrounding texture samples are observed. By regression the correlation of point displacement and change of texture is set up for each point individually:

$$\mathbf{D}_{x,i} = \mathbf{R}_i \mathbf{W}_i \boldsymbol{\Delta}_g. \tag{4.17}$$

Here $\mathbf{D}_{x,i}$ denotes the matrix whose column vectors represent the displacements of the i-th point. When the model is built in 3D each such a column vector contains 3 elements. $\boldsymbol{\Delta}_g$ includes the according changes of texture as column vectors. This is very similar to the way in which the standard AAM search is prepared with the

difference that this time parameter changes are replaced by changes of individual points. So for each point a local linear regression is introduced.

4.6.2 Restrictions for Local Deformations

So far local deformations have been introduced that allow shape points to change their position almost independently. "Almost independently" because two neighboring shape points are allowed to displace in completely different directions. This can lead to more or less chaotic deformations of contours. To avoid this behavior two important restrictions are made to local deformations:

- The deformations have to be limited to some magnitude. This is claimed because the shape of the model still has to represent a plausible instance of the object after the additional deformations have been applied.
- The deformations have to be smooth in order to avoid overfitting of the model.

The first restriction can easily be realized by limiting all deformations by some threshold. The second restriction is a little harder to implement. Some post processing of the complete deformation vector $\boldsymbol{\delta}_x$ has to be carried out in order to make the predicted shape change smoother.

The problem we have to deal with can be formulated as follows. A set of points as a single shape vector $\boldsymbol{x}$ together with the predicted deformation $\boldsymbol{\delta}_x$ is given. How can a vector $\tilde{\boldsymbol{\delta}}_x$ be derived which describes a similar but smoother deformation? One method is to use a smoothing filter. To the displacement of an individual point the weighted sum of displacements of adjacent points is added. Close points should have great influence and distant points should have no influence. In other words, the weights depend on the distance to the currently considered point. Let $d_{i,j}$ be the Euclidean distance of two points $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$. Then the weight for point $\boldsymbol{x}_j$ when smoothing the displacement of point $\boldsymbol{x}_i$ is

$$m_{ij} = \frac{e^{-d_{ij}/(2\sigma^2)}}{\sum_j e^{-d_{ij}/(2\sigma^2)}} \tag{4.18}$$

where σ is a parameter which determines how strong distant points influence the currently considered point. We remind the reader that shape vectors include coordinates point by point. A 3D shape, for example, has the elements $\boldsymbol{x} = (x_1, y_1, z_1, x_2, y_2, z_2, \ldots x_n, y_n, z_n)^T$. The same weights m_{ij} have to be applied to all coordinates. Weighting a vector can thus simply be written as a multiplication with the diagonal matrix $\mathbf{M}_{ij} = \operatorname{diag}(m_{ij}, m_{ij}, \ldots, m_{ij})$: $\tilde{\boldsymbol{x}} = \mathbf{M}_{i,j}x$.

To describe the complete smoothing operation as matrix multiplication, a bigger matrix $\mathbf{M}$ is considered whose elements are the matrices $\mathbf{M}_{ij}$ mentioned above. Smoothing of the predicted deformation can finally be written as:

$$\tilde{\boldsymbol{\delta}}_x = \mathbf{M}\boldsymbol{\delta}_x. \tag{4.19}$$

Note that for distant points the weights are very small due to the exponentially decreasing weights with increasing distance. This means that small elements (weights

for distant points) in matrix $\mathbf{M}$ can be considered zero. In other words $\mathbf{M}$ is a sparse matrix. This feature has to be exploited for an implementation. The matrix should be represented by adequate data structures in order to increase performance.

4.7 Fine Tuning

The standard AAM model search as described above converges very fast. Anyway there is still the possibility that the final matching achieved is not optimal. This is especially true if the given unseen data differs significantly from the data in the training set.

Stegmann (2000) suggests to carry out some fine tuning of the actual AAM search. This is legitimate since in general one can assume that the AAM search has already found a solution near the overall optimum.

In general any optimization technique could be applied. Possible options include the Nelder Simplex or Simulated Annealing. A great problem with such general optimization techniques is that the evaluation of the function to be optimized usually is computationally very costly. This is the case because all texture samples of the model have to be compared with the original data. Below we review a method to reduce the amount of texture. Similar strategies have already been used with other types of deformable models [Jones and Poggio, 1998].

4.8 Multiresolution Search

The performance of standard AAMs often suffers from the huge number of texture samples used to represent a model. Especially for 3D AAMs the huge amount of texture slows down the matching process. A relatively simple way to reduce texture [Cootes *et al.*, 1998b; Cootes and Taylor, 2000] is to use low resolution model representation. The drawback of this is that such a low resolution model leads to inexact matching. In the following we discuss an approach for speeding up model matching by treating texture efficiently.

The easiest way to reduce texture is to leave out texture that is not important. The question is what should be understood by "not important". In general it can be said that for model matching only the texture samples are interesting which help to predict the changes of the model parameters to obtain a better matching [Cootes *et al.*, 1998b]. In general the predicted update of a single model parameter p_i can be calculated using the following equation.

$$\delta_{p,i} = \mathbf{r}_i \boldsymbol{\delta}_g \tag{4.20}$$

where $\boldsymbol{r}_i$ denotes the i-th row of the regression matrix $\mathbf{R}$. The elements with the highest absolute values in $\boldsymbol{r}_i$ indicate that the according texture sample has a great influence on parameter p_i. Because of this the most important texture samples (denoted u) can be determined for each parameter p_i. If u is chosen small enough the

union of most important texture samples of all parameters is still much smaller than the total number of texture samples.

Once the samples with the greatest influences $\boldsymbol{\delta}'_g$ are found the model can be trained using only these samples. In other words another linear regression is calculated:

$$\boldsymbol{\delta}_p = \mathbf{R}'\boldsymbol{\delta}'_g. \tag{4.21}$$

In practice different sets of texture samples are considered. Each set represents a different resolution of the model. The search is started with the lowest resolution. After the search has converged for one resolution it is continued at the next finer level with the next higher resolution.

Thus the amount of texture that has to be resampled for each prediction of model parameter updates can be reduced dramatically, at least for the beginning of the search. This is important since resampling of difference texture is crucial to the computational efficiency of the model matching process. Further it was shown that the robustness of the search also is increased [Cootes *et al.*, 1998b].

4.9 Texture Compression

In chapter 4.8 we have already outlined how the amount of texture data needed for matching the AAM can be reduced. The basic idea of the outlined method is to simply remove texture samples that do not contribute much information to the AAM search. In the field of image compression, however, there exist more sophisticated methods to compress image data. Using some kind of frequency analysis usually allows to decrease the amount of texture data strongly while reducing quality only to some small extent. Usually texture is the most critical entity of an AAM implementation with respect to size. Thus efficient compression techniques offer an interesting approach to reduce the amount of texture significantly and improve the efficiency of AAMs. Several authors have proposed wavelet compression or similar techniques for the use with AAMs [Wolstenholme and Taylor, 1999; Stegmann *et al.*, 2004; Darkner *et al.*, 2004]. To our knowledge only implementations in 2D have been reported so far.

The main idea behind the application of elaborated compression techniques for AAMs is to replace the original texture information by the compressed data. The idea is that not the difference in texture but the difference in the compressed data is used to predict model parameter updates. This makes AAMs not only computationally more efficient but also more robust.

4.9.1 Wavelets for Image Compression

Wavelet analysis is an extremely powerful method used for compression of image data at high compression rates with relatively little loss of information. The fast wavelet transform (FWT) [Mallat, 1989] makes the computation of wavelet compression especially attractive. It is a fast implementation of the discrete wavelet transform and allows efficient handling of digital image data.

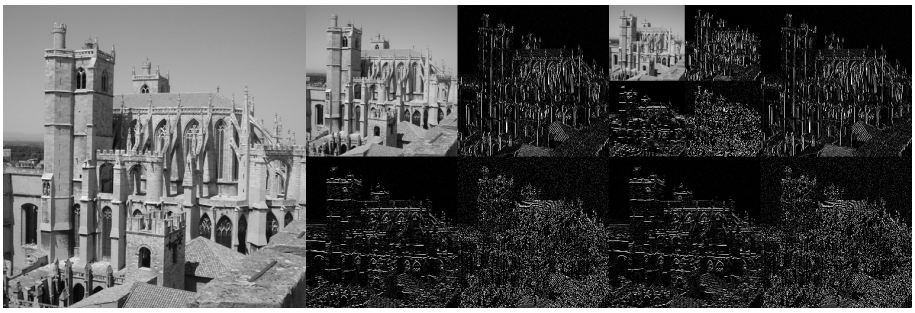


Figure 4.2: Two steps of the wavelet decomposition using the Haar wavelet.

The basis for fast wavelet analysis is the repeated application of high- and low-pass filters to the original signal. The low-pass filters extract an approximation of the global signal. The high-pass filters extract details. The result of the iterative filtering operations is a set of wavelet coefficients. From the wavelet coefficients the original image can be reconstructed. In order to achieve a reasonable compression the coefficients which are close to zero are removed completely. This leads to only very little loss of quality since the low-valued coefficients do not contribute much information to the image.

In fact the wavelet decomposition can be regarded as a change of the basis. The new basis functions after decomposition are the wavelet functions, which are all a scaled copy of the mother wavelet.

For wavelet decomposition the rows and columns of a 2D image are filtered with both low- and high-pass filters. This is illustrated in figure 4.2. The figure shows the original image and the first two steps of wavelet decomposition. The four possible combinations of the high- and low-pass filters result in a scheme that creates a copy from the original image at a lower resolution and three additional images representing the coefficients of high frequencies which were filtered out. Always the low-resolution image from the last iteration is decomposed again. This scheme can directly be extended to 3D [Rodler, 1999].

4.9.2 Wavelets and AAM Texture

Wavelets can be used to efficiently represent texture in AAMs [Wolstenholme and Taylor, 1999; Stegmann *et al.*, 2004]. We have already mentioned that the large amount of texture is the bottleneck for matching AAMs. Wavelets offer the possibility to improve performance significantly.

A great advantage of texture compression for AAMs is that for the matching process texture does not have to be decompressed. The wavelet coefficients are directly used for prediction of parameter updates. Only if the difference in real texture is desired the wavelet compressed texture has to be reconstructed.

In principle the use of wavelet compression for texture representation in AAMs does not influence the overall structure of the model. The texture model from the standard AAM simply is replaced by a wavelet coefficient model. For this wavelet

coefficient model the considered texture vector $\boldsymbol{g}$ is compressed using wavelets and further analysis including PCA is performed on the vector of wavelet coefficients $\boldsymbol{w}$.

When building a standard AAM the texture is traversed in the mean shape and texture samples are directly appended to the texture vector which is then analyzed statistically. When wavelets are used only another step between texture extraction in mean shape space and statistical analysis has to be introduced. This step in between takes the raw texture samples, performs a wavelet decomposition with truncation of small coefficients and hands the coefficients over to the statistical analysis.

When a wavelet enhanced model is matched to unseen data similarly in a first step the raw texture is extracted from the data. Then a wavelet compression is performed on it. For the prediction of model parameter updates the vector of wavelet coefficients is multiplied by the regression matrix built for compressed texture. Immediately after this the model parameters can be updated without reconstruction of original texture data.

Recently Stegmann *et al.* (2004) have reported that wavelets allow compression of AAM texture up to a rate of about 1:40. As basis functions the Haar Wavelet and the Daubechies-9/7-Wavelet were used.

4.9.3 Multi-Level Wavelet-Enhanced AAMs

In chapter 4.8 a method was introduced for improving the model matching process using multiple resolutions of texture. Only texture samples were used which contribute most to the parameter updates during model search.

Wavelets allow the use of a similar strategy. The idea also is to use multiple resolutions of texture. For the multiresolution search of the standard AAM the contribution of an individual texture sample is the criterion for whether the sample is used for parameter update or not. When wavelets are used the reduction of texture data is performed implicitly by the wavelet compression.

Wavelet compression is carried out at different levels resulting in images with higher or lower quality and more or less data needed to represent the image. As for the multiresolution search of uncompressed data the model can be trained for multiple levels of wavelet compression. The AAM search then starts with the maximum compression and switches to lower compression rates as the search converges. Figure 4.3 shows a 2D AAM of the human brain with 6 different levels of compression.

As mentioned above the multiresolution search of uncompressed models and multi-level search of wavelet-compressed models is insofar different as different criteria for compression (contribution to search and wavelet coefficients) are used. In principle both methods can be combined.

As we have already mentioned that wavelet enhanced AAMs have already been applied successfully to 2D images. An adaptation to 3D has not yet been implemented and will be one of the next logical steps in the evolution of volumetric AAMs.

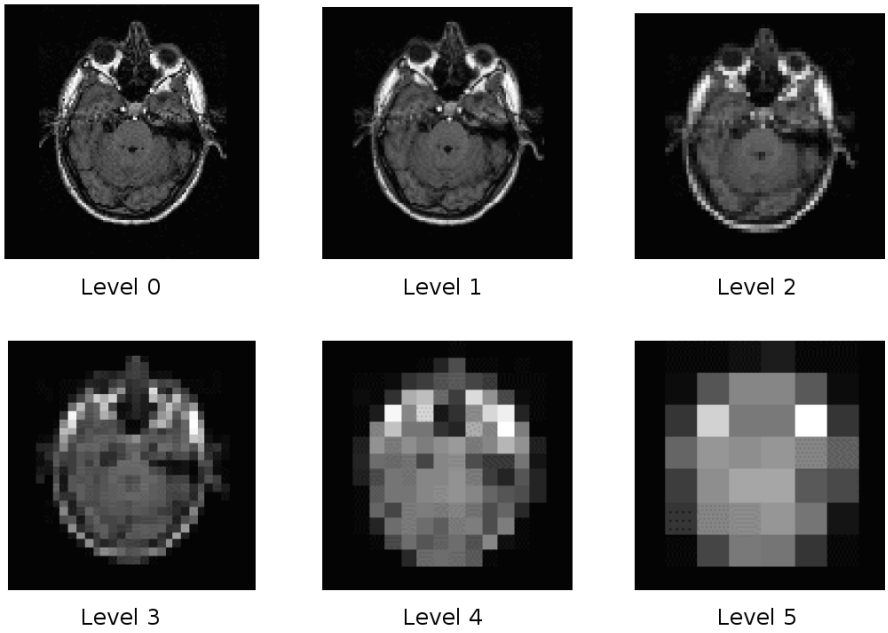


Figure 4.3: Different levels of a wavelet compressed AAM of the human brain. Figure from [Wolstenholme and Taylor, 1999].

4.10 Analytical Approaches

Above we have already described in detail how linear regression is used to estimate optimal model parameter updates from measured texture differences. This is the standard approach for AAM search and in general leads to a very fast convergence after only a few iterations. The assumption is made that parameter changes and texture changes correlate linearly. It has been shown that this assumption at least roughly holds as long as the model parameters are displaced only slightly [Cootes *et al.*, 1998a; Stegmann, 2000].

Although linear regression for AAM search is fast and relatively easy to implement it has one great drawback. The estimates for parameter updates can be more or less inaccurate depending on the stability of the calculated linear regression. Researchers recently have proposed techniques to analytically calculate the parameter updates [Baker and Matthews, 2001]. In this section we outline the main ideas of such analytical approaches.

4.10.1 The Lucas-Kanade Algorithm

The Lucas-Kanade algorithm was originally introduced to solve the problem of rigid registration of images [Lucas and Kanade, 1981]. Given two similar images the algorithm aims to find the transformation optimally aligning one image to the other. In the original formulation only translations are considered.

The idea of the Lucas-Kanade algorithm is illustrated for the 1D case in figure 4.4. Imagine function F represents the target image and G stands for the model. The problem is to find the optimum shift h which aligns both functions. For a small h it can be assumed that

$$F'(x) \approx \frac{F(x+h) - F(x)}{h} = \frac{G(x) - F(x)}{h} \tag{4.22}$$

so that

$$h(x) \approx \frac{G(x) - F(x)}{F'(x)}. \tag{4.23}$$

Equation 4.23 shows how to calculate an estimate of the desired optimum shift $h(x)$. The only problem with this estimate is that in general it depends on the position x where it is calculated. The solution is to calculate $h(x)$ at multiple positions and to use an average as the final result. For positions x where $F(x)$ is approximately linear the estimate $h(x)$ is more accurate and worse where $|F''(x)|$ is large. This is the reason why for calculation of the final optimum h a weighted sum is used where individual elements are weighted with the inverse of the estimated $|F''(x)|$.

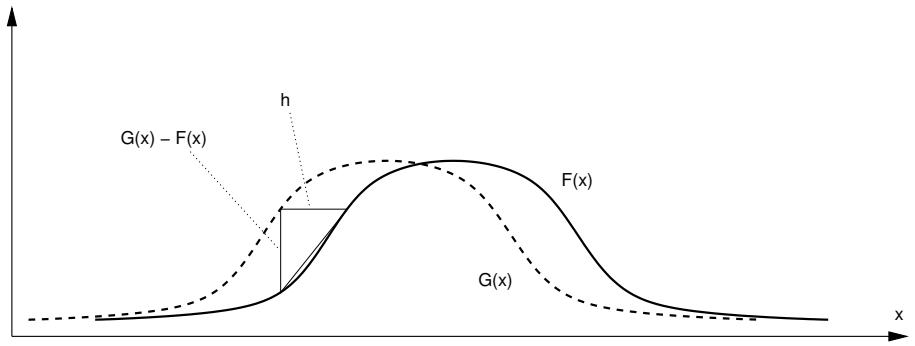


Figure 4.4: Lucas-Kanade in 1D.

We do not go too much into detail on the Lucas-Kanade algorithm. It is possible to extend it to multiple dimensions and to consider transformations that are more complex. It can be shown that not only translation but also rotation [Lucas and Kanade, 1981] and even warping can be handled [Baker and Matthews, 2001].

4.10.2 Analytically Matching Models

In the following we review some issues of the work of Baker and Matthews (2001). They suggest to analytically calculate estimates of model parameter updates in order to achieve higher accuracy in matching.

An algorithm similar to Lucas-Kanade can only handle transformations of shape. This is the reason why separate shape and texture models are used for matching instead of the combined model formulation of AAMs. In fact only the shape model can be optimized in this way.

Baker and Matthews (2001) compare their analytical approach with linear regression and report a much higher accuracy for matching. They apply AAM search

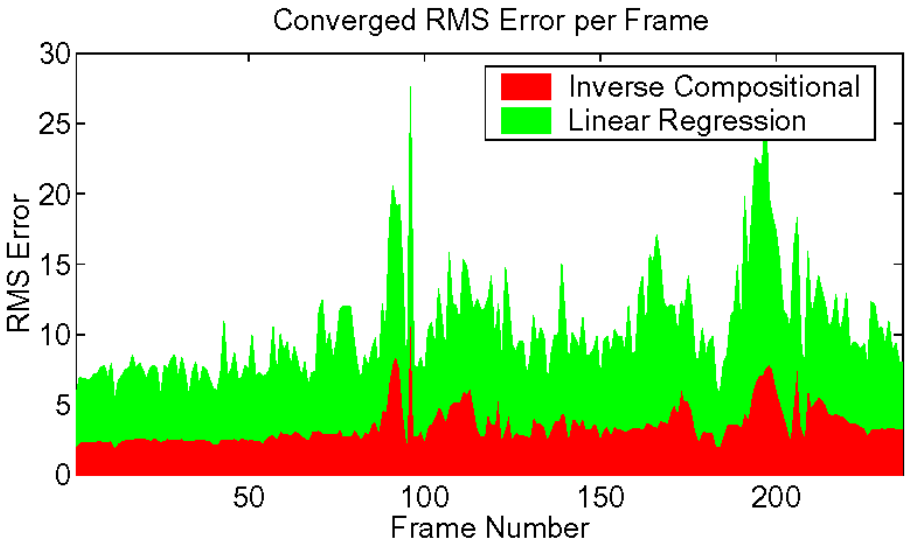


Figure 4.5: Comparison: linear regression versus inverse compositional matching. The figure shows the RMS in texture error of the converged model for all frames of a face video. Figure from [Baker and Matthews, 2001]

with both strategies to a video showing a face. Figure 4.5 illustrates the matching accuracy they achieve with linear regression and with their inverse compositional approach.

Chapter 5

Implementation Issues

While AAMs have been applied to 2D images very successfully, the implementation of volumetric AAMs still is a challenge. Theoretically AAMs in 3D can be regarded as a straightforward extension of 2D AAMs. In practice volumetric data is much harder to handle. In this chapter we try to outline the most important issues concerning an implementation of 3D AAMs.

Our implementation of AAMs includes two main functionalities:

- Building AAMs in 3D (as outlined in section 3.4) together with calculation of regression (as described in algorithm 4)
- AAM search (as described in algorithm 2)

The first task, model building, usually is carried out offline only once. For this in general computational efficiency is not so important. Once a model is built only the second functionality, the AAM search, is of importance. An application running in a clinical environment in principle only needs to be capable of matching a model to unseen data. The precalculated model can be delivered together with the application. However, it is crucial for such a software that the matching is computed as fast as possible.

We experienced that model building and calculation of the regression for later use in AAM search are relatively hard to implement while AAM search is quite a simple process. In principle the AAM search only comprises iterative updating of parameters in a loop. In this chapter we mainly focus on how model building is implemented. We also discuss the MRI data which we used to test our implementation.

5.1 Model Building

Since the implementation developed in the context of this work is a research prototype, the design was kept simple and as modular as possible. We especially tried to keep the implementation of the model building process very modular in order

to make it flexible. The intention was an implementation that allows to exchange individual steps of the algorithm by exchanging different modules that implement the individual steps of the algorithm differently.

5.1.1 Building a Combined Model

The complete process of building a combined (shape and texture) model includes several steps one following the other. First a shape model has to be built. Then a tetrahedralization of the mean shape points has to be calculated. After this the texture model can be built and so on. We decided to use a pipe-and-filter architecture to realize individual functionalities in separate processing steps. Figure 5.1 illustrates the individual filters (software modules) and pipes (input data or the resulting data from previous filters). Filters are represented by boxes including bold font. Pipes are shown as boxes with round corners. The arrows illustrate the flow of data.

In our current implementation the individual model building steps (buildShapeModel, buildTextureModel, and buildCombinedModel) are based on Eigenanalysis which is calculated by a C-version of the linear algebra package LAPACK [CLAPACK, 2005]. The subdivision of shapes into tetrahedra (buildSubdivision) is accomplished with the help of the computational geometry library CGAL [CGAL, 2005] which performs a Delaunay tetrahedralization.

Our architecture makes it possible to simply exchange individual filters. For example, currently Delaunay tetrahedralization is used for subdivision of the model's interior. For some shapes it might be necessary to apply some other kind of triangulation, for example, a non-convex one. In this case only the filter "buildSubdivision" has to be replaced while the rest of the software remains unmodified.

5.1.2 Calculating Regression for AAM Search

Training the model means to generate model instances differing slightly from the actual training sets and to measure the resulting texture differences. The applied changes of parameter values and the according texture differences are stored linear regression is used to estimate the relation between them. Our implementation realizes algorithm 4. Obtaining the data for this regression turned out to be the computationally most time-consuming part of our AAM framework.

Imagine there are 30 examples in the data set. The model is trained for 10 modes of variation plus additional parameters like position or scaling. In 3D there are (at least) 7 additional parameters to describe position, scaling and rotation of the model. This leads to a total of 17 parameters for training. Usually each model parameter is displaced $\pm 25\%$ and $\pm 50\%$ which means each parameter is displaced 4 times.

In order to get the data for the linear regression the model is superimposed over each of the 30 examples and all the 17 parameters are updated 4 times in order to measure texture differences. This means texture differences have to be measured $30 \times 17 \times 4 = 2040$ times. Keeping in mind that each texture vector contains

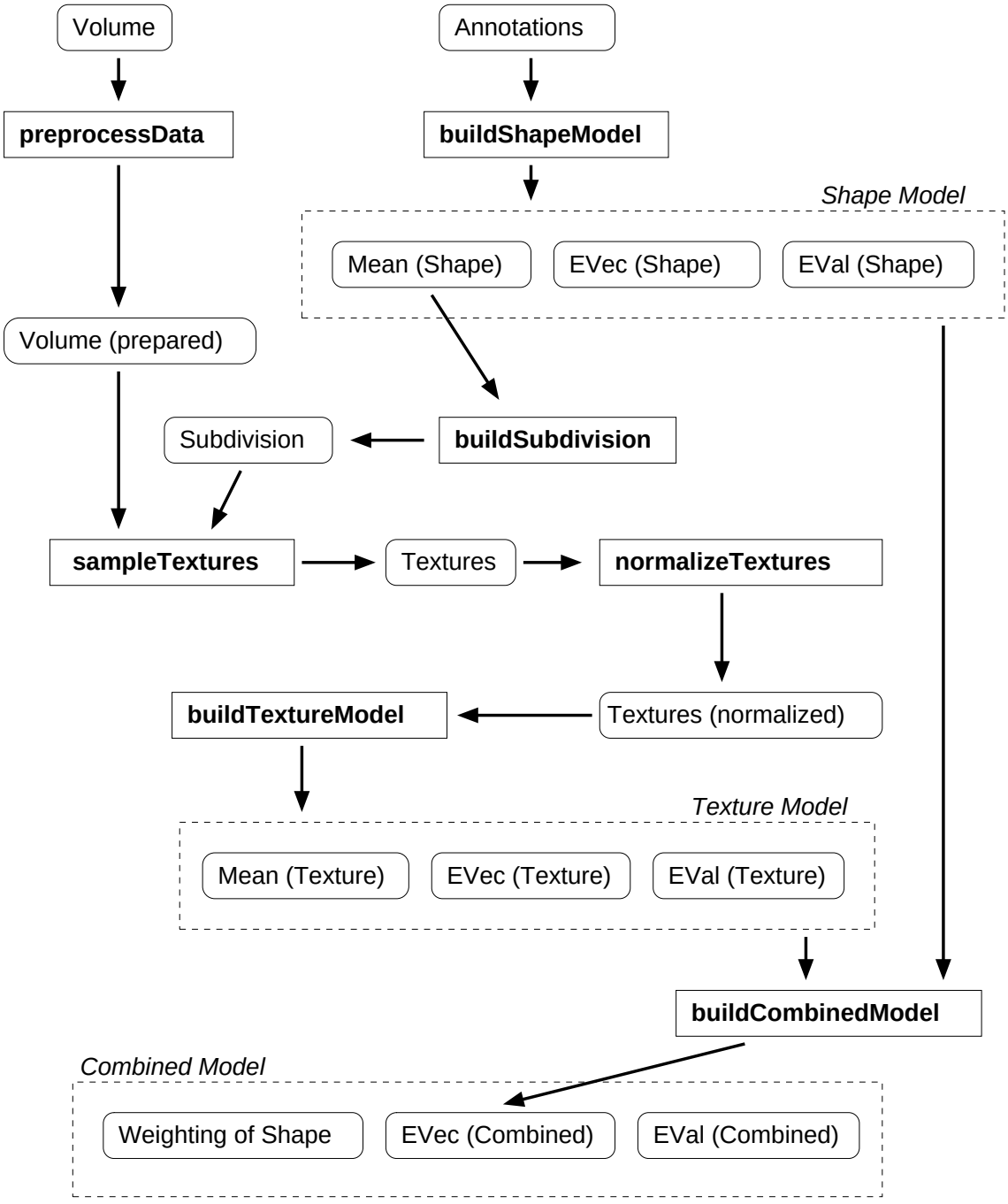


Figure 5.1: Model building overview. The boxes including bold font represent functionality. The boxes with smooth edges represent processed data. “EVec” and “EVal” denotes eigenvectors and eigenvalues respectively.

many thousand texture samples one can imagine that this takes considerable time to calculate even on modern computer hardware.

Building a model with our implementation takes between one or two minutes when the model includes about 14.000 texture samples and is built from 30 data sets. Calculating the data for regression for the same model with 17 parameters that

are displaced 4 times each one takes about half an hour. These times were measured on a notebook with a Mobile AMD Athlon XP 2500+ (1.867GHz) processor and 512MB RAM.

5.2 Model Matching

For matching the model to unseen data iteratively the texture differences have to be measured and the model parameters have to be updated accordingly. This is done as outlined in algorithm 2. The only two problematic issues concerning model matching are obtaining a good initialization and finding a reasonable termination criterion.

For the initialization of the matching algorithm the model has to be placed relatively close to the object which is to be segmented. This can be done manually and is no great effort since it only requires a simple initial positioning of the model. If manual interaction has to be avoided the model can also be placed at multiple random initial positions. For each of the positions the matching algorithm is started. Initial positions for which the search does not converge are eliminated. However, this can significantly increase the total time needed for matching since multiple AAM searches have to be executed in parallel.

The second crucial point concerning an implementation of AAM matching is to determine when the best match is achieved and how good it is. The central problem here is that the algorithm only gets volume texture as input. In other words only the difference of texture samples can be used to calculate an estimate for the quality of a match. In general it can be said that when texture difference gets small enough also the matching of the model's shape is good enough. This usually is the termination criterion of the AAM search. It is also used as the termination criterion in our implementation. However, it has to be kept in mind that texture difference is only an estimate (and not an exact measure) of the actual matching quality of the model's shape. Especially for medical applications the matches found by the algorithm should thus afterwards be inspected by medical doctors.

5.3 General Properties of the Test Data

Our training set included 32 MRI data sets showing the part of the heart illustrated in figure 2.1. Each data set contains between 7 and 13 slices each with a resolution of $256 \times 256 \times 12$bit. All data sets show the heart in the end diastole. In this state the cardiac ventricles reach their largest diameters.

All shape data is given in units of millimeters. Textures are measured as gray value intensities ranging from 0 to 1. Every time a texture sample is taken from a volume, we apply trilinear interpolation. The spacing of samples in individual slices of the original volume data ranges from 0.93mm to 1.56mm. The inter-slice distance is 7.2mm for all data sets except for data set 30 (8.4mm) and data set 32 (4.8mm).

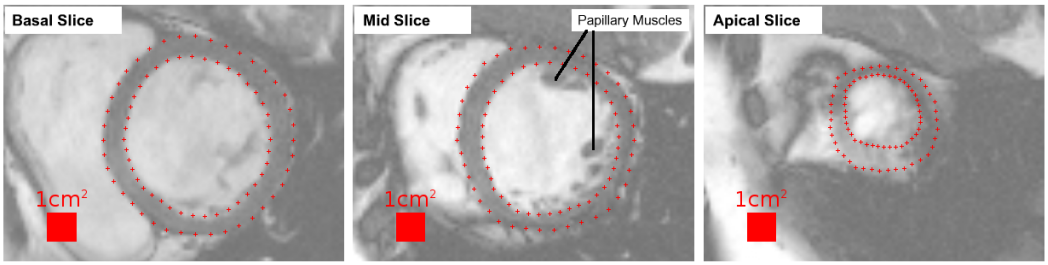


Figure 5.2: Manual annotation of a basal, a mid and an apical slice of data set 13. The annotated contours separate the left ventricle from the blood inside (endocardium) and its surrounding (epicardium, endocardium of the right ventricle).

5.4 Preparation of the Training Set

Before an AAM can be built the volume data has to be prepared. Two important things need to be considered. First, an annotation has to be obtained so that a shape model can be built. Second, the variations in histogram intensities have to be reduced as much as possible. In the following we discuss how landmark points were obtained, how shapes were aligned and how intensity variations in the original data were reduced.

5.4.1 Obtaining Annotations

Obtaining good landmark points is much more challenging when building AAMs in 3D compared to the case in 2D. We solved this problem by defining landmark points manually for each data set and slice by slice. The number of points varied as the user was free to place a random number of landmark points on each slice. The points were placed on the outside (epicardium) and inside (endocardium) of the ventricle. This was done clockwise when looking towards the apex and starting with the point where the outer muscle of the right ventricle meets the left ventricle. Figure 5.2 shows three slices of data set 13 with according landmark points.

After the initial annotation the landmark points were resampled in order to get an equal number of 40 points upon the epicardium and 40 points upon the endocardium of the left ventricle. This was achieved by calculating the central axis for each ventricle and interpolating landmark points along the normal of this axis changing the angle evenly. The axis is assumed to be perpendicular to the slices. This is illustrated in figure 5.3 (a).

Since the number of slices varies from data set to data set, the landmarking information was interpolated such that a constant number of 13 layers was obtained. This is illustrated in figure 5.3 (b). In other words, the complete irregular landmarking information is interpolated so that the shape of each data set finally is represented by 13 layers each containing 40 points representing the endocardium and another 40 points representing the epicardium of the left ventricle. This makes up the annotation which finally is used for calculation of the shape model.

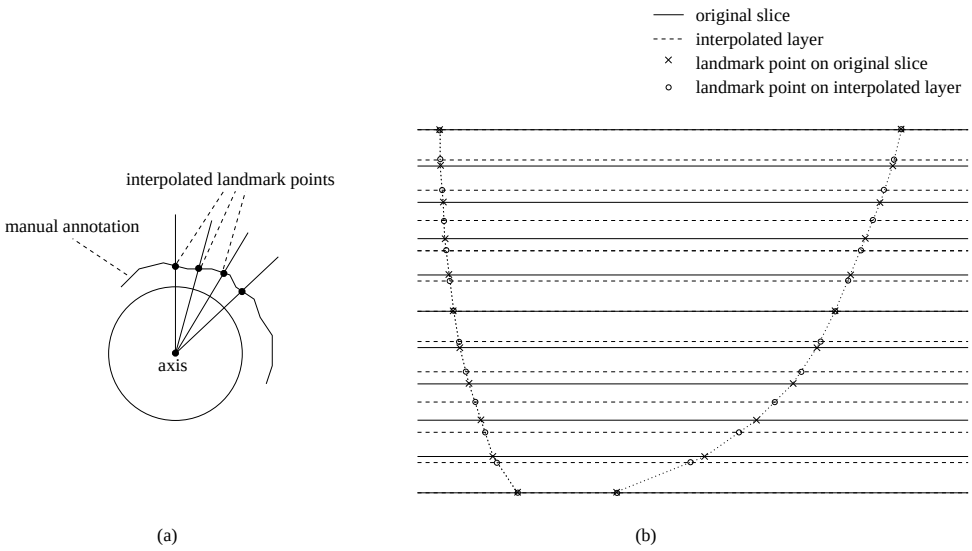


Figure 5.3: Interpolation of landmark points upon (a) and in between slices (b).

The outlined way of interpolating landmark points is very specific to the left ventricle. Landmark points are interpolated twice: upon slices and in between slices. To some extent this already falsifies the original (manually created) shape information. Another critical aspect with our strategy of obtaining annotations is outlined in the following. We use the convention that the first landmark point aligns with the point where left and right ventricle meet. This is the only real anatomical landmark point that is used. It would be much better to integrate more such anatomically meaningful points as landmark points. This would lead to a better model where correspondences of landmark points between individual data sets are represented much better. The automatic improvement of correspondence in shape is one suggestion for future work. However, in this work we stick to simple manual landmarking with additional interpolation in order to get a constant number of final landmark points.

It is desirable to include additional texture surrounding ventricle into the model. This is why we interpolated and added another ring of landmark points outside the outer border of the ventricle.

As it is the case with many medical objects also the left ventricle includes very irregular structures. The papillary muscles lying inside the ventricle (figure 5.2) can vary strongly from patient to patient. In the present work papillary muscles were not modeled with extra shape information. In other words our AAMs can only represent them in terms of texture variation.

5.4.2 Alignment of Shapes

The reference frame for annotations was chosen such that the axis of the ventricles aligns with the Z-axis. X- and Y-axes are then parallel to the slices. For the alignment the centroids of all shapes were shifted to the same position. Then a scaling was applied to unify the sizes of the individual shapes.

For the alignment we did not account for rotation. This is legitimate since all data sets were acquired such that the axes of the ventricles point into the same direction. For model matching, however, rotation was considered in order to improve the matching capabilities of the model.

5.4.3 Inter-Slice Variations of Texture

A difference in intensity distributions can be observed varying from one data set to the other and also from one slice to the other. Figure 5.4 shows the mean and standard deviations of gray value intensities of the slices of two data sets. A significant difference in the mean and variances can be observed. Since both show the same anatomical objects it can be assumed that the different distributions of mean and variance in texture intensities depend on the exact settings and circumstances of the acquisition process. To hide these variances from the model, we performed a texture normalization step as described in section 3.3.2 before using the volume data to build the model.

The same texture normalization then also has to be carried out for every data set before the AAM search can be applied. The additional normalization of the complete volume data does not replace the normalization of model textures but is intended to reduce global variances even before the actual model building process starts.

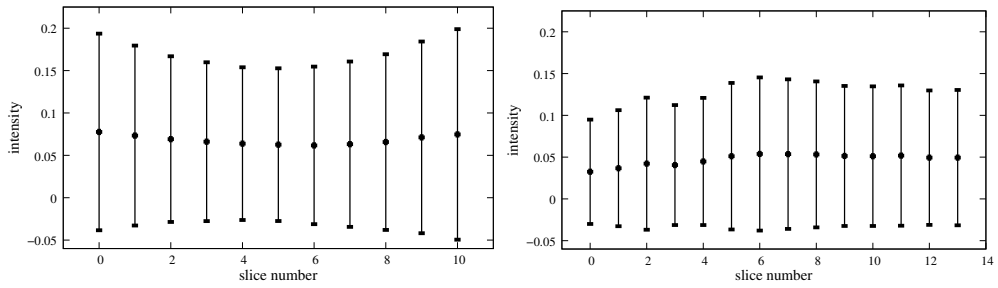


Figure 5.4: The mean values and standard deviations of texture in individual slices of data sets 1 (left) and 5 (right).

Chapter 6

Results

In this chapter we present results obtained with our implementation. We proceed in the following fashion:

1. We review different error measures in section 6.1.
2. In section 6.2 we present the results obtained with models which were matched to all data sets. The intention is to show how the considered models generalize to a large number of data sets.
3. In section 6.3 we pick out a single data set and examine how well it can be matched using AAMs. Several parameters are varied and the influence on the matching quality is inspected. Compared to the quantitative results we give more detailed information about the exact matching behavior.
4. The results obtained with local AAMs are presented in section 6.4.
5. Finally, performance issues are discussed in section 6.5 and the results are summarized in section 6.6.

6.1 Error Measures

To evaluate the quality of a specific model matching process it is important to be able to measure the difference between the target (the unsegmented volume data) and the model. There are basically two different types of error measures:

- **Differences in shape** can only be measured if the annotation of the target is explicitly known.
- **Differences in texture** can be calculated without knowing the shape of the target.

The first class of difference measurement is much more interesting for segmentation. This simply is the case because a successful segmentation is desired to identify the geometrical features (the shape) of the unknown data. The quality of the

matched texture is of minor interest. Also for segmentation of the left cardiac ventricle the geometrical accuracy is of major interest.

However, comparing the matched shape to the real shape presumes that the real shape is known. This is only the case for already segmented data. When we perform a leave-one-out test with annotated data we can calculate this measure. In order to estimate the quality of a match applied to completely unknown data, only differences in texture can be calculated. Later in this chapter, when we present some of our results for model matching, we will use both a measure for differences in shape and a measure for differences in texture. Next we introduce and explain the most commonly used error measures.

Procrustes distance

The Procrustes distance describes the distance between two shapes in terms of squared distances between pairs of corresponding points. The Procrustes distance is defined by

$$PCD = \sum_{i=1}^{N_x} \|\boldsymbol{x}_i - \boldsymbol{y}_i\|^2. \tag{6.1}$$

Average point-to-point distance

The average point-to-point distance can be understood more intuitively. It is the sum of distances of corresponding points divided by the number of points N_x:

$$APP = \frac{1}{N_x} \sum_{i=1}^{N_x} \|\boldsymbol{x}_i - \boldsymbol{y}_i\|. \tag{6.2}$$

Average point-to-surface distance

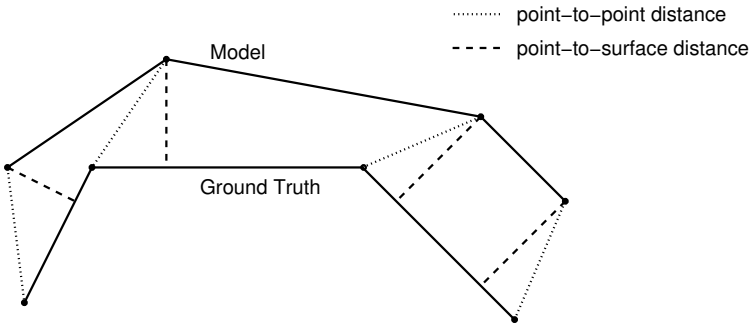


Figure 6.1: Point-to-point and point-to-surface distances.

The point-to-point distance has one little disadvantage. Shapes defined for a 3D AAM mainly consist of surfaces. When manual annotation is used, usually only few

distinctive anatomical points are identified. To increase the number of landmark points, intermediate points are added. These intermediate landmark points may slide a little upon a surface without changing the position or shape of the surface.

Imagine the point-to-point distance is calculated for two shapes which are very much the same. Although the shapes might overlap perfectly well, the point-to-point distance can be relatively large since the shift between pairs of points along the surface is included. For segmentation purposes the exact distance of pairs of points is not as important as the actual difference or distance of two shapes. For this reason another measure, the average point-to-surface distance (figure 6.1), is introduced. This is the average distance of surface points in the model to the surface of the ground truth:

$$APS = \frac{1}{N_x} \sum_{i=1}^{N_x} d_s(\boldsymbol{x}_i, s), \tag{6.3}$$

where $d_s(\boldsymbol{x}, s)$ denotes the distance between a point $\boldsymbol{x}$ and a surface mesh s. Here we consider only triangle meshes. The distance $d_s(\boldsymbol{x}, s)$ then is equal to the minimum distance between $\boldsymbol{x}$ and the closest triangle to $\boldsymbol{x}$ in s.

One could invert the roles of model shape and ground truth shape. In other words it would be possible to calculate the average distance of the ground truth points to the surfaces of the model. Although this is not the same it should result in a similar distance value and the failure should in general be smaller than the failure introduced by the assumption that surfaces can sufficiently well be approximated by triangle meshes.

If no shape information is available one has to revert to pure texture information to estimate the quality of a match. The following two measures express the difference between model and target in terms of texture.

Average texture error

The average texture error is the mean deviation of model and target texture values [Stegmann, 2000]:

$$ATE = \frac{1}{N_g} \sum_{i=1}^{N_g} |g_{V,i} - g_{m,i}|, \tag{6.4}$$

where $g_{V,i}$ and $g_{m,i}$ denote the i-texture samples of the volume data and the model respectively. N_g denotes the number of texture samples contained in the model.

Root Mean Square Error of Texture

The root mean square error of texture differences is also used by some authors [Baker and Matthews, 2001]. It is calculated as:

$$RMS = \sqrt{\frac{1}{N_g} \sum_{i=1}^{N_g} (g_{V,i} - g_{m,i})^2}. \tag{6.5}$$

The RMS involves a sum of squared values. This is why few large texture errors increase the RMS more than many small differences. In other words the RMS accounts better for individual high differences than ATE.

6.2 Quantitative Results

The quantitative results comprise tests carried out for all data sets. The results of these tests show how well a model generalizes to a large number of data sets. The quantitative results comprise three tests:

1. A model built from all 32 data sets is matched to all data sets (leave-all-in). The results obtained with this test show how good data sets are matched when these data sets are contained in the training set.

2. A total of 32 models (each built from 31 data sets) are matched to the one data set which was left out (leave-one-out). This type of test reflects how well the model can be matched to unknown data which is not included in the training set.

3. A model built from 15 data sets is matched to all data sets (leave-15-in). The 15 data sets were manually identified as qualitatively good with respect to texture and shape. We carried out this test in order to find out how the removal of statistical outliers affects the matching quality.

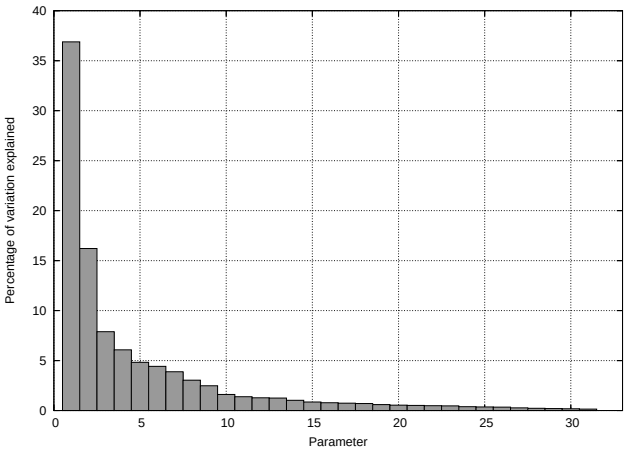
The quantitative results are intended to give a rough survey of how well the algorithm works in general when applied to a large number of data sets. In section 6.3 on qualitative results we will inspect the behavior of matching processes for individual data sets in detail.

6.2.1 Leave-All-In

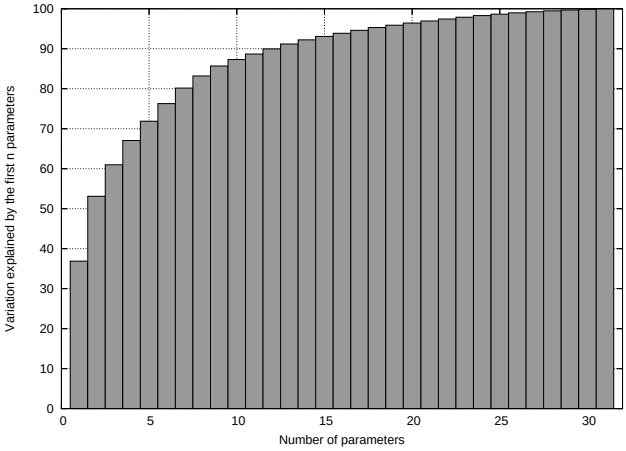
The model including the largest amount of variations that we were able to build was a model including all 32 available data sets. Before we discuss the matching quality of this large model we present some interesting properties of the model itself.

The amounts of variation explained by model parameters are illustrated in figure 6.2. Figure 6.2(a) shows the percentage of variance explained by the individual parameters. As it is characteristic of PCA the first parameter relating to the largest eigenvalue describes the largest separate amount of variance (36.9%). The amount of variance described by the subsequent parameters decreases significantly. Figure 6.2(b) illustrates the amount of variance explained by the sum of the first n parameters. Ten parameters (about a third of all parameters) explain 87.3% of the total variation found in the training set.

Figure 6.3 shows the variations of shape for the first three modes of variation of this model. The parameters illustrated in the figure each were changed by $\pm$ 1.5 times the standard deviation $\sigma = \sqrt{\lambda}$.



(a) The amount of variance explained by the individual parameters

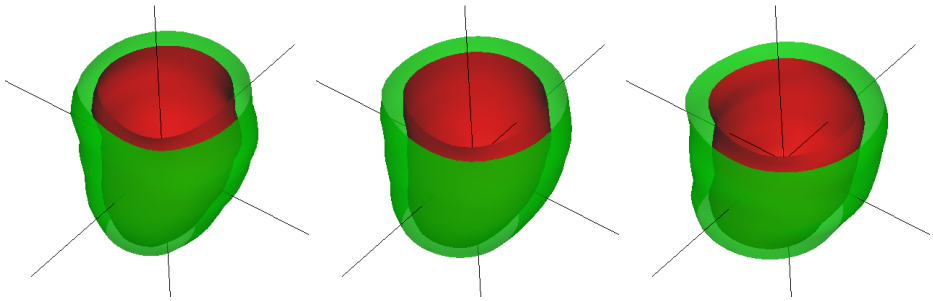


(b) The amount of variance explained by the sum of the first n parameters

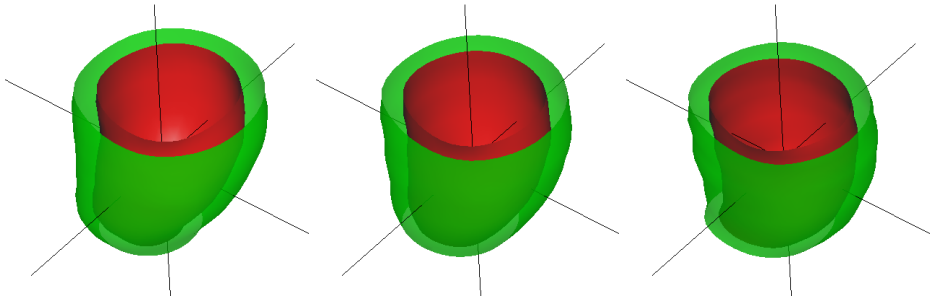
Figure 6.2: Variance explained by model parameters.

Figure 6.4 shows the training instances projected to model parameter space (parameters 1 and 2 in figure 6.4(a) and parameters 1 and 3 in figure 6.4(b)). Every point in the figure represents one instance in the training set.

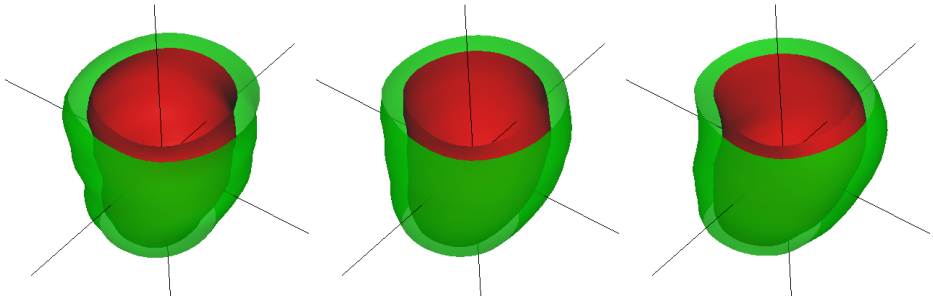
The texture resolution for this model was chosen such that the model included 14,168 individual texture samples. This relates to a grid spacing of about 3 millimeters (depending on the data set). For the tests the matching processes were limited to a maximum of 15 iterations.



(a) 1st mode of variation ($\pm 1.5\sigma$)

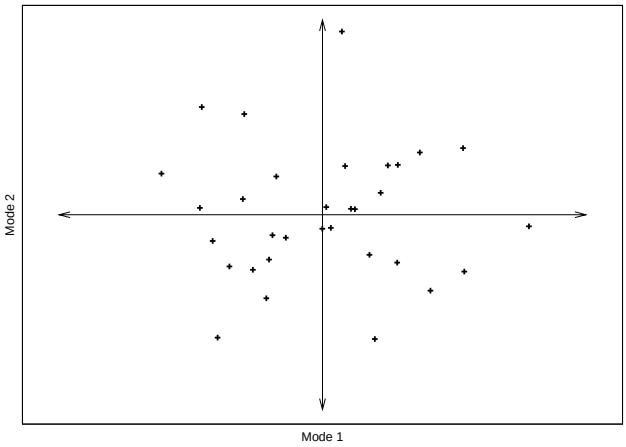


(b) 2nd mode of variation ($\pm 1.5\sigma$)

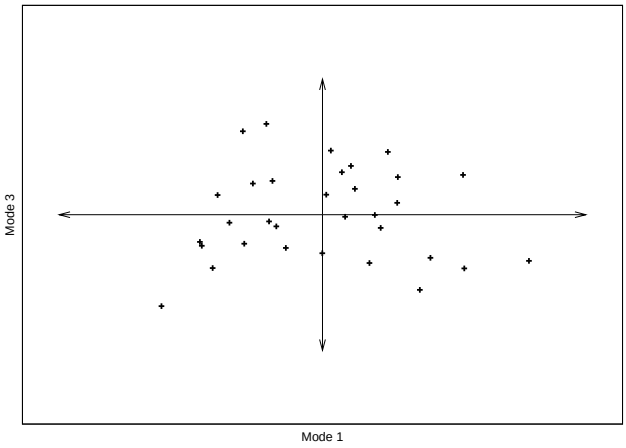


(c) 3rd mode of variation($\pm 1.5\sigma$)

Figure 6.3: The first three modes of variation with respect to shape.



(a) The complete training set projected to the hyper-plane spanned by the first two modes of variation



(b) The complete training set projected to the hyper-plane spanned by the first and third mode of variation

Figure 6.4: Projected model parameters of the training data sets.

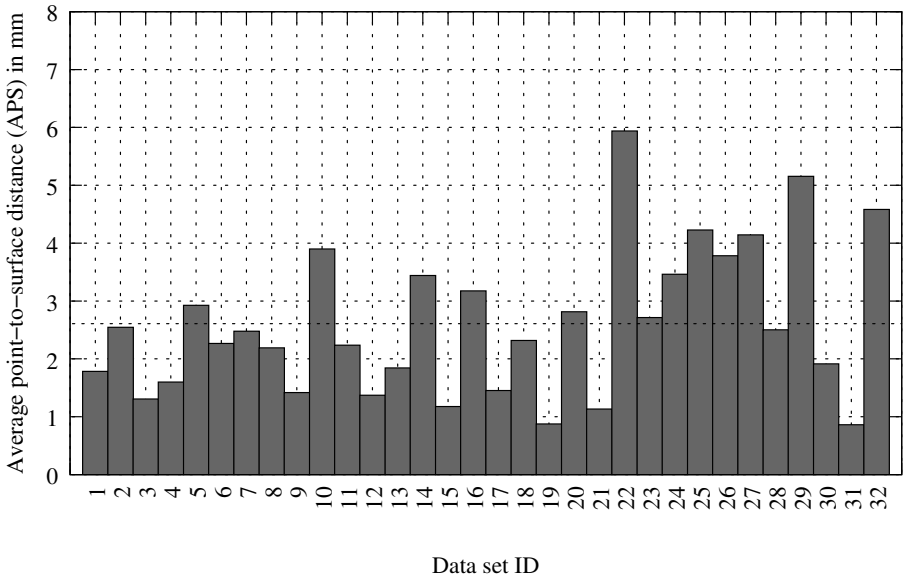


Figure 6.5: Result of a leave-all-in test from all 32 data sets.

We have already mentioned that for segmentation purposes the shape error measure of the average point-to-surface (APS) seems to be most interesting. Thus we review the results mainly with respect to APS. Matching the model to each of the training data sets using 10 modes of variation gives the results illustrated in figure 6.5. The average quality over all data sets in terms of APS is 2.61mm. This is about two times the sampling frequency upon slices of the original data. The two best matches were achieved for data sets 31 (APS = 0.86mm) and 19 (APS = 0.87mm). A total of 10 data sets could not be matched with an accuracy below 3.0mm APS. These matches can be considered complete failures.

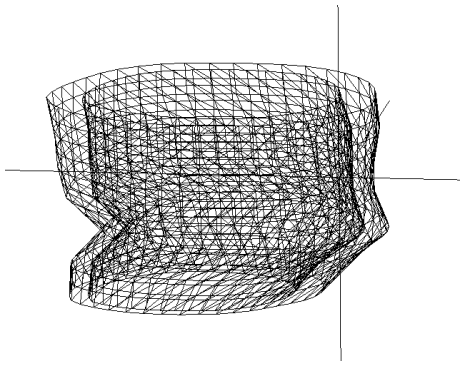


Figure 6.6: Shape of data set 10.

The reason for failed matches – here as well as in the following tests – seem to be strong statistical deviations. For example data set 10 is not matched very well. Figure 6.6 shows the shape of this data set after interpolation to 13 slices. It seems to be the case that this data set includes a strong shift of slices due to patient movement

during data acquisition. Further more this data set comprises only 7 slices of texture data. Only two other data sets also contain only 7 slices, all other data sets include denser volume data information (up to 13 slices). In other words the shape of data set 10 is a statistical outlier and texture information is not very precise. Indeed this relatively poor quality of data set 10 seems to be the reason for the bad behavior of model matching achieved for it.

6.2.2 Leave-One-Out

So far we have reported properties of a single model built from all 32 available data sets. Such a complete model should be able to match all examples in the training set very well as long as a sufficiently large number of model parameters is used. To better estimate the quality of the method it is necessary to apply the model matching process also to data which is not used for building the model. Such a situation is much more realistic for real world applications.

We carried out a classical leave-one-out test by building 32 different models each one derived from 31 instances in the training set. For each model the attempt was made to match it to the one data set that was not used for building the considered model. The first model was built from all data sets except the first one. The second model was built from all data sets except the second one and so on. Then model number one was matched to data set number one, model two to data set two and so on. The purpose of such a leave-one-out test is to show how well the models generalize to unseen data.

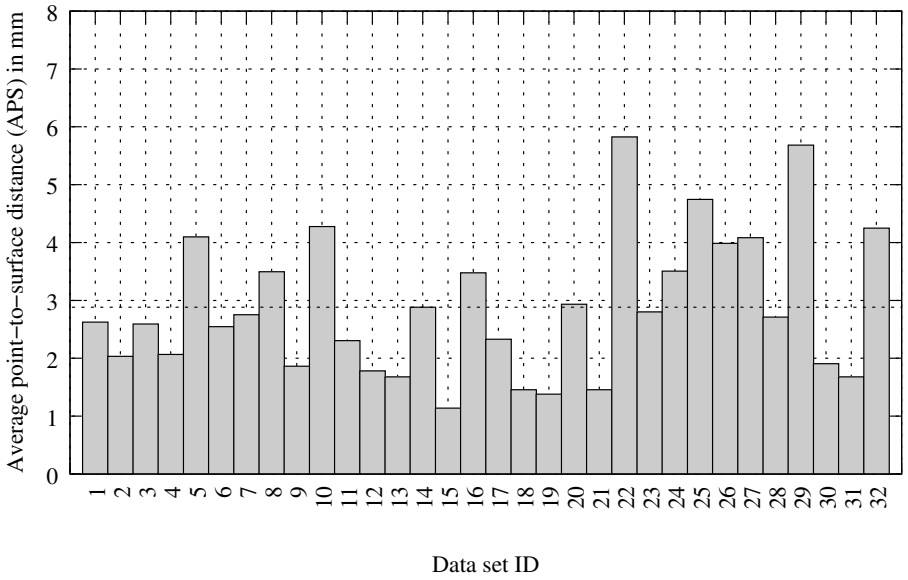


Figure 6.7: Result of the leave-one-out test for all 32 data sets. For each data set a model was built from all other training examples and matched to it.

As for the leave-all-in test above again a total of 10 model parameters were

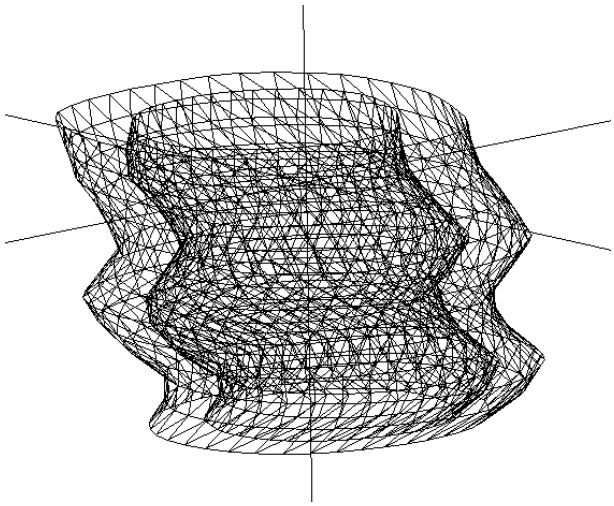


Figure 6.8: Shape of data set 8. This data set shows strong distortions of shape which might be a result of patient movement during acquisition.

used. Figure 6.7 shows the best match with respect to APS for each data set. The very best matches were obtained for data sets 15 (APS = 1.14mm) and 19 (APS = 1.37mm). This time 11 data sets produced a match with an APS larger than 3.0mm. The average APS was 2.883mm. Compared to the leave-all-in test the leave-one-out test produced worse results as it was expected. The reason for this is that the models this time were matched to completely unknown data. The data set to which a model was matched was not contained in the training set used to build this model.

6.2.3 Removing Outliers

We have described properties of large models built from more than 30 data sets. A closer look at individual data sets shows that relatively much qualitatively bad data is included in our test data. Figure 6.8 shows the annotation (after interpolation to 13 slices) of data set 8. It seems as if the patient moved during acquisition and individual slices were shifted relatively to each other. Such distortions appear not only for shape but also for texture as we have already discussed in chapter 5.4.3. Furthermore, some data sets do not represent exactly the same portion of the heart. Because of this the correlation of landmark points suffers heavily. In general one wants to filter out all these negative effects in order to build a model which is more compact and accurate. This was the motivation for us to test another model incorporating fewer but qualitatively better data sets.

We decided to separate the good data sets from the bad ones by manually inspecting the data. 15 data sets which obviously did not include strong distortions were selected. From these 15 data sets we built one model and matched it to all data sets (including the ones used for model building). This small model explains the relative amount of its variation with fewer modes of variation. This is why we used only 8 modes for model matching.

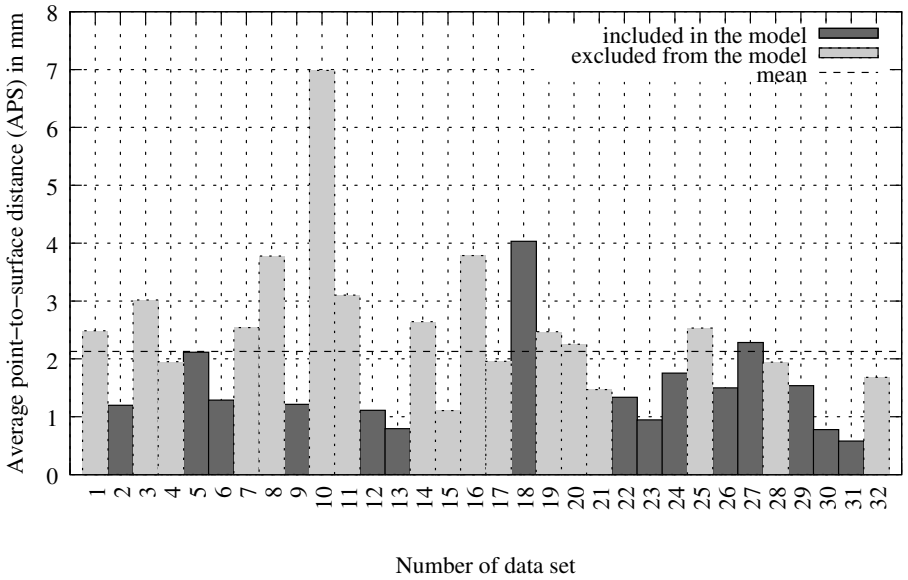


Figure 6.9: Matching all data sets to a model built from 15 instances.

Figure 6.9 shows the best matches with respect to APS. The dark instances are those used to build the model. It can be seen that this smaller and compacter model matches even better than the model containing all data sets. Only 6 data sets led to an APS larger than 3.0mm. The average best APS for this model was 2.13mm. Table 6.1 summarizes all the quantitative results. The small and compact leave-15-in model seems to behave best.

	leave-all-in	**leave-one-out**	**leave-15-in**
average APS	2.61mm	2.88mm	2.13mm
# data sets with APS $>$ 3.0mm	10	11	6

Table 6.1: Quantitative results.

6.3 Qualitative Results

Above we have considered models which were matched to all data sets. In this section we concentrate on more details of individual AAM searches applied to selected single data sets. The intention is to show how different parameters influence the matching process.

6.3.1 Leave-One-Out

From the quantitative results it can be learned that data set number 13 leads to better matching results than data set number 18. The reason for this difference in quality of matches seems to be that data set 13 explains its appearance by modes that represent statistically frequent details. Data set 18 on the other hand seems to comprise statistically rare features.

Both data sets 13 and 18 were taken from the set of 15 data sets manually identified as qualitatively good. A leave-one-out test was carried out such that for both data sets a model of the remaining 14 data sets was built and then matched with the one that was left out.

For both data sets multiple AAM searches were performed which differ in the initial displacements of the model's position. Figure 6.10 shows the progress of matching in terms of APS. For all initial positions the model converges on data set 13. The matching of data set 18 proceeds not so stable and diverges for two of the four tests.

In the following we present results of matching the model from 14 good data sets to data set 13. In figure 6.11 the model initially was displaced by 15mm in direction of the X- and by 30mm in direction of the Y-axis. Red color represents a point-to-surface distance of 10mm or more. Blue indicates a point-to-surface distance of 5mm and green a distance of 0mm. Other color values are interpolated accordingly. The color coded surface distance is only calculated for individual model points and not over the whole surface. Colors are smoothly interpolated between points in the mesh. The black wire frame represents the shape of the ground truth for the considered data set 13.

Figure 6.12 shows the converged model together with the ground truth. Endo- and epicardium of model and ground truth are shown separately.

Figure 6.13 illustrates the differences in texture of the same matching process using a volume visualization. The three volumes shown are again the difference volumes before the first, after the 4th and after the 8th (converged) iteration.

The same matching process is visualized differently in figure 6.14. Three slices were interpolated for the initial model placement, after the 4th iteration and after the 8th (converged) iteration. The data together with the model superimposed on it are shown. The bottom row shows the original data with the shape of the converged model.

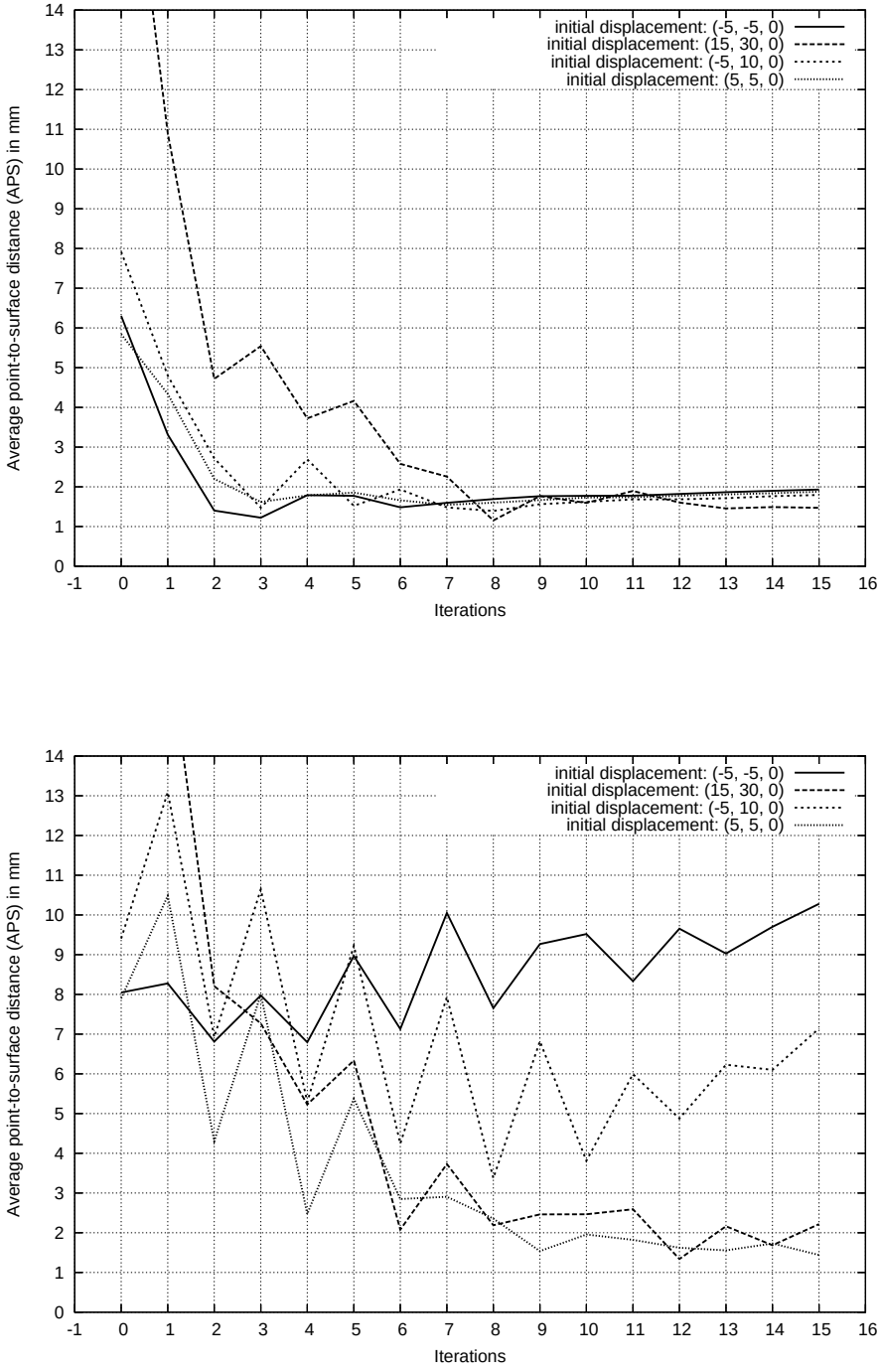


Figure 6.10: Progress of APS in AAM search for data sets 13 (top) and 18 (bottom). After convergence both APS and ATE continue to vary slightly. The reason for this is noise in the linear regression. One can see that the model on data set 13 converges well for all initial displacements. The test on data set 18 leads to worse results.

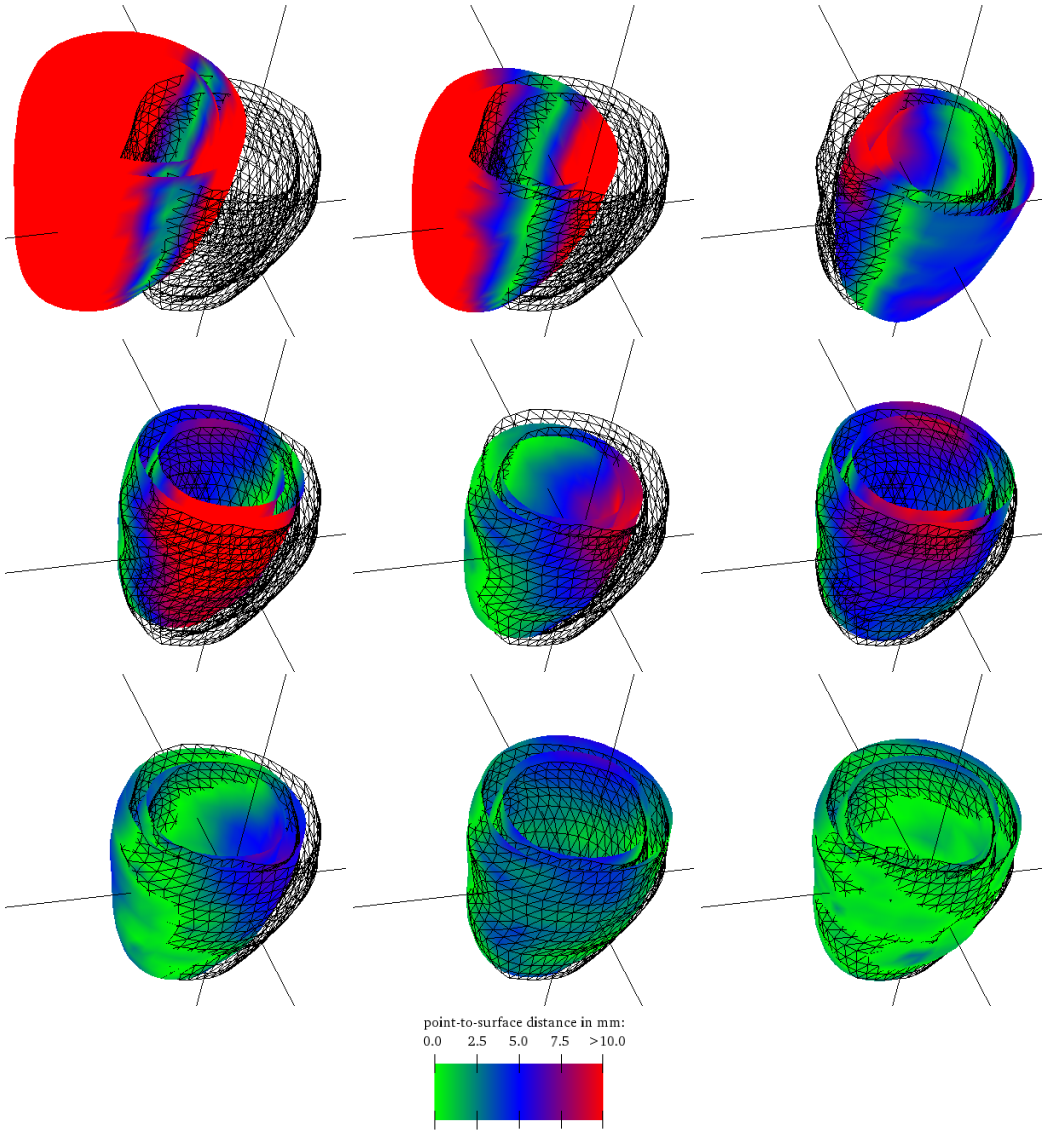


Figure 6.11: Matching data set 13 (all iterations) with a model built from 14 data sets not including data set 13. The matching process starts at the image on the top left and ends at the right bottom. Each image represents one iteration.

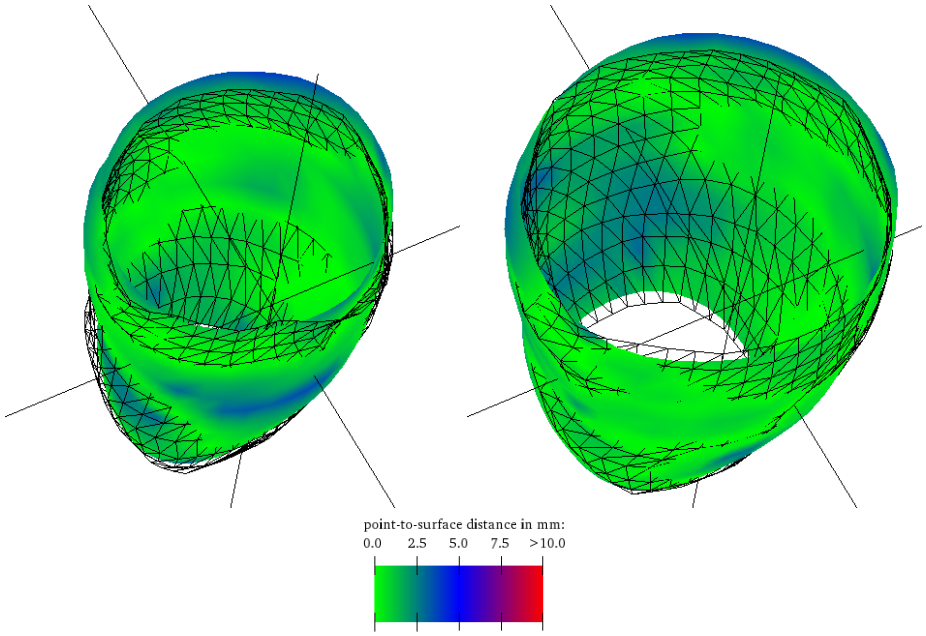


Figure 6.12: Matching data set 13. The Result is shown for endocardium (left) and epicardium (right) separately.

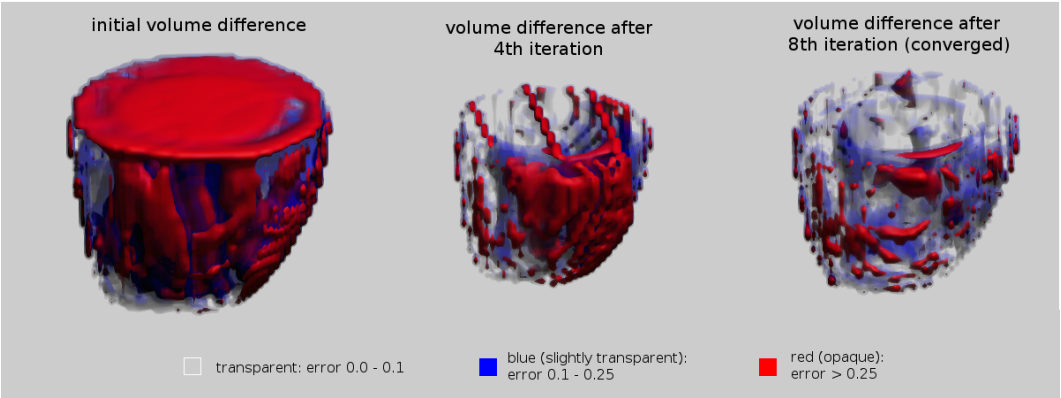


Figure 6.13: Matching data set 13 (difference volumes). This figure shows the initial difference volume, the difference volume after 4th, and after 8th (converged) iteration.

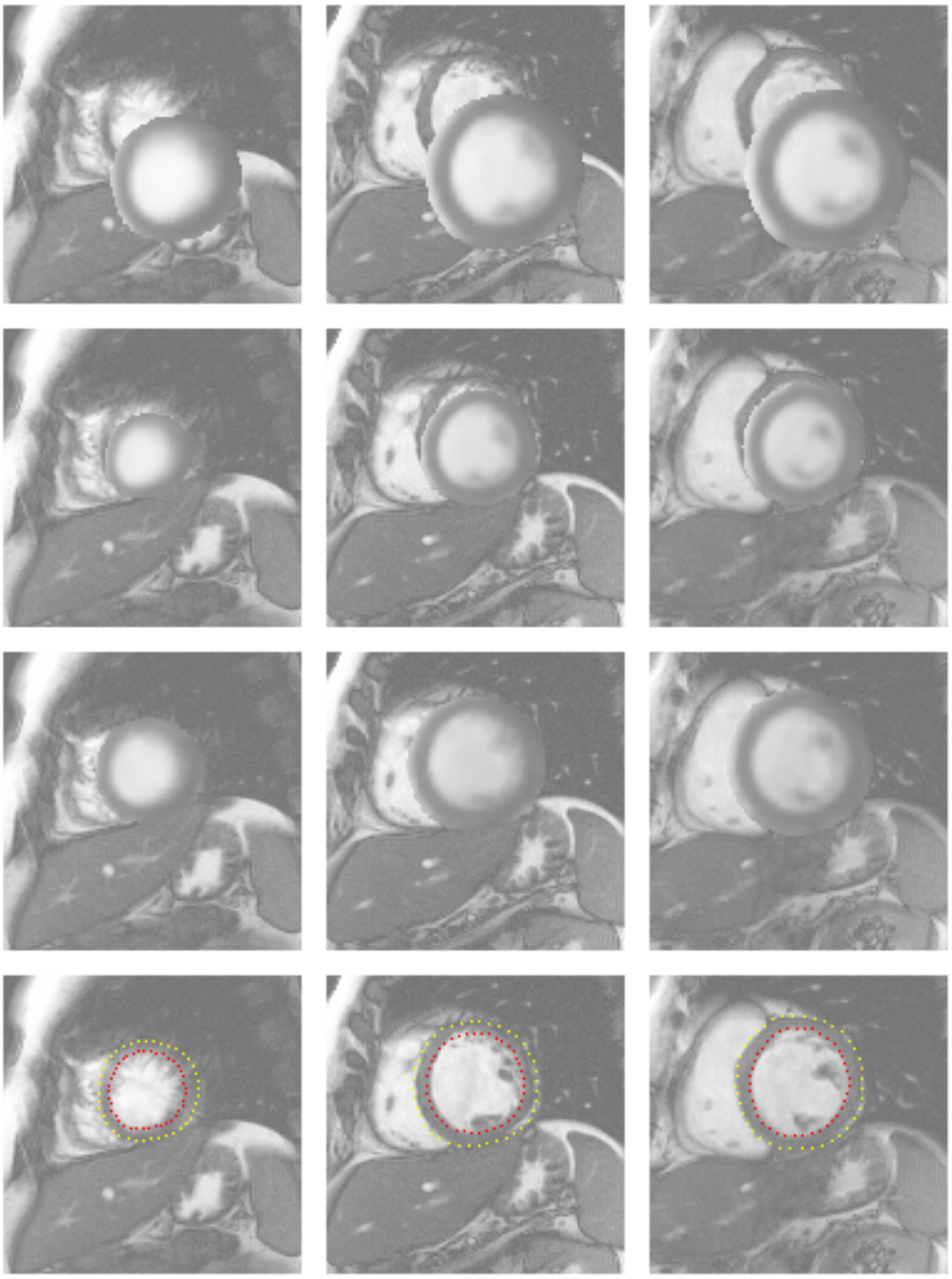


Figure 6.14: Matching data set 13 with slice-wise texture visualization. The top row shows three slices of data with the initial model superimposed. The second and third rows show the model after the 4th and 8th (converged) iteration respectively. The bottom row shows the data with the matched model's shape points.

6.3.2 Variation of Texture Accuracy

The amount of texture which is used has a great influence on how fast an AAM can be built and matched. It is thus very interesting to see how texture accuracy influences the quality of AAM search. We built several models from the 14 good data sets, again leaving out only data set 13. The model was built several times with exactly the same settings but with different numbers of texture samples. Figure 6.15 illustrates the individual matching processes. The initial matching position was disturbed by 20mm, -15mm, and 0mm in X-, Y-, and Z-direction. It can be seen that with at least about 2400 samples the AAM search still converges very well. With 800 texture samples, however, convergence is not satisfying. Table 6.2 lists the numbers of texture samples in the model together with the according sample spacing and time needed for a single matching iteration (see section 6.5 for more details on performance).

# texture samples	Spacing of samples	Time per iteration
62,532	~ 1.8 mm	~ 2.65 sec.
13,911	~ 3.0 mm	~ 1.40 sec.
2,398	~ 5.4 mm	~ 0.86 sec.
803	~ 7.8 mm	~ 0.82 sec.

Table 6.2: Grid spacings with according number of texture samples and time used for matching.

6.3.3 Varying the Number of Modes

The chosen number of parameters or modes of variation determines the deformability of a model. The more modes are used the higher is the model's capability to match unseen data. We carried out several matching experiments where we used different numbers of core model parameters. The core model parameters are those calculated by PCA. The core model parameters do not include position, orientation and intensity of the model.

It turned out that for very representative data sets the core model parameters are not that important. Data set 13 is such a representative data set. With 8 core model parameters an APS of 1.50mm could be achieved. With no core model parameters the matching was only a little worse with an APS of 1.61mm (figure 6.16). The reason for the good quality of matching in this case can be explained by the fact that the considered data set has a very frequent appearance and thus can already very well be approached by the mean model.

In general the influence of the number of modes on the quality of matching depends strongly on the data set. Imagine a data set that includes a shift of slices which does not appear frequently. If a model is built from other data sets and matched with the one with the rare variation, an increase in model parameters would not help since the training set simply does not include the variation. This again is a motivation to reduce variances in the data before running an AAM search on it.

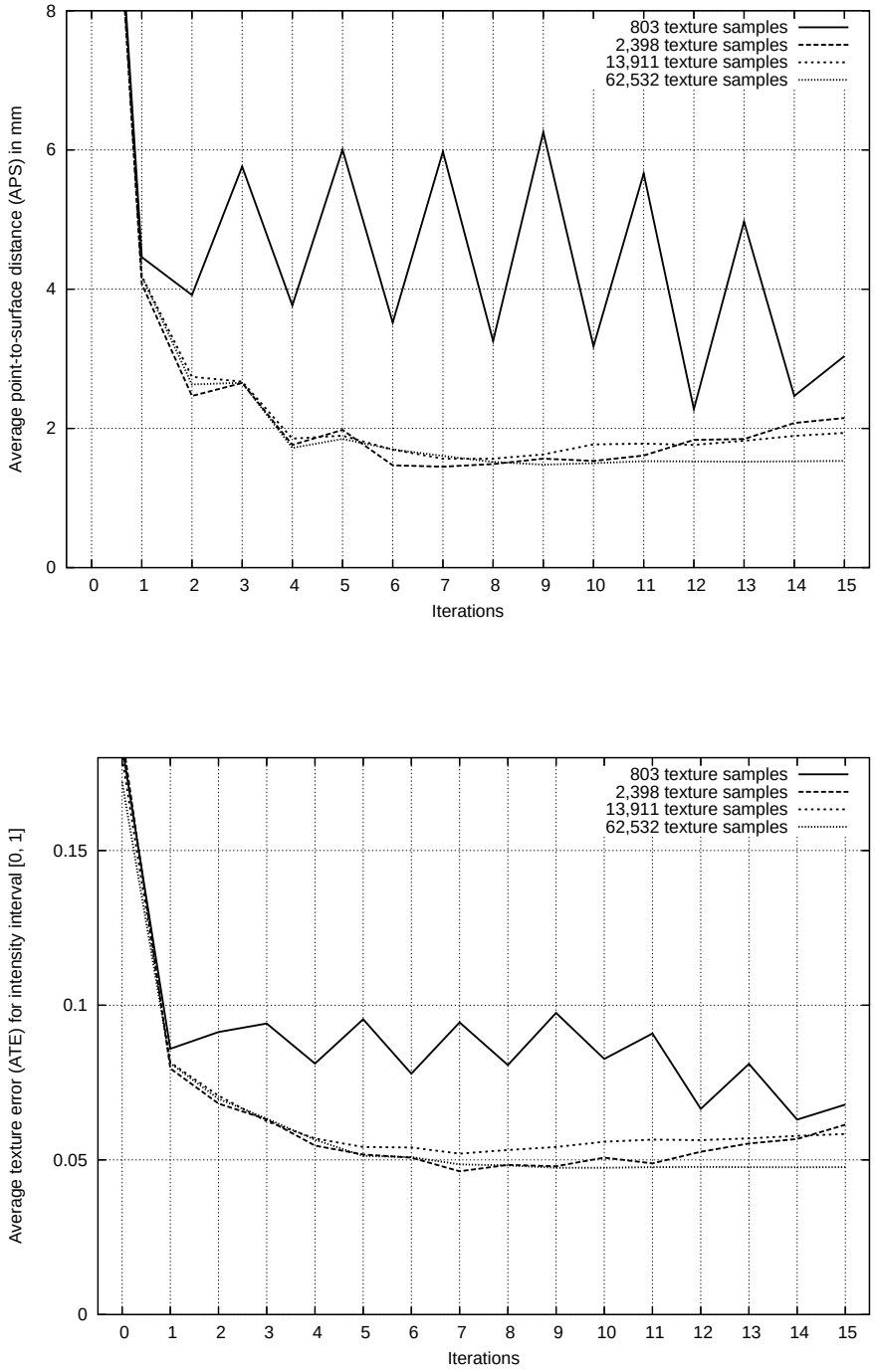


Figure 6.15: Progress of matching the model to data set 13 with different texture resolutions and with respect to APS (top) and ATE (bottom).

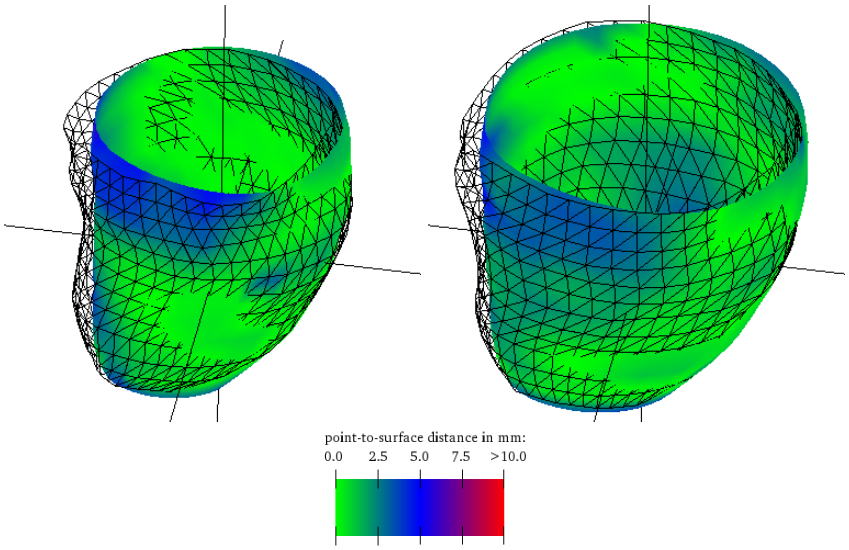


Figure 6.16: Matching data set 13 with no core model parameters. It can be seen that the optimal match achieved has relatively bad quality due to the lack of deformability of the model.

6.3.4 Compact Training of AAM Search

Training the AAM search means to calculate differences in parameters and textures and to find a correlation between them. The standard approach is to calculate changes in model parameters and textures for each and every example in the training set.

Above we have shown that when a large training set is used it makes sense to remove statistical outliers or data that has bad quality. This is done to increase the quality of the model. The same should hold for the training of AAM search. The idea is to use not all the training sets to calculate regression for the AAM search. Only the most representative ones are taken into account. Only training sets that contain qualitatively good texture should be used for calculating the regression.

We trained the AAM search for one model using only two data sets. With the compact regression it turned out that the accuracy of matches increased. Thus data set 13 could be matched with an accuracy of 0.84mm APS. The matching is visualized in figure 6.17.

Not only the accuracy of the AAM search is increased with a smaller number of training sets. Also the computational effort is decreased. Setting up the data for the regression is the most time-consuming task when building a model. With a smaller number of data sets considered, the complete creation of a model (including calculation of regression) can be done significantly faster.

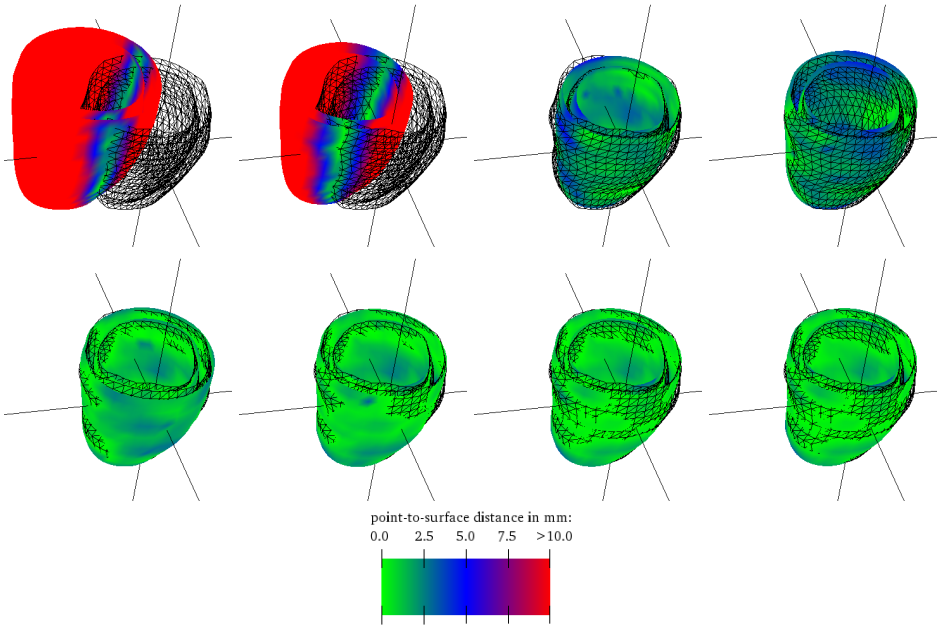


Figure 6.17: Matching the compactly trained model with data set 13. After the 8th iteration convergence with an APS of 0.84mm is reached.

6.4 Local AAMs

In principle an AAM should reproduce all shape and texture variances that appear in the training set. The problem with this is that statistical outliers cannot be matched correctly because they differ too much from the instances from which the model was built. Also if there are only few data sets available for putting up a training set, the resulting model is not general enough. In both cases it is desirable to increase the model's flexibility using other strategies.

In section 4.6.1 we have discussed local AAMs [Taylor and Cootes, 2000]. There are other ways to increase the model's flexibility, for example using mechanical deformations. However, local AAMs fit very well to the framework of standard AAMs since they use similar techniques. The idea is to locally deform the model and to learn the correlation between local deformations and local texture differences. In fact for every point in the shape model the change of texture in its surrounding is observed as its position changes. To maintain a nice shape only smooth deformations are considered. The idea is to use local deformations when the standard AAM search with its global deformations is not able to improve a match.

Figure 6.18 shows local smooth deformations applied to our model of the left ventricle. In the previous sections we have shown that most data sets can be matched with an accuracy of about 2mm. Thus the ideal magnitude for local deformations seems to be of a magnitude of not more than 4 or 5mm. A match of about 2mm APS could then be achieved by standard AAM search and refined by local AAMs. We used random deformations with a magnitude between 0 and 5mm for calculating the local regressions.

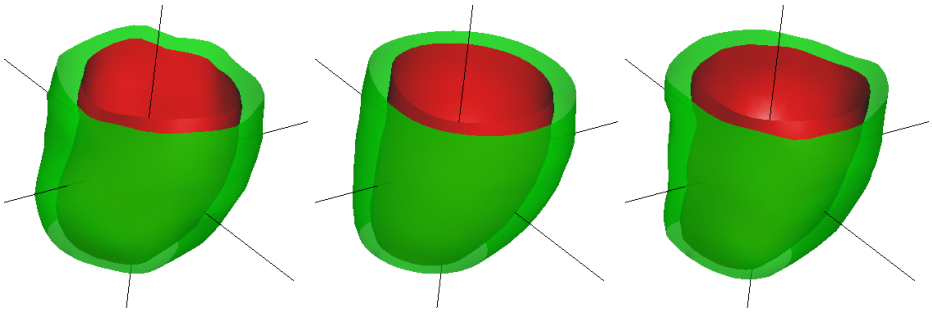


Figure 6.18: Smooth local deformations for local AAMs.

We applied the local AAM strategy to several data sets. It turned out that for most of them the local AAMs lead to only small improvements. For data set 13, for example, the best global match with an APS of 0.84 could be improved to an APS of 0.82mm.

A great problem of local AAMs is that if the data is noisy the local linear regression does not work very well. Individual points deform differently even though the deformation is smoothed. In other words the model's shape gets very implausible. The robustness, which is a typical property of standard AAMs, is lost if only local texture is considered.

6.5 Performance Issues

The tests were carried out on a notebook with a Mobile AMD Athlon XP 2500+ (1.867GHz) processor and 512 MB RAM. Most of the computational effort comes from the resampling of texture values and thus depends on the amount of texture used. In table 6.2 the approximate times needed for a single iteration in an AAM search are listed on the right.

An average AAM search converges after about 8 iterations. The standard resolution which we used was 3.0mm spacing of texture samples. In this case a complete AAM search takes about 11 seconds.

What has to be mentioned is that we calculate barycentric coordinates on the fly as the model volume is traversed. Of course this information could also be precalculated since it does not change as the model is deformed. In this case the matching time should even be decreased.

6.6 Summary

We have shown that AAMs can be used to robustly segment volumetric cardiac data. Depending on the considered data set, our 3D AAMs align with an accuracy between about 0.9mm and 2.5mm relative to the manually annotated surfaces. Recall that the data itself is captured with an accuracy of about 1.3mm upon slices and

an inter-slice distance of about 7.3mm. Under these circumstances an accuracy of about 1.5mm can be considered quite good.

We presented the behavior of models applied to all data sets (quantitative results) and analyzed individual matching processes of individual data sets (qualitative results). Additional issues such as influence of the amount of texture were also discussed and the according results of model matching were shown.

Chapter 7

Conclusion and Future Work

In this thesis we have discussed segmentation of 3D cardiac MRI data. We have outlined the importance of fast automatic and semiautomatic segmentation of such data. We shortly reviewed the anatomical background and outlined special properties of cardiac MRI data.

The theory of building, training and matching AAMs was discussed with focus on 3D AAMs for volume data. The successfully achieved goals of this thesis are listed below.

- We gave a survey of the theory behind 3D AAMs and tried to show the advantages of using statistical methods for building robust deformable models. We reviewed the idea of AAMs to incorporate both shape and texture into a model and mentioned several extensions to the standard AAM. Furthermore the problem of model matching was discussed. We also showed how AAM search can be done using linear regression of model parameter changes and texture changes.

- We discussed our implementation of 3D AAMs and roughly outlined its structure.

- Finally we presented the results obtained for a model of the left cardiac ventricle. We showed that for the majority of our 32 cardiac MRI data sets the AAM search converged relatively fast and with high accuracy.

In general there is a great variety of approaches for automatic segmentation. Often it depends very much on the problem domain and additional constraints – such as the claimed performance or accuracy – to determine which method is best. To better identify their role among other segmentation techniques we now give a list of the advantages and disadvantages of AAMs.

Advantages

+ The strong statistical background of AAMs in general makes them very robust even if the quality of analyzed data is bad. The prior information about an object's appearance helps to compensate missing texture.

+ The result of segmentation with AAMs always is valid with respect to the training set that was used. Contrary to other deformable models an AAM changes its appearance only in such a way that implausible segmentations are avoided.

+ User interaction is reduced to a minimum or can even completely be omitted when AAMs are used. If multiple AAM searches are performed on the unseen data no user interaction is needed at all.

+ The convergence of AAM search in general is very fast and we showed that also for 3D AAMs only about 6 iterations are needed to derive an optimum match.

Disadvantages

- In order to use AAMs one has to possess a representative training set. For many applications it is relatively hard to assemble such a training set consisting only of representative examples.

- The training set is only half of the data necessary to build an AAM. The creation of annotations is also of great importance. It is often very hard to get the expert knowledge needed in order to derive good annotations for given data.

- It is important to remove statistical outliers from a training set in order to prevent the model from creating implausible instances. For many applications it is not easy to determine which data sets should be excluded from the model.

- Depending on the type of data that is analyzed the assumption of linearity in model parameters is often hard to justify. When the variation of an object is highly nonlinear, the standard AAM approach can lead to implausible model instances and thus to bad segmentation results.

- The texture deformation in the simplest case is done by affine warping. This requires a dense placing of shape points in order to approximate non-linear deformation. Often it is hard to achieve a large evenly distributed number of shape points. Furthermore the affine warping has the drawback that it can produce kinks in the deformed texture.

- AAMs are not suitable for modelling objects which can only be identified in a special context or which do not have a fixed shape. Therefore the AAM approach cannot be used for e.g. matching clouds, stones, trees, or blood vessels.

- For AAM search with linear regression a linear correlation of texture differences and parameter updates is assumed. This is a very strong simplification and allows only small parameter changes.

- If models with high texture resolution and in higher dimensions, such as volumetric models, are considered, the extraction of texture becomes a bottleneck for the AAM search.

Many of the listed disadvantages are issues of ongoing research. In our implementation we have so far implemented standard AAMs and local AAMs. Future work will be the adaptation, evaluation and improvement of extensions and improvements of AAMs. This will include the following issues:

- Incorporation of techniques for automatic creation and improvement of annotations [Davies *et al.*, 2001; Kotcheff and Taylor, 1998; Hladůvka and Bühler, 2005] in our implementation.
- Sophisticated methods for preprocessing captured data. This is intended to improve the quality of both the model and the matching process.
- Identification and elimination of statistical outliers in the training set.
- Alternative model formulations such as the use of ICA or mixture models. The goal of this is to improve handling of non-linearities [Cootes and Taylor, 1999; Üzümcü *et al.*, 2003].
- Evaluation of alternative techniques to estimate the gradient in the AAM search. For example analytical approaches such as the Lucas-Kanade algorithm [Lucas and Kanade, 1981] could be used.
- Compression of AAM texture in order to reduce the amount of texture data incorporated in the model [Stegmann *et al.*, 2004; Darkner *et al.*, 2004; Wolstenholme and Taylor, 1999].

We finally remark that the interpretation of medical images is a highly critical task with respect to a patient's health. Therefore a visualization tool offering the possibility of automatic segmentation also has to offer a user interface that allows medical doctors to inspect and eventually correct automatically computed segmentations.

Bibliography

Ben Appleton. Optimal geodesic active contours: Application to heart segmentation. In *Workshop on Digital Image Computing*, pages 27–32. University of Queensland, 2003.

Simon Baker and Iain Matthews. Equivalence and efficiency of image alignment algorithms. In *Proceedings of Computer Vision and Pattern Recognition (CVPR)*, volume 1, pages 1090–1097, 2001.

Paul J. Besl and Neil D. McKay. A method for registration of 3D shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239–256, 1992.

Ella Bingham and Aapo Hyvärinen. A fast fixed-point algorithm for independent component analysis of complex valued signals. *International Journal of Neural Systems*, 10(1):1–8, 2000.

Fred L. Bookstein. Thin-plate splines and the atlas problem for biomedical images. In *Proceedings of Information Processing in Medical Imaging (IPMI)*, volume 511 of *LNCS*, pages 326–342, 1991.

Fred L. Bookstein. Landmark methods for forms without landmarks: Morphometrics of group differences in outline shape. *Medical Image Analysis*, 1(3):225–243, 1997.

Johan G. Bosch, Steven C. Mitchell, Boudewijn P. F. Lelieveldt, Francisca Nijland, Otto Kamp, Milan Sonka, and Johan H. C. Reiber. Automatic segmentation of echocardiographic sequences by active appearance motion models. *IEEE Transactions on Medical Imaging*, 21(11):1374–1383, 2002.

David Breen, Igor Guskov, John Wood, Leonid Zhukov, and Zhaosheng Bao. Dynamic deformable models for 3D MRI heart segmentation. In *Proceedings of SPIE Medical Imaging*, pages 1398–1405, 2002.

Alan D. Brett and Christopher J. Taylor. A method of automated landmark generation for automated 3D PDM construction. In *Proceedings of the British Machine Vision Conference (BMVC)*, 1998.

Manuel D. Cerqueira, Neil J. Weissman, Vasken Dilsizian, Alice K. Jacobs, Sanjiv Kaul, Warren K. Laskey, Dudley J. Pennell, John A. Rumberger, Thomas Ryan, and Mario S. Verani. Standardized myocardial segmentation and nomenclature for tomographic imaging of the heart. *Circulation*, 105(4):539–42, 2002.

CGAL. `http://www.cgal.org`, May 24 2005. Computational Geometry Algorithms Library.

CLAPACK. `http://www.netlib.org/clapack/`, May 24 2005. C-version of the linear algebra package LAPACK.

Laurent D. Cohen. On active contour models and balloons. *Computer Vision, Graphics, and Image Processing: Image Understanding*, 53(2):211–218, 1991.

Timothy F. Cootes and Christopher J. Taylor. A mixture model for representing shape variation. *Image and Vision Computing*, 17(8):567–573, 1999.

Timothy F. Cootes and Christopher J. Taylor. Statistical models of appearance for computer vision. Technical report, University of Manchester, 2000.

Timothy F. Cootes and Christopher J. Taylor. Constrained active appearance models. In *Proceedings of the International Conference on Computer Vision (ICCV)*, pages 748–754, 2001.

Timothy F. Cootes and Christopher J. Taylor. Statistical models of appearance for medical image analysis and computer vision. In *Proceedings of SPIE Medical Imaging*, volume 4322, pages 236–248, 2001.

Timothy F. Cootes, Christopher J. Taylor, D. H. Cooper, and J. Graham. Active shape models – Their training and application. *Computer Vision and Image Understanding*, 61(1):38–59, 1995.

Timothy F. Cootes, Gareth J. Edwards, and Christopher J. Taylor. Active appearance models. In *Proceedings of the European Conference on Computer Vision (ECCV)*, volume 2, pages 484–498, 1998.

Timothy F. Cootes, Gareth J. Edwards, and Christopher J. Taylor. A comparative evaluation of active appearance model algorithms. In *Proceedings of the British Machine Vision Conference (BMVC)*, volume 2, pages 484–498, 1998.

Timothy F. Cootes, Christine Beeston, Gareth J. Edwards, and Christopher J. Taylor. A unified framework for atlas matching using active appearance models. In *Proceedings of Information Processing in Medical Imaging (IPMI)*, pages 322–333, 1999.

Timothy F. Cootes, Gavin V. Wheeler, Kevin N. Walker, and Christopher J. Taylor. Coupled-view active appearance models. In *Proceedings of the British Machine Vision Conference (BMVC)*, volume 1, pages 52–61, 2000.

Timothy F. Cootes, Carole J. Twining, and Christopher J. Taylor. Diffeomorphic statistical shape models. In *Proceedings of the British Machine Vision Conference (BMVC)*, volume 1, pages 447–456, 2004.

Evangelos A. Coutsias, Chaok Seok, and Ken A. Dill. Using quaternions to calculate RMSD. *Journal of Computational Chemistry*, 25(15):1849–1857, 2004.

Surre Darkner, Rasmus Larsen, Mikkel B. Stegmann, and Bjarne K. Ersbøll. Wedgelet enhanced appearance models. In *Proceedings of the 2nd International Workshop on Generative Model Based Vision*, 2004.

Rhodri H. Davies, Timothy F. Cootes, Carole J. Twining, and Christopher J. Taylor. An information theoretic approach to statistical shape modelling. In *Proceedings of the British Machine Vision Conference (BMVC)*, pages 3–11, 2001.

Rhodri H. Davies, Carole J. Twining, Timothy F. Cootes, John C. Waterton, and Christopher J. Taylor. Automatic construction of optimal 3D statistical shape models. In *Proceedings of Medical Image Understanding and Analysis*, pages 77–80, 2002.

Molly M. Dickens, Shaun S. Gleason, and Hamed Sari-Sarraf. Volumetric segmentation via 3D active shape models. In *Proceedings of the 5th IEEE Southwest Symposium on Image Analysis and Interpretation*, pages 248–252, 2002.

Gareth J. Edwards, Christopher J. Taylor, and Timothy F. Cootes. Interpreting face images using active appearance models. In *Proceedings of the International Conference on Face and Gesture Recognition*, pages 300–305, 1998.

Henry C. Edwards. Ladders, moats, and lagrange multipliers. *The Mathematica Journal*, 4(1):48–52, 1994.

Ralph Gross, Iain Matthews, and Simon Baker. Generic vs. person specific active appearance models. In *Proceedings of the British Machine Vision Conference (BMVC)*, 2004.

Tony Heap and David Hogg. Extending the point distribution model using polar coordinates. *Image and Vision Computing*, 14(8):589–599, 1996.

Jiří Hladůvka and Katja Bühler. MDL spline models: Gradients and polynomial reparametrisations. Accepted for *the British Machine Vision Conference (BMVC)*, 2005.

Berthold K. P. Horn. Closed-form solution of absolute orientation using unit quaternions. *Journal of the Optical Society of America*, 4(4), 1987.

Peter J. Huber. Projection pursuit. *The Annals of Statistics*, 13(2):435–475, 1985.

HyperDictionary. `http://searchbox.hyperdictionary.com/dictionary/shape`, May 24 2005.

Aapo Hyvärinen. Survey on independent component analysis. *Neural Computing Surveys*, 2(1):94–128, 1999.

Michael J. Jones and Tomaso Poggio. Multidimensional morphable models. In *Proceedings of the International Conference on Computer Vision (ICCV)*, pages 683–688, 1998.

Michael Kass, Andrew P. Witkin, and Demetri Terzopoulos. Snakes: Active contour models. In *Proceedings of the International Conference on Computer Vision (ICCV)*, pages 259–268, London, 1987.

Aaron C. W. Kotcheff and Christopher J. Taylor. Automatic construction of eigenshape models by direct optimization. *Medical Image Analysis*, 2(4):303–314, 1998.

Boudewijn P. F. Lelieveldt, Steven C. Mitchell, Johan G. Bosch, Rob J. van der Geest, Milan Sonka, and Johan H. C. Reiber. Time-continuous segmentation of cardiac image sequences using active appearance motion models. In *Proceedings of Information Processing in Medical Imaging (IPMI)*, pages 446–452, 2001.

Frederic F. Leymarie and Martin D. Levine. Tracking deformable objects in the plane using an active contour model. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(6):617–634, 1993.

Steven Lobregt and Max A. Viergever. A discrete dynamic contour model. *IEEE Transactions on Medical Imaging*, 14(1):12–24, 1995.

Bruce D. Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision (DARPA). In *Proceedings of the DARPA Image Understanding Workshop*, pages 121–130, 1981.

Stéphane Mallat. A theory for multiresolution signal decomposition: The wavelet representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(7):674–693, 1989.

Iain Matthews and Simon Baker. Active appearance models revisited. *International Journal of Computer Vision*, 60(2):135 – 164, 2004.

Ivana Mikic, Slawomir Krucinski, and James D. Thomas. Segmentation and tracking in echocardiographic sequences: Active contours guided by optical flow estimates. *IEEE Transactions on Medical Imaging*, 17(2):274–284, 1998.

Steven C. Mitchell, Boudewijn P. F. Lelieveldt, Rob J. van der Geest, Johan G. Bosch, Johan H. C. Reiber, and Milan Sonka. Multistage hybrid active appearance model matching: Segmentation of left and right ventricles in cardiac MR images. *IEEE Transactions on Medical Imaging*, 20(5):415–423, 2001.

Steven C. Mitchell, Johan G. Bosch, P. F. Lelieveldt, Rob J. van der Geest, Johan H. C. Reiber, and Milan Sonka. 3D active appearance models: Segmentation of cardiac MR and ultrasound images. *IEEE Transactions on Medical Imaging*, 21(9):1167– 1178, 2002.

Pradit Mittrapiyanuruk, Guilherme N. DeSouza, and Avinash C. Kak. Calculating the 3D-pose of rigid-objects using active appearance models. In *Proceedings of the International Conference on Robotics and Automation*, 2004.

Johan Montagnat and Herve Delingette. 4D deformable models with temporal constraints: Application to 4D cardiac image segmentation. *Medical Image Analysis*, 9(1):87–100, 2005.

André Neubauer and Rainer Wegenkittl. Analysis of four-dimensional cardiac data sets using skeleton-based segmentation. *Journal of WSCG*, 11(1), 2003.

Stanley Osher and James A. Sethian. Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton-Jacobi formulations. *Journal of Computational Physics*, 79(1):12–49, 1988.

Nikos Paragios, Mikael Rousson, and Visvanathan Ramesh. Knowledge-based registration & segmentation of the left ventricle: A level set approach. In *Workshop on Applications of Computer Vision*, pages 37–42, 2002.

Charnchai Pluempitiwiriyawej, José M.F. Moura, Yi-Jen Lin Wu, and Chien Ho. STACS: New active contour scheme for cardiac MR image segmentation. *IEEE Transactions on Medical Imaging*, 24(5):593–603, 2005.

Maximilian Riesenhuber and Tomaso Poggio. Models of object recognition. *Nature Neuroscience*, 3(11 suppl.):1199–1204, 2000.

Flemming F. Rodler. Wavelet based 3D compression with fast random access for very large volume data. In *Proceedings of the 7th Pacific Conference on Computer Graphics and Applications*, pages 108–147, 1999.

Mikael Rousson, Nikos Paragios, and Rachid Deriche. Implicit active shape models for 3D segmentation in MR imaging. In *Proceedings of Medical Image Computing and Computer–Assisted Intervention (MICCAI)*, pages 209–216, 2004.

Sam T. Roweis and Lawrence K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326, 2000.

Peter H. Schoenemann. A generalized solution of the orthogonal procrustes problem. *Psychometrika*, 31(1):1–10, 1966.

Stan Sclaroff and John Isidoro. Active blobs. In *Proceedings of the International Conference on Computer Vision (ICCV)*, pages 1146–1153, 1998.

Stan Sclaroff and John Isidoro. Active blobs: region-based, deformable appearance models. *Computer Vision and Image Understanding*, 89(2-3):197–225, 2003.

James A. Sethian. *Level Set Methods and Fast Marching Methods*. Cambridge University Press, 1999.

Ken Shoemake. Animating rotation with quaternion curves. *ACM SIGGRAPH Computer Graphics*, 19(3):245–254, 1985.

Mikkel B. Stegmann and David D. Gomez. A brief introduction to statistical shape analysis. Technical report, Informatics and Mathematical Modelling, Technical University of Denmark, DTU, 2002.

Mikkel B. Stegmann and Rasmus Larsen. Multi-band modelling of appearance. *Image and Vision Computing*, 21(1):61–67, 2003.

Mikkel B. Stegmann and Dorthe Pedersen. Bi-temporal 3D active appearance models with applications to unsupervised ejection fraction estimation. In *Proceedings of the International Symposium on Medical Imaging*, volume 5747, 2005.

Mikkel B. Stegmann, Bjarne K. Ersbøll, and Rasmus Larsen. FAME - A flexible appearance modelling environment. *IEEE Transactions on Medical Imaging*, 22(10):1319–1331, 2003.

Mikkel B. Stegmann, Søren Forchhammer, and Timothy F. Cootes. Wavelet enhanced appearance modelling. In *International Symposium on Medical Imaging*, volume 5370, pages 1823–1832, 2004.

Mikkel B. Stegmann. Active appearance models: Theory, extensions and cases. Master's thesis, Informatics and Mathematical Modelling, Technical University of Denmark, DTU, 2000.

Mikkel B. Stegmann. Analysis of 4D cardiac magnetic resonance images. *Journal of The Danish Optical Society*, 4:38–39, 2001.

Mikkel B. Stegmann. Object tracking using active appearance models. In *Proceedings of the 10th Danish Conference on Pattern Recognition and Image Analysis*, volume 1, pages 54–60, 2001.

Christopher J. Taylor and Timothy F. Cootes. Combining elastic and statistical models of appearance variation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 149–163, 2000.

Christopher J. Taylor, Kevin N. Walker, and Timothy F. Cootes. Automatically building appearance models from image sequences using salient features. *Image and Vision Computing*, 20(5–6):435–440, 1999.

Christopher J. Taylor, Kevin N. Walker, and Timothy F. Cootes. View-based active appearance models. In *Proceedings of the International Conference on Face and Gesture Recognition*, pages 227–232, 2000.

Joshua B. Tenenbaum, Vin de Silva, , and John C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, 2000.

Texas Heart Institute. `http://www.tmc.edu/thi/anatomy2.html`, May 24 2005. Texas Heart Institute – St. Luke's Episcopal Hospital.

M. Üzümcü, A.F. Frangi, J. Reiber, and B. Lelieveldt. The use of independent component analysis in statistical shape models. In *Proceedings of SPIE Medical Imaging*, volume 5032, pages 375–383, 2003.

Thomas Vetter and Tomaso Poggio. Linear object classes and image synthesis from a single example image. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(7):733–742, 1997.

C. B. H. Wolstenholme and Christopher J. Taylor. Using wavelets for compression and multiresolution search with active appearance models. In *Proceedings of the British Machine Vision Conference (BMVC)*, pages 473–482, 1999.

Jing Xiao, Simon Baker, Iain Matthews, and Takeo Kanade. Real-time combined 2D+3D active appearance models. In *Proceedings of Computer Vision and Pattern Recognition (CVPR)*, pages 535–542, 2004.