A 4-week project in

Active Shape and Appearance Models

by Vedrana Andersen
vedrana@itu.dk
(130274-xxxx)

and Renée Anderson
renee@itu.dk
(081169-xxxx)

Project supervisor:
Marleen de Bruijne

IT University of Copenhagen
December 2005

Abstract

This paper describes a four-week project in active shape and appearance modeling, including experiments in shape and appearance modeling and a brief look at the fundamental theory. Shape modeling derives a statistical shape model from a set of example objects annotated with landmark points to generate new, similar object shapes. Shape modeling can be extended by gray-level modeling, where a similar technique is used to derive a statistical gray-level model by sampling the gray levels from example images. Shape and gray-level models can be combined into an appearance model, which describes the way shape and gray levels of an object vary from image to image.

We used MATLAB to design and implement an appearance model based on preannotated images. Our primary goal was to design a model that can be used to generate new, similar images. Our secondary goal was to use the model for segmentation by fitting the model to a new unknown image.

A key component of this project is our research into the field of active appearance modeling, in the interest of learning more about the field and apprising ourselves of the recent work of scientists in this area. T.F. Cootes and C.J. Taylor of the University of Manchester have provided definitive work in statistical shape and appearance modeling for the past few years [3], thereby enriching the fields of computer vision and medical image analysis. We have sought to explore their methodology by creating our own basic shape and appearance modeling system that can be adapted to many different types of objects.

Acknowledgments

Our thanks to Mikkel Bille Stegmann, former Master's student of the Technical University of Denmark, whose Master's thesis, *Active Appearance Models: Theory, Extensions and Cases* [9], provided the heart images and a fair amount of practical methodology for our work. The faces data base was also courtesy Mikkel Stegmann [6] as well as Stegmann et al. (2004, [11]). The cardiac data base was originally developed by Jens Chr. Nilsson and Bjørn A. Grønning, Danish Research Centre for Magnetic Resonance (DRCMR) [10].

Heartfelt thanks also to our project advisor Marleen de Bruijne, Assistant Professor, IT University of Copenhagen, for her advice, support, and encouragement, and for her unique way of lifting her eyebrows whenever we suggested omitting something difficult from the report.

Contents

Chapter 1

Background

There is no denying the inherent diversity of all natural life. Humans come in all shapes, sizes, and colors. Even identical twins are not absolutely identical. Yet, as much as we vary, our physiology and our health and well-being are dependent upon strict parameters that set us apart from all other creatures, parameters that physicians use to perform a basic appendectomy, for example, or to determine whether a CT scan reveals a deadly astrocytoma or a benign acoustic neuroma. Parameters that allow security cameras to differentiate a potted palm in the lobby of the First National Bank from the bank's president. We have come to rely more than ever on the aid of computers for analysis and diagnosis of medical maladies; for generating descriptions of missing persons; for biometric security systems; and much more.

Ironically, at the core of our diversity lies an instinctive inclination to spot examples of nonconformity. We are attracted to a beauty spot on the cheek, an asymmetric smile, or an interesting skin tone. Our reaction to severe deformity can range from mild fascination to utter revulsion. But regardless of how we react to something new or out of the ordinary, the fact that we react at all, given this wealth of diversity, indicates our sensitivity to certain limits in appearance. As we will show later in this paper, the manipulation of an average human face more than four or five standard deviations from the mean will produce an image that is caricaturish or grotesque, if not unrecognizable.

Model-based vision is a type of computer vision that makes use of some prior knowledge of the object to be modeled, how it looks, how it can change, and so on. A key issue when building models is not only the need for flexibility to cover the variations present among the objects in such a training set, but also the need to restrict this flexibility so as to be able to represent only "legal" or "plausible" examples of similar objects. Deciding how to deal with the balance between rigidness and flexibility often entails knowing something about the nature of the modeled objects, for example, finding a set of permissible transformations of the model shape.

Active shape modeling (ASM) and active appearance modeling (AAM) are two interrelated kinds of model-based vision that provide solutions to deal with the

rigidity/flexibility problem using statistical analysis of the training corpora—a set of images of similar objects. ASM and AAM make use of patterns of variability based on shape and gray-level intensity values, respectively. They capture how the object changes by statistically modeling the training set, not by considering the nature of the modeled objects.

Processing the training set produces a mean object, from which all objects in the training set may be seen to vary. Modeling allows us to define specific descriptive parameters for the objects, but because it also provides ranges for those parameters, it also allows us to evaluate whether similar objects *not* belonging to the original training set may be considered of the same class. It is therefore suitable for "objects" such as living creatures, which can vary widely within the strict confines of what precisely defines them as a species.

The normal distribution is a perfect stage for ASM and AAM, but these are not without limitation. Discretized behavior or nonlinear dependencies cannot be modeled with ASM and AAM without modifying them to deal with those nonlinearities.

Timothy F. Cootes and Christopher J. Taylor of the Division of Biomedical Engineering, University of Manchester, have led the field in these areas for the past few years [2] [3] [4] [5], providing methods for building shape and appearance models that we have also sought to emulate for this project.

Although we have limited our scope to building statistical shape and appearance models and running tests with AAM, it is worth noting that both ASM and AAM have found use in medical image segmentation and analysis, criminal investigations and in generating facilitated sketches of alleged criminals, simulated age progression for finding missing children, predicting the outcomes of plastic surgery, and much more. Lanitis et al. [5] have performed experiments in visualizing age progression. Vetter and Romdhani [12] have developed illustrations of facial manipulations, showing ranges between masculine and feminine, thin to round features, and more. Other applications are even finding their way into mainstream attention. A recent study [7] used appearance-modeling techniques to analyze the emotional content of the Mona Lisa's smile. It is certainly evident that new applications and improved techniques in these areas will continue to be developed as time goes on.

Chapter 2

Project Overview

In this project we have built statistical shape and appearance models (and attempted to build active appearance models) of the human face, heart, and spine. These models are flexible in the sense that they can also be used with other kinds of annotated image sets. We describe the methods we used to build these models and illustrate a number of applications.

The construction of shape and appearance models involves several steps, which were developed by Cootes and Taylor [3]. First, images to be analyzed must be consistently annotated with landmark points that can be used for correspondence among the images. Prior to building a shape model, images must be aligned along these landmark points. A shape model is built by applying principal component analysis to the coordinates of the landmark points.

The next step is building a gray-level model.[1] To do this, all objects must be warped to a mean shape using corresponding landmark points. The gray-level values of the resulting shape-free patches are sampled and normalized. The statistical model of the gray-level variation is then built by applying principal component analysis to the samples of gray-level values.

The last step is to build an appearance model by combining the shape and gray-level models into one model, and applying principal component analysis to the data one more time. In this way, the appearance model controls both the shape of the image and its gray-level variations, taking into account the correlation between shape and gray level.

We have mostly followed Cootes and Taylor's methods, incorporating some modification as became appropriate. One of the most significant changes we made was in the order of the steps: we took the steps of aligning, cropping, and normalizing the images out of the algorithm, and incorporated them into a preprocessing phase, so that these smaller tasks would not become part of the computations for the models. We felt this step was necessary but quite time-consuming to implement.

[1] We have favored the term *gray level* for our report, where others use this term interchangeably with *texture*. Additionally, where we use the term *appearance*, we mean it exclusively in the sense of a combined shape and gray-level model.

This was one area in which we perceived the time constraints of the four-week scope of the project period. (Another task that proved infeasible during the allotted time frame proved to be implementing an active appearance model.) We decided to focus our priority on being able to implement our models in as many ways as possible during the project period, instead of spending the time in refining and optimizing the steps involved in building the models themselves. In this report, we describe the steps we took, along with any modifications, in the same order that we implemented them.

Chapter 3, Data, describes in detail the three data sets we were given for our implementation, including the number of images in each set and how they were annotated.

Chapter 4, Preprocessing, explains the several steps we needed to take before we could use the data. The chapter describes the aligning of objects in the images, warping techniques we tested, and the process of gray-level normalization we used.

Chapter 5, Building the Models, includes descriptions of principal component analysis, the covariance matrix and modes of variation, a presentation of the statistical appearance models we constructed, as well as examples of approximation/reconstruction of faces.

Chapter 6, Manipulating Faces, shows some interesting applications of the appearance model, including generating caricatures and "anti-images," and generating smiles upon faces that originally bore a neutral expression.

Chapter 7, Active Appearance Models, describes correcting model parameters during a search and iterative model refinement.

In the final chapter, we present an assessment of what we have done, and offer some concluding remarks, and in the Appendix, we present the body of our MATLAB code that we developed and used to perform the processes in this report.

Chapter 3

Data

3.1 Our data sets

We looked at images of the human heart, lower spine, and face, which had been preannotated with two-dimensional landmark points.

The faces data comprised 240 JPEG images of 40 different people, 7 female and 33 male. Individuals were photographed in various aspects—facing slightly left, facing slightly right, facing forward under diffuse lighting conditions, and facing forward while illuminated from one side—and with various facial expressions, including neutral, smiling, and "joker" (some arbitrary expression). Some of the males had beards or mustaches, and none of the subjects wore glasses. These images were annotated manually using 58 landmark points lying on seven paths: the jawline, mouth, two eyes, two eyebrows, and nose.

The cardiac data consisted of 14 MR slices, annotated manually with 30 landmark points outlining two concentric rings that make up the inner and outer wall of the left ventricle. For both the cardiac and face data, each image was accompanied by a file containing the full set of landmark coordinates for that image. Landmark points were numbered consecutively in a single path. That file also contained information on the type of annotation points, such as whether points were internal or located on an outside border, and whether they lay on closed or open paths.

The spine data, which was given as a single MATLAB structure, included 20 X-ray images, annotated with 100 landmark points. The landmark points were given in two matrices, one for the x-coordinates, and one for the y-coordinates. No information about the paths was provided.

Our first task was bringing the data into a consistent format. For the faces and cardiac data sets, it was a matter of automating the loading procedure that would allow us to work with the data in MATLAB, which included manually stripping the files of any lines containing nondata (headings and other commented-out information). Finally, the faces and cardiac landmark points were given in coordinates relative to the size of the images, so it was necessary to recalculate those coordinates, so that all coordinate systems corresponded.

3.2 Annotation

Annotation is the marking of images with landmark points to describe some object of interest contained in the image. These points may be set at key intersections and along object boundaries. Although the images we utilized were preannotated, it is worth noting that the placement of landmark points in medical images is not trivial. There are many techniques for annotating images, and among the many goals of model-based vision are accurate and robust methods for automating this vital step. Landmark points are commonly used for two- or three-dimensional images. Their purpose is to describe the invariant shape of an object, one that should not change even after rotation, translation, or scaling. Each object in a set of images must be demarcated with the same number of landmark points. In a data set of similar shapes, although object shapes themselves may vary from image to image, a landmark point in one image should correspond precisely to a landmark point in another image that fulfills the same structural description, such that all landmark points in all images correspond to one another consistently.

3.3 Paths

Our landmark points lay on artificial "paths" that that allowed us to better visualize contours when studying the shapes. Some of these paths were open, such as the paths outlining each vertebræ. Other paths were closed, as with the eyes and mouth of the faces.

Designing one algorithm that could handle the different kinds of paths using the information provided would be a challenge. We could have ignored the paths, and simply plotted the unconnected landmark points, but in some cases, such as when we wanted to view extreme modes of variation, the result would be not more than a collection of random-looking dots, too jumbled to interpret meaningfully. We therefore preferred keeping the paths for visualization, so we needed to find an appropriate means to work with them.

We decided to implement a `shape_plot` function that allowed us to manually divide paths for the face, heart, and spine images. For example, in the cardiac images, where the landmark points formed two roughly concentric rings, we divided the single path into two, so that the result would not include an extraneous, undesired line connecting the last point of the outer circle with the first point of the inner circle. The face data sets included seven paths, some of open, some closed. For the spine data, we used four open paths, one for each vertebra.

Chapter 4

Preprocessing

The preprocessing of the raw, original images was a vital first step toward being able to work meaningfully with them. Before the objects in the image files could be properly analyzed, compared, or manipulated, we needed to level the playing field: cardiac cross sections and spines needed to occupy the same area in the coordinate plane, and faces needed to be brought into some semblance of consistency. The preprocessing steps we used included the following and are illustrated in Figure 4.1:

1. Iterative shape alignment, which included the translation, rotation, scaling, cropping, and masking of the image;
2. Warping of the image landmark points to the mean;
3. Normalization of gray values to reduce variation in lighting conditions and other factors, such as the presence of facial hair, freckles, etc.

Preprocessing all 228 face images took 625 seconds on a standard PC. The resulting images and their corresponding shapes thus served as the input data for our statistical appearance model.

4.1 Iterative shape alignment

We built an initial shape model from the raw, unprocessed data of the face and cardiac images. In both models, the two most-significant modes of variation were translations in the x and y directions. The third mode of variation was scale. To remove variations attributable to translation, rotation, and scale, it is generally desirable to align the shapes into a common coordinate frame.

Translation and rotation in face and cardiac cases were unlikely to be important factors for shape modeling, given that they were artifacts of the image-capturing process. However, the correlation between scale and shape could be useful information, especially when dealing with medical images—differences in scale could

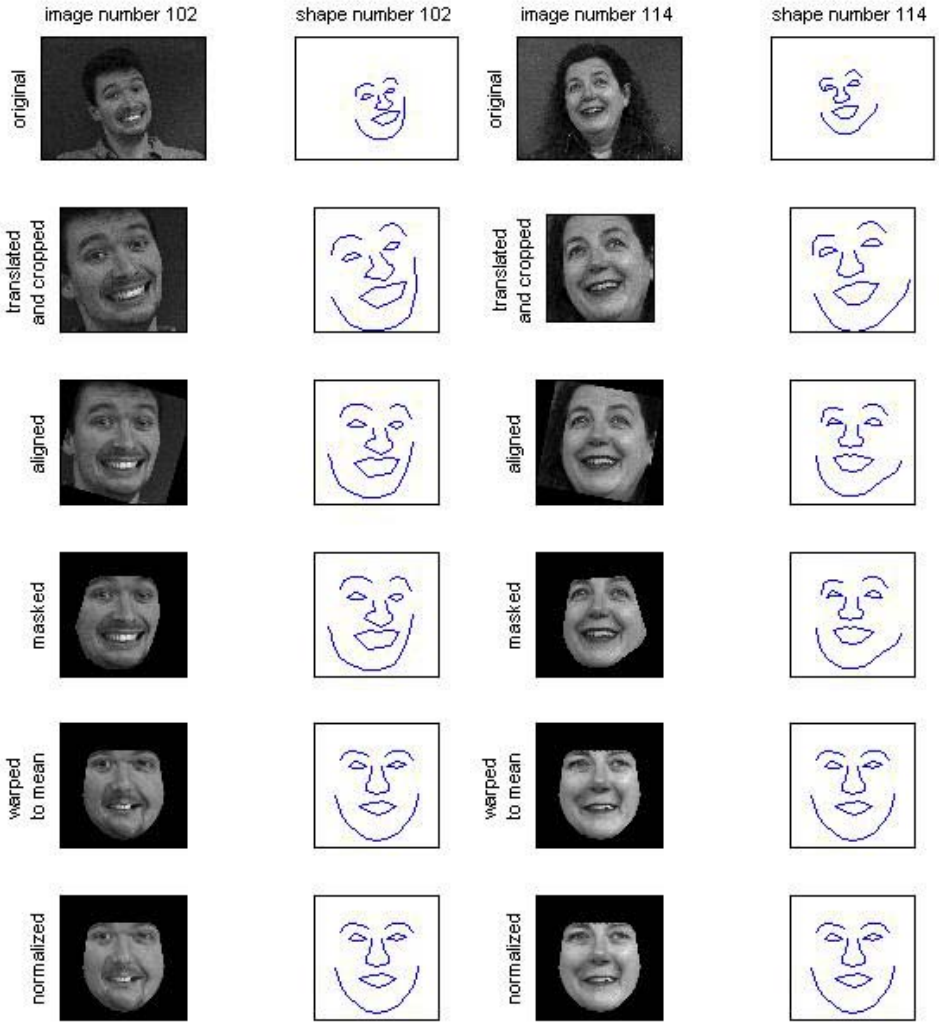

Figure 4.1: Preprocessing. Two sample face images and their corresponding shapes, depicting preprocessing steps: translation and cropping, shape alignment, masking, warping of landmark points to the mean, and normalization of gray values.

be important symptoms of a real physiological anomaly, but we could not evaluate this ourselves. Similarly, if children's faces had been included with the faces data set, correlation between scale and shape would be very useful indeed, as children's faces are not merely smaller than adults', their features are also in different proportion compared with adults.

When subsequently informed that when analyzing the shape of the heart, the diameter is not as important as the relative thickness of the walls, and in consideration of the fact that children's faces were not included with the face images, we decided to permit scale as a mode of variation for the cardiac images. Doing so also made our preprocessing procedure simpler, as the same steps could then be applied to all of our data.

"Procrustes analysis" is a commonly used iterative method [4] [9] for aligning shapes so that each shape occupies a common coordinate system. This alignment procedure is applied to the vectors $\mathbf{x}$ obtained by concatenating x and y coordinates of landmark points.

1. Translate all images so that the center of gravity of landmark points is at the origin.
2. Find an initial initial estimation of the mean shape $\bar{\mathbf{x}}_\mathbf{0}$ and scale it so that $|\bar{\mathbf{x}}| = 1$.
3. Record the first estimate as $\bar{\mathbf{x}}_0$ to define the default reference frame.
4. Align all the shapes with the current estimate of the mean shape.
5. Re-estimate mean from aligned shapes.
6. Apply constraints on the current estimate of the mean by aligning it with $\bar{x_0}$ and scaling so that $|\bar{\mathbf{x}}| = 1$.
7. If not converged, return to **4**.

Convergence is declared if the estimate of the mean does not change significantly after an iteration.

Aligning the two images in the step 4 could be done in different ways, depending on which transformations are allowed. In the case of 2-D similarity, permitted transformations include translation, scaling, and rotation. In other words, aligning the images properly entails finding a transformation

$$T(\mathbf{x}) = \begin{pmatrix} a & -b \\ b & a \end{pmatrix} \mathbf{x} + \begin{pmatrix} t_x \\ t_y \end{pmatrix}$$

so that the sum of the distances of each shape to the mean is minimized.

Implementation

We aligned our shapes using similarity transformations, and in accordance with the iterative steps outlined by Cootes and Taylor, which we modified slightly. We initially translated images in such a way that the center of gravity was in the center of the image, and after the final alignment, we scaled all the shapes back using the same scaling factor for all shapes, so that landmark point coordinates would correspond to pixel coordinates.

Instead of deciding upon criteria for achieving convergence beforehand, we elected to always run the alignment over ten iterations. We then looked at the changes of mean estimate. As can be seen in Figure 4.2, it is clear that convergence is relatively easily achieved after only two iterations.

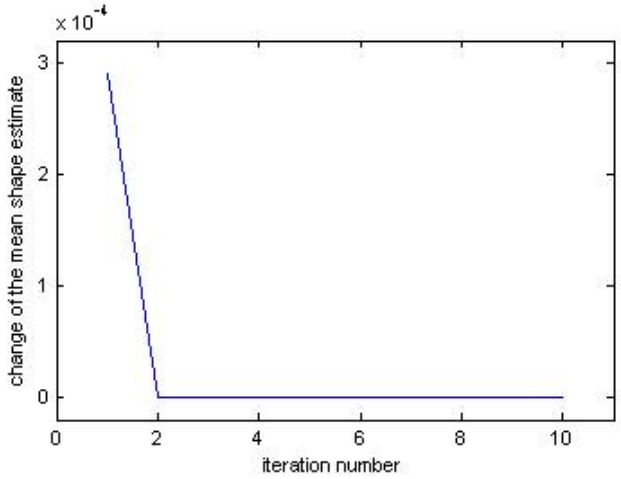

Figure 4.2: Iterative shape alignment. Graph depicts convergence occurring in two iterations of Procrustes alignment algorithm.

4.2 Warping the images

To build a gray-level model of the object, we needed to sample gray-levels across the image. However, it would have been pointless to do the gray-level sampling at this stage: although the object shapes were aligned, they were not identical and there was no guarantee that pixels with same coordinates represented the same part of the object from image to image.

Therefore, an intermediate step was to warp (geometrically transform) all images in such a way that the objects in the resulting images all had the same shape. The logical choice for the target shapes was the mean shape itself, so our goal was to transform each training image $\mathbf{I}$ into a new image $\mathbf{I}'$ in such a way that the set of n landmark points $\mathbf{x}_i$ were mapped to new positions $\mathbf{x}'_i$. To do this, we needed to

find a vector-valued mapping function **f** that satisfied

$$\mathbf{f}(\mathbf{x}_i) = \mathbf{x}'_i, \quad \forall i = 1 \ldots n$$

We studied and compared two warping functions, piecewise-linear and thin-plate spline warps.

Piecewise-linear warping assumes the mapping function **f** to be bilinear in a local region and zero everywhere else. The convex hull of the landmark points is first partitioned using a triangulation algorithm, and a different linear transformation is applied to each triangle. Piecewise-linear warping is applicable only to the convex hull of landmark points, and it provides a continuous but not quite smooth deformation.

Thin-plate spline warping, popularized by Bookstein [1], leads to smooth deformation and is not constrained to the convex hull of landmark points, but is more difficult to calculate.[1]

Figure 4.3 illustrates the comparison between piecewise-linear and thin-plate spline warping. The difference between the two warping methods is evident outside the convex hull of landmark points, but it is rather small within the convex hull.

Implementation

We encountered a few images that could not be warped using the piecewise-linear method, for example, that method could not handle the inversion of a triangle—a triangle that has one orientation in the original image, but another orientation in the target image. The thin-plate splines method could deal with the inverted triangles, but the result was often so distorted and implausible that it should be eliminated. See Figure 4.4 for an example of such an implausible result.

Although all of our algorithms support piecewise-linear and thin-plate spline warping, we elected to use the piecewise-linear warping combined with bilinear interpolation for the faces data set, given its much smaller computing time compared with the thin-plate spline implementation available to us. We decided to use thin-plate spline warps for the heart and spine data sets, however, because each set contained only a few images, which did not pose too great a problem computationally.

In order to remove unwanted effects from the area outside the convex hull of the landmark points, we applied a "mask" to all images, leaving only the area inside the convex hull visible. The result of the warping was a set of shape-free image patches, each image with different gray-level structures; our gray-level model is based on those structures.

[1] Whereas MATLAB is prepackaged with a piecewise-linear warp function, we relied on code written by John F. Meinel, Jr. (copyright 2002, jfmeinel@engineering.uiowa.edu) to perform the thin-plate spline warps.

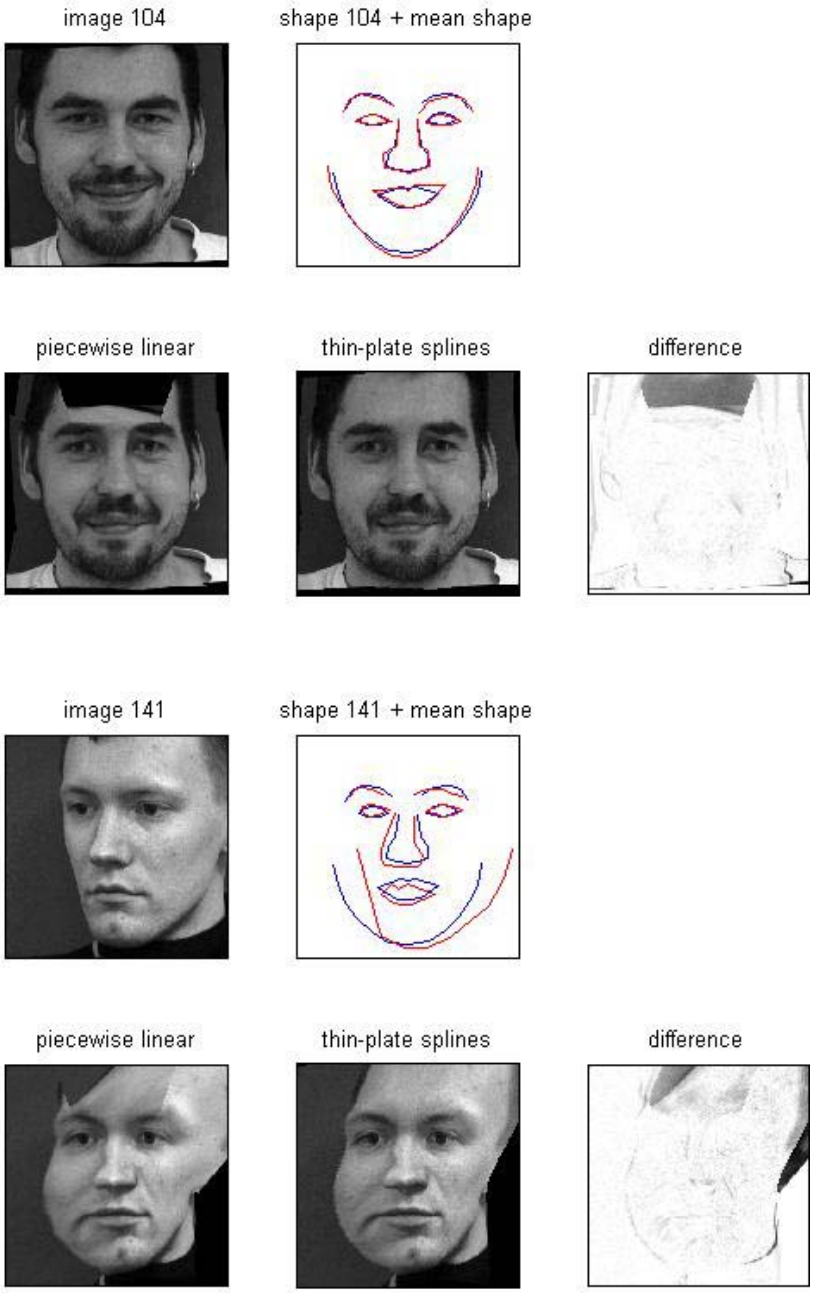

Figure 4.3: Comparison of piecewise-linear and thin-plate spline warping techniques. The top set of images depict a relatively straightforward warp, given the similarity of the mean and the image shapes. The bottom set of images depict a more complicated warp. In both cases, the comparison between the two warping techniques is very subtle inside the convex hull boundaries. The "difference" subfigures show the difference between the two warped images.

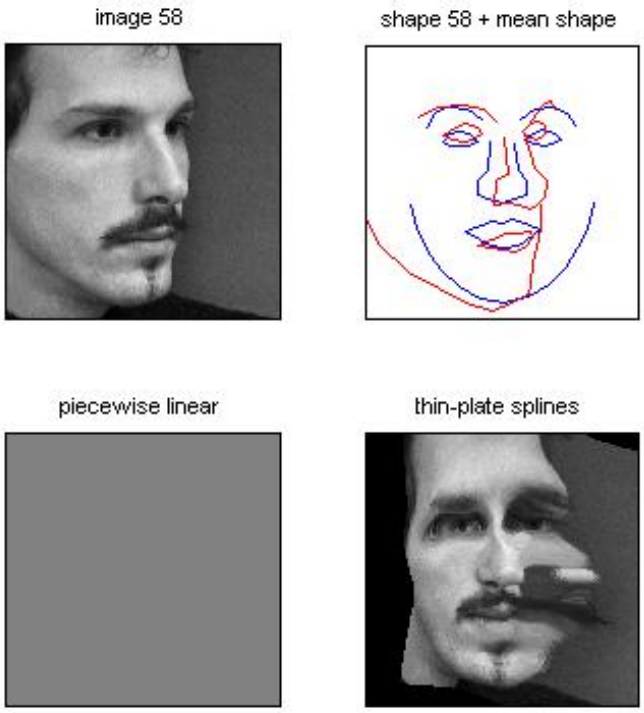

Figure 4.4: Example of an unacceptable warp. Piecewise-linear method simply gives up and produces no result; thin-plate spline method produces implausible result.

4.3 Iterative Gray-Level Normalization

Having warped the images to the mean shape, we were just one preprocessing step away from being able to build a gray-level model. Some of our shape-free image patches tended to be rather light or dark, so we needed to normalize the images to minimize the effects of global lighting.

First, we needed to decide on the area of interest (the object we plan to model), which is usually the area inside a convex hull of the outer landmark points, but it can also be made to incorporate some of the background. From this step, we then sample the gray values of each pixel inside the area of interest.

Gray-level normalization is an iterative process, analogous to the shape alignment. We applied normalization to the length n vectors $\mathbf{g}$ obtained by sampling the gray-level values of every pixel within the convex hull of landmark points, according to the following steps:

1. Find an initial estimation of the mean vector $\bar{\mathbf{g}}$.
2. Normalize all the samples $\mathbf{g}$ by applying scaling α and offset β that best match the vector of normalized mean

$$\alpha = \mathbf{g} \cdot \bar{\mathbf{g}}, \qquad \beta = (\mathbf{g} \cdot \mathbf{1})/n$$

$$\mathbf{g}_{\text{new}} = (\mathbf{g} - \beta \mathbf{1})/\alpha$$

3. Re-evaluate the normalized mean $\bar{\mathbf{g}}$ by finding a mean of the new sample values and then applying scaling and offset to the mean vector, so that the resulting vector has zero mean and unity variance.
4. If not converged, return to **2**.

Convergence can be declared when the estimate of the mean does not change significantly after iteration.

Implementation

We elected simply to iterate a certain number of times, and plot the change of mean estimation. An example of this plot is given in Figure 4.5. It is evident that the convergence was achieved after just three iterations, and very nearly so after only two.

To be able to visualize our images, after the normalization we returned the sample values to the range of $[0, 255]$, using the same scaling and offset for all samples.

We considered including part of the background to serve as a border for the cardiac images, but ultimately we decided not to do that. First, it was not necessary in our estimation, and it made our process easier, as all data sets could be handled in the same way.

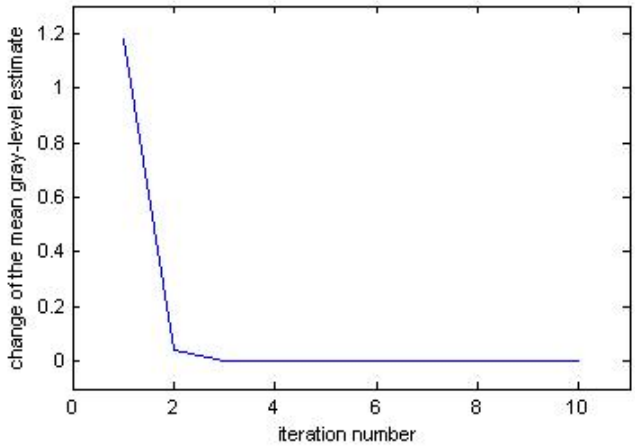

Figure 4.5: Iterative gray-level alignment, depicting convergence after just three iterations.

Normalization resulted in low-contrast images with a washed-out, gray look to them. This was because the range $[0, 255]$ must cover the gray-level variety of all the normalized images, but the result was that most images utilized only a small portion of this range.

Chapter 5

Building the Models

After all the preprocessing was finished, we were left with a certain number of images, each one contributing two components to the models:

1. A set of landmark points aligned to the mean shape.
2. A shape-free patch with gray-level values normalized to match the mean gray level.

The term *mean shape* indicates the mean of all landmark points in an entire set of annotated images. *Mean gray level* signifies the mean of all vectors obtained by sampling the gray levels inside the shape-free patch.

We used the sets of landmark points to build a statistical shape model, and the shape-free patches to build a statistical gray-level model. The procedures for building the shape and gray-level models are very similar, the key method being principal component analysis (PCA). To build an appearance model we combined shape and gray-level modeling and applied principal component analysis once more, so as to take into account the correlation between shape and gray level.

We begin this chapter with a short explanation of PCA, as it is obviously an important method used in building the models of statistical distribution. We then explain the process of building shape, gray-level, and appearance models in more detail, and finally, we present the models themselves, as given below.

We ended up building a total of six complete appearance models:

1. The complete faces data set: 228 images—all facial expressions, all positions, including frontal and oblique ($\pm 30^{\circ}$)
2. Frontal-only faces, all facial expressions: 151 images
3. Frontal-only faces, neutral facial expressions only: 40 images
4. Frontal-only faces, smiling facial expressions only: 40 images
5. The cardiac data set: 14 images
6. The spinal data set: 20 images

5.1 Principal component analysis

Principal component analysis (PCA) [8] is a powerful statistical technique that is used for reducing the dimensionality of multi-dimensional data. We'll start the explanation of the basic idea of PCA with a small illustration.

A tailor taking measurements for making a suit will be able to represent each of his customers with a set of numbers. Assuming that there are n measurements taken, each customer can be represented as a point (vector) in n-dimensional space

$$\mathbf{x} = (x_1, x_2, \ldots, x_n)^T$$

A set of s customers is then represented as a distribution of s points in n-dimensional space

$$\mathbf{x}_i = (x_{i_1}, x_{i_2}, \ldots, x_{i_n})^T, \quad i = 1, \ldots, s$$

There are some observations we can make about this distribution. First, each individual measurement is confined to a certain range and has a more-or-less normal distribution within that range. Second, many of the measurements are correlated: as one measurement increases, the others will generally also tend to increase. The first observation makes us assume that the measurements form a kind of cluster, or cloud, in n-dimensional space, the density of the cloud being highest in the middle. The other observation makes us infer a certain shape (hyper-ellipsoid) or direction for this cloud. It is impossible to visualize this in many dimensions, but in 3-D it might be a flattened zeppelin shape pointing in a certain direction.

We can easily imagine that an experienced tailor should be able to make a suit even if some of the measurements are missing. It is also reasonable that some of the measurements are more important than others. PCA takes this a step further—it is used to find a combination of measurements that will provide the best information about the data.

The basic purpose of performing PCA is to find an orthogonal coordinate system for our data cloud, in such a way that the greatest variance lies on the longest axis (first principal component), the second greatest variance lies on the second-longest axis, and so on. For a data cloud with the flattened zeppelin shape, the first principal component would be the line through the middle along the length of it. We can then use the projection of the data cloud on the first axis as the initial approximation of the data cloud, and we can improve the approximation by adding more axes. The spread of the data along a certain axis provides information on how important that axis is for the approximation.

It is important to note that PCA uses a linear model to describe the data, so it can only handle linear behavior. Any nonlinear dependencies between the measurements (for example a quadratic dependency or discretized data) would present a problem.

Our (math-loving) tailor could then build a statistical model from the data he collected, and he could decide to use a smaller number of values to represent each

of his clients. Those values would generally be linear combinations of the original measurements. This model could also be used to generate plausible, statistically probable measurements, from which our tailor could make suits that will probably fit some of his customers.

A way of calculating principal components is by finding eigenvectors and eigenvalues of the covariance matrix.

5.1.1 The covariance matrix

Looking just at one measurement $\mathbf{X}$, for example neck-to-shoulder, across the whole data set, we can measure its spread by calculating the variance

$$\mathrm{var}(\mathbf{X}) = \frac{\sum_{k=1}^{s}(X_k - \bar{X})^2}{s-1}$$

Standard deviation is the second root of variance.

To see how two measurements $\mathbf{X}$ and $\mathbf{Y}$ vary with respect to one another, we can use covariance

$$\mathrm{cov}(\mathbf{X},\mathbf{Y}) = \frac{\sum_{k=1}^{s}(X_k - \bar{X})(Y_k - \bar{Y})}{s-1}$$

Obviously, $\mathrm{cov}(\mathbf{X},\mathbf{Y}) = \mathrm{cov}(\mathbf{Y},\mathbf{X})$ because of the commutativity of multiplication. Positive covariance indicates that the two measurements $\mathbf{X}$ and $\mathbf{Y}$ increase together. If covariance is negative, it means that as one measurement increases, the other decreases. Zero covariance indicates independent measurements.

Having n measurements, we can calculate the covariance of any two measurements. The covariance matrix is the matrix containing these covariances of all pairs of measurements—the element in the ith column and jth row is the covariance between ith and jth measurement. The covariance matrix is of size $n \times n$, and it is symmetrical because of the commutativity of covariance. The diagonal gives the variance for each individual measurement.

Expressed in terms of s points $\mathbf{x}_i$ in n-dimensional space, the covariance matrix is

$$\mathbf{S} = \frac{1}{s-1}\sum_{i=1}^{s}(\mathbf{x}_i - \bar{\mathbf{x}})(\mathbf{x}_i - \bar{\mathbf{x}})^T$$

where $\bar{\mathbf{x}} = \frac{1}{s}\sum_{i=1}^{s}\mathbf{x}_i$ is the mean of the data, or rather, a vector containing the means of all measurements.

The covariance matrix for the tailor's data set would probably contain only positive values, since we expect that as one measurement increases, the others generally do also.

To achieve the task of PCA, one basically needs to find a coordinate system in which the covariance matrix for the data is diagonal (coordinates are not mutually dependent), and elements on the diagonal are in sorted order. This is equivalent to finding the eigenvectors of the covariance matrix and sorting them by their corresponding eigenvalues.

5.1.2 Eigenvectors and eigenvalues

After finding and sorting the eigenvalues λ_i of the covariance matrix $\mathbf{S}$, the eigenvector ϕ_1 corresponding to the largest eigenvalue is the direction of the first principal component, and λ_1 is the covariance along the first principal component. The second eigenvector corresponds to the second principal component, and so on.

If we put all eigenvectors ϕ_i into an $n \times p$ matrix Φ, we can express any of the points $\mathbf{x}$ using

$$\mathbf{x} = \bar{\mathbf{x}} + \Phi\mathbf{b}$$

where $\mathbf{b}$ is a p-dimensional vector of new coordinates given by

$$\mathbf{b} = \Phi^T(\mathbf{x} - \bar{\mathbf{x}})$$

If we put only the first $t < p$ eigenvectors into an $n \times t$ matrix Φ, we can approximate the points $\mathbf{x}$ by

$$\mathbf{x} \approx \bar{\mathbf{x}} + \Phi\mathbf{b}$$

where $\mathbf{b}$ is a t-dimensional vector given by

$$\mathbf{b} = \Phi^T(\mathbf{x} - \bar{\mathbf{x}})$$

The number of eigenvectors to retain, t, can be decided by looking at the proportions of the total variance that are represented by the first t principal components. For example, we might look at the proportion between $\sum_{i=1}^{t} \lambda_i$ and $\sum_{i=1}^{p} \lambda_i$.

5.1.3 Having fewer points than dimensions

The covariance matrix will always be of size $n \times n$, no matter how many (or how few) points there are in the n-dimensional space. But, if there are just two points, there is only mode of variation, and it is on the line connecting the points. Adding a third point will increase the number of modes of variation only if that third point is not on the line between the first two points. With s points, there will always be at most $s-1$ modes of variation. This means that the covariance matrix $\mathbf{S}$ will never have more than $s-1$ nontrivial eigenvalues, because it will never have more than $s-1$ linearly independent rows (or columns).

If we want to apply PCA to a data set in which there are fewer points than dimensions, we can calculate the eigenvectors and eigenvalues of the $n \times n$ covariance matrix $\mathbf{S}$ using a smaller $s \times s$ matrix, which can provide considerable savings computationally.

The covariance matrix $\mathbf{S}$ can be written as

$$\mathbf{S} = \frac{1}{s}\mathbf{D}\mathbf{D}^T$$

where $\mathbf{D}$ is obtained by subtracting the mean from each data vector and storing these means in a new matrix.

We define an $s \times s$ matrix $\mathbf{T}$ as

$$\mathbf{T} = \frac{1}{s}\mathbf{D}^T\mathbf{D}$$

and calculate its eigenvectors $\mathbf{e}_i$. It can be shown that the vectors $\mathbf{De}_i$ are all eigenvectors of $\mathbf{S}$, but not necessarily of unit length. The remaining eigenvectors of $\mathbf{S}$ are trivial, i.e., they have zero eigenvalues.

In conclusion, if there are s points in n-dimensional space, the number of modes of variation will not exceed $\max(n, s-1)$.

5.2 Building the shape model

To build the shape model of an object, we need a set of aligned example shapes described by landmark points. With 2-D images annotated with n landmark points, we can represent each shape with a $2n$ element vector $\mathbf{s}$, obtained by concatenating the x and y coordinates of landmark points

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

A set of s images is a then distribution of s points $\mathbf{s}_i$ in $2n$-dimensional space, and we can apply PCA to it to obtain the model

$$\mathbf{s} = \bar{\mathbf{s}} + \mathbf{P}_s\mathbf{b}_s$$

where $\bar{\mathbf{s}}$ is the mean shape vector, $\mathbf{P}_s$ is a set of orthogonal modes of variation (the eigenvectors of the covariance matrix), and $\mathbf{b}_s$ is a set of shape parameters.

The variance for each mode of variation is given by λ_s (the eigenvalues of the covariance matrix). The total number of modes can be chosen so that the model represents some proportion of the total variance of the data.

By varying the parameters $\mathbf{b}_s$, we can vary the shape. To generate plausible shapes, similar to those in the training set, each of the parameters b_{s_i} needs to be limited to a certain range, usually ± 3 standard deviations of the corresponding mode, i.e., $\pm 3\sqrt{\lambda_{s_i}}$.

We can visualize each mode of variation by varying a corresponding parameter within its specified range, and leave other parameters at zero. Figure 5.1 illustrates our shape model, based on 151 images from the faces data set, in which we see the first four modes of variation, visualized using the landmark paths alone. It is not difficult to guess at what some of the modes of variation might be, for example, slight left/right oblique angle, small/large mouth, upturned/downturned angle. It is important to note that these guesses will often not be fully accurate, as modes of variation usually represent some unknown combination of characteristics.

In Figure 5.3, we see some plausible and implausible shapes, all made using the frontal-faces data base. The first two shapes have parameters that are all within the range of ± 3 s.d. from the mean. The second two shapes, however, have parameters that vary widely from this range, and the resulting face shapes, although

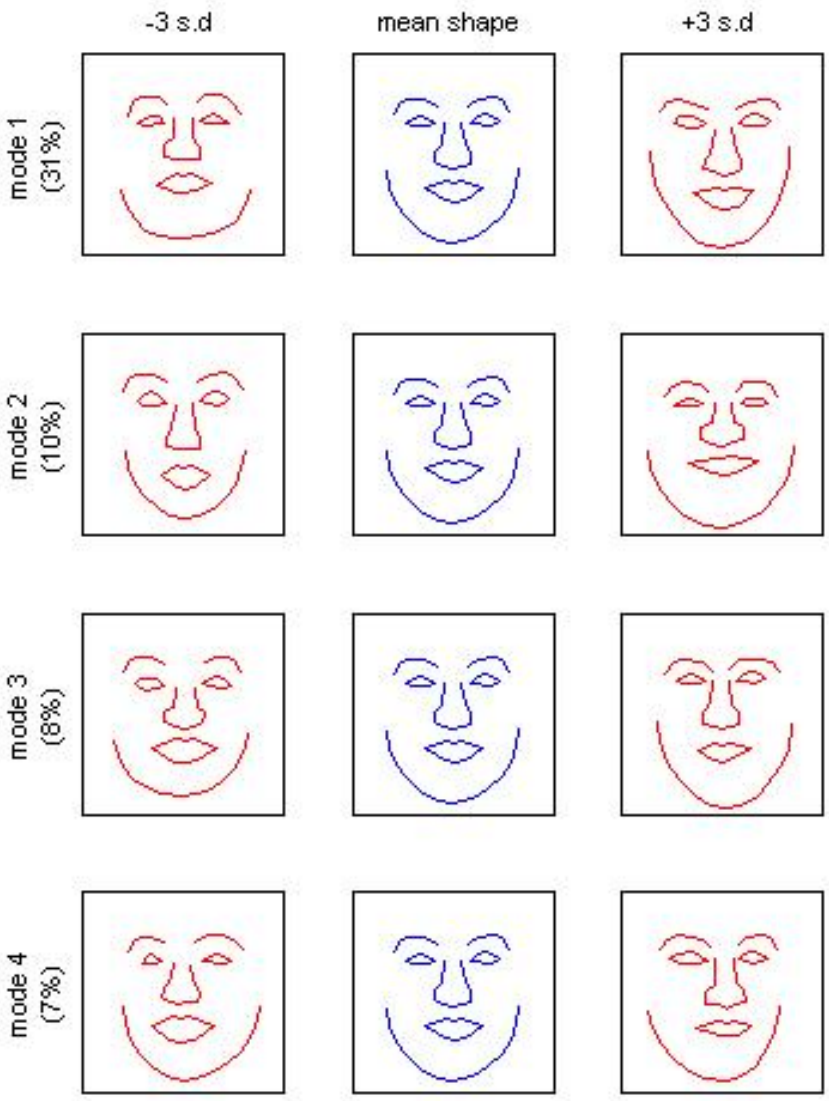

Figure 5.1: Visualization of faces shape model, showing the four most significant modes of variation from top to bottom. The middle column depicts the mean shape, with left and right columns showing $\pm$ 3 s.d. for each of the three modes. Percentages represent the portion of total variations in the set of images used to build the model, and are cumulative in nature. The first two modes of variation together represent 41% of all variation; the first three modes together represent 49%; and so on.

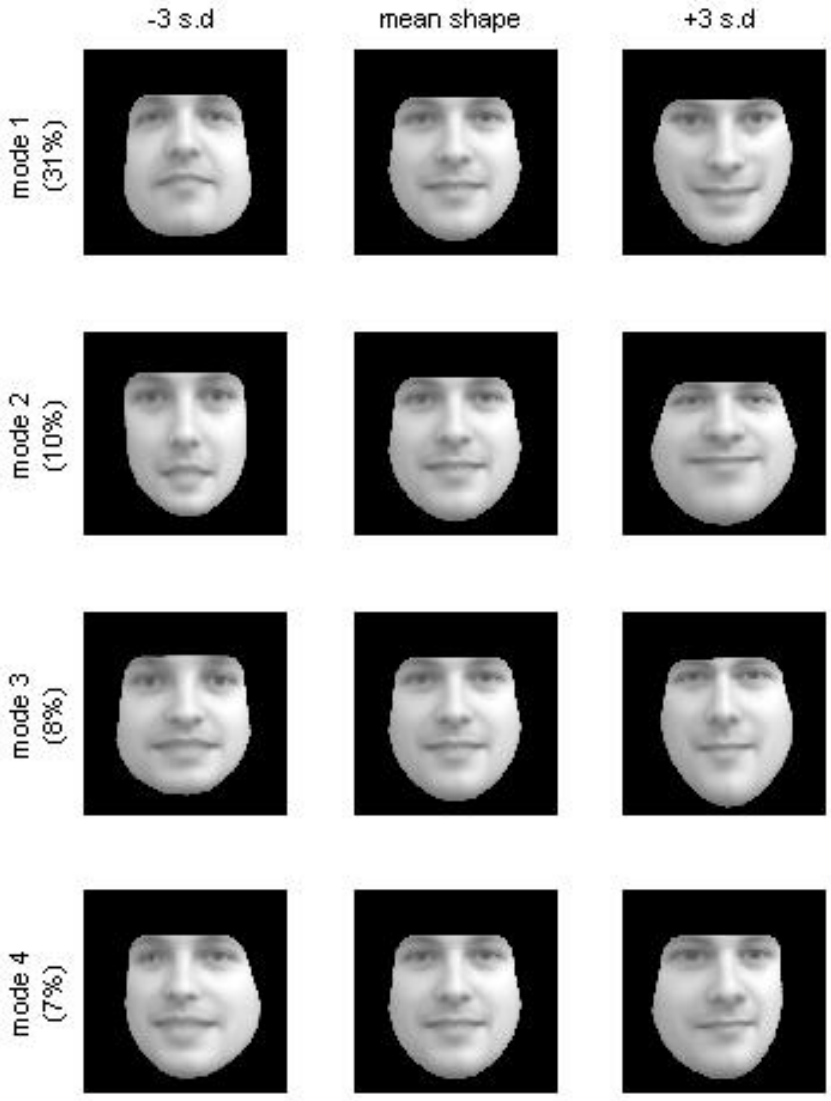

Figure 5.2: Another visualization, face shapes as shown in Fig. 5.1, made using mean gray levels.

still vaguely human, are clearly unusable for representing average human faces. To generate these implausible shapes, it was necessary to use the more robust thin-plate spline method, as piecewise-linear warping produced no result.

Another way of visualizing the shape model is seen in Figure 5.2, which depicts the same face shapes as in Figure 5.1, but with images that use the mean gray-level variation. We include this figure here for the purposes of visualization only, as it is not otherwise possible to produce this visualization until after building the gray-level model.

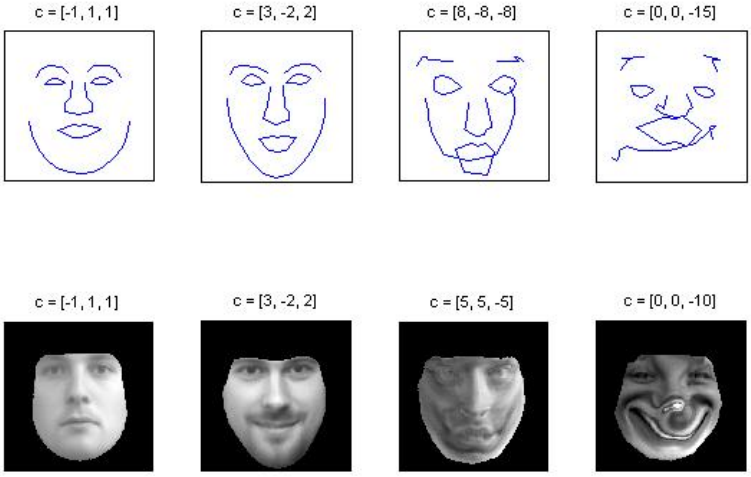

Figure 5.3: Four example shapes. Top row, shape only. Bottom row, shape visualized using mean gray-scale values. The first two shapes are within the range ± 3 s.d. from the mean. The second two shapes are examples of implausible results that occur when given parameters are too far beyond the specified range.

5.3 Building the gray-level model

To build the gray-level model of an object, we need a training set of gray-level-normalized shape-free patches obtained by warping each image to the mean shape. We sample the gray value of each pixel in the patch, and this must be done over the same area that we sampled when doing the gray-level normalization. If, at that phase, we incorporated some of the background into the area of interest, we would need to include it at this phase as well. If there are m pixels inside the patch, we can represent the gray-level variation of each patch by a m-element vector $\mathbf{g}$.

A set of s images is therefore a distribution of s points $\mathbf{g}_i$ in m-dimensional

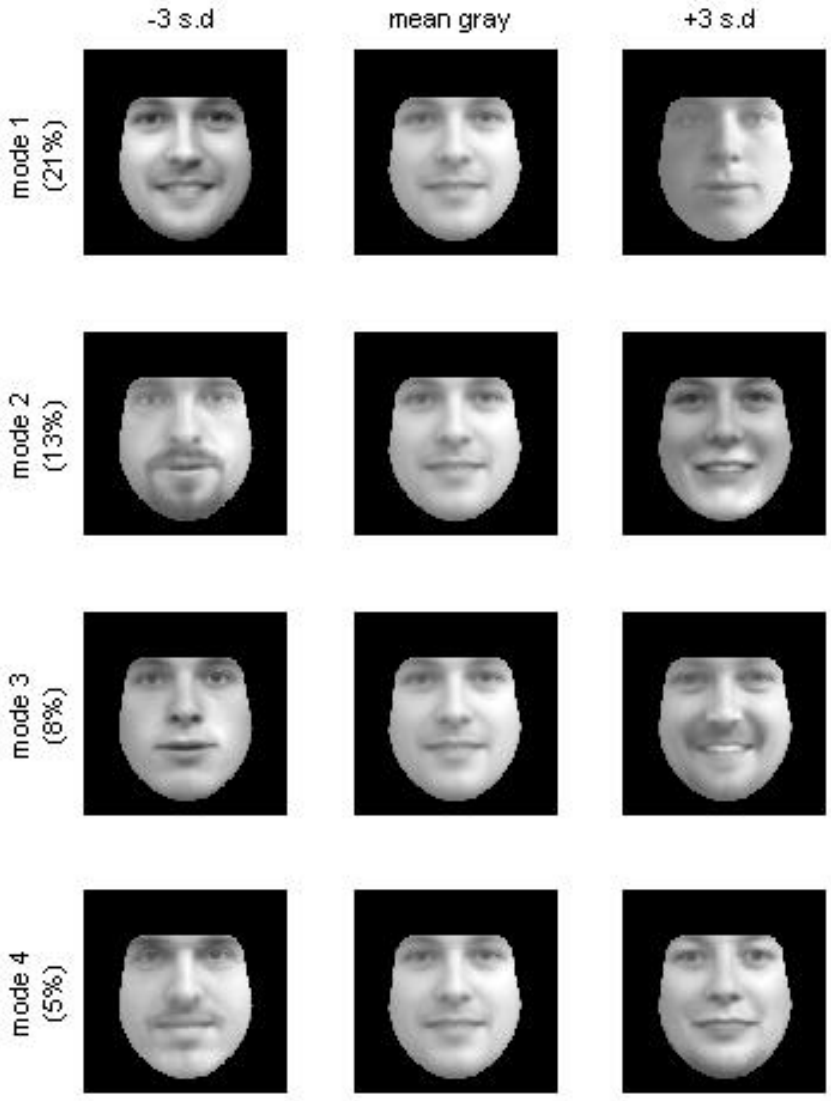

Figure 5.4: The faces model based on gray-level intensity values. The first four modes of variation are shown.

space, and we can apply PCA to it to obtain the model

$$\mathbf{g} = \bar{\mathbf{g}} + \mathbf{P}_g \mathbf{b}_g$$

where $\bar{\mathbf{g}}$ is the mean gray-level vector, $\mathbf{P}_g$ is a set of orthogonal modes of variation, and $\mathbf{b}_g$ is a set of shape parameters.

The gray-level variations of the image can be generated from the parameters $\mathbf{P}_g$, and the normalization parameters α and β.

The gray-level shape model exhibits the same mean shape throughout the model. The modes of variation are based solely on the gray-level intensity values of the images in the data set. These might be expressed in terms of overall contrast (low vs. high), white teeth vs. dark opening between the lips, dark facial hair vs. light, and so on. Figure 5.4 illustrates the first four modes of variation obtained with this model.

When implementing the gray-level model, we did not consider the normalization parameters, so the images generated by our model often have relatively low overall contrast. For example, note the upper-left image of Figure 5.4, which shows a face image source-lit from the person's left. When viewed together with all the images in the data set, images with source lighting stand out in marked contrast to the rest of the images, as these faces are starkly shaded on one side. With overall shape not in consideration for this model, it is not surprising that source lighting would be represented in this first mode of variation. Source lighting was only ever used on the left side of the face, however. The opposite of this image (upper left), was generated according to model calculations, even though there was no face in the original image set that resembled the face depicted here.

A similar example of such a phenomenon can be seen with face images that include dark facial hair and their opposites. The opposite images feature the semblance of matching facial hair that is all white, or white skin tones in the same areas, although there were no images in the original data set that featured people with white facial hair.

5.4 Building the appearance models

One rather straightforward way of combining shape and gray-level variation into a single model would be to concatenate the shape vectors **s** and gray-level vectors **g** into a single observation vector, and then apply PCA to such data. However, Cootes and Taylor ([3]) propose another method, which we have followed, and which is described below.

First shape and gray-level models need to be built, and each object needs to be represented by the shape parameters $\mathbf{b}_s$ and the gray-level parameters $\mathbf{b}_g$. For each object, we then generate the concatenated vector

$$\mathbf{a} = \begin{pmatrix} \mathbf{W}_s \mathbf{b}_s \\ \mathbf{b}_g \end{pmatrix} = \begin{pmatrix} \mathbf{W}_s \mathbf{P}_s^T (\mathbf{s} - \bar{\mathbf{s}}) \\ \mathbf{P}_g^T (\mathbf{g} - \bar{\mathbf{g}}) \end{pmatrix}$$

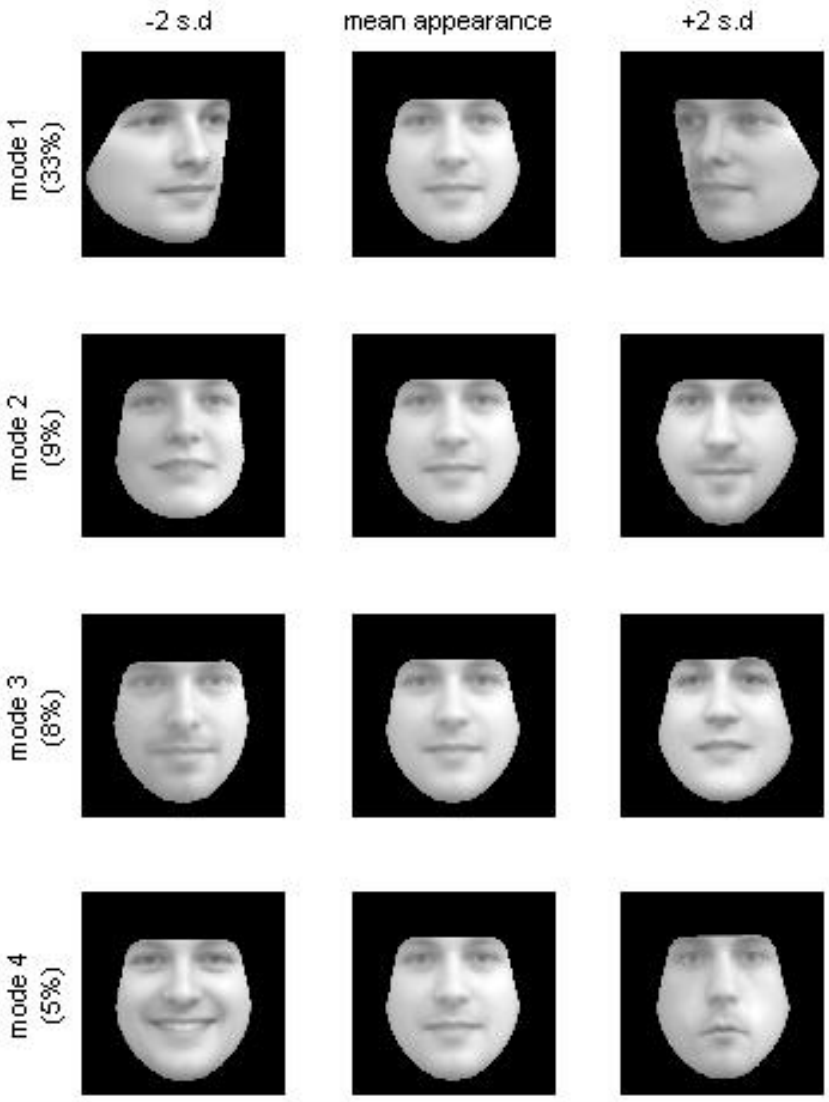

Figure 5.5: Appearance model of *all* faces in the faces data set, depicting variations in both shape and gray level. The middle column depicts the mean appearance, with left and right columns showing $\pm$ 3 s.d. for each of the three most significant modes of variation. Percentages cumulatively represent the portion of total variations in the set of images used to build the model.

where $\mathbf{W}_s$ is a diagonal matrix of weights for each shape parameter, allowing for the difference in units between the shape and gray-level models.

This leaves us with a set of appearance vectors $\mathbf{a}_i, i = 1,\ldots,s$ that represent both the shape and gray levels of the objects. Since there may be correlations between the shape and gray-level variations, we apply a further PCA to these vectors, obtaining the model

$$\mathbf{a} = \mathbf{P}_a \mathbf{b}_a$$

where $\mathbf{P}_a$ is a set of orthogonal modes of appearance variation, and $\mathbf{b}_a$ is a set of appearance parameters controlling both the shape and gray levels of the mode. Note the absence of the mean appearance vector $\bar{\mathbf{a}}$ from the expression. Mean appearance is necessarily zero, as both $\mathbf{b}_s$ and $\mathbf{b}_g$ have zero mean.

Due to the linear nature of the model, we can express the shape and grey-level vectors directly as functions of appearance parameters

$$\begin{aligned}\mathbf{s} &= \bar{\mathbf{s}} + \mathbf{P}_s \mathbf{W}_s^{-1} \mathbf{P}_{as} \mathbf{b}_a \\ \mathbf{g} &= \bar{\mathbf{g}} + \mathbf{P}_g \mathbf{P}_{ag} \mathbf{b}_a\end{aligned}$$

where

$$\mathbf{P}_a = \begin{pmatrix} \mathbf{P}_{as} \\ \mathbf{P}_{ag} \end{pmatrix}$$

in such a way that the number of rows of $\mathbf{P}_{as}$ corresponds to the length of the shape vector $\mathbf{s}$, and the number of rows of $\mathbf{P}_{ag}$ corresponds to the length of the gray-level vector $\mathbf{g}$.

To summarize:

$$\begin{aligned}\mathbf{s} &= \bar{\mathbf{s}} + \mathbf{Q}_s \mathbf{b}_a \\ \mathbf{g} &= \bar{\mathbf{g}} + \mathbf{Q}_g \mathbf{b}_a\end{aligned}$$

where

$$\begin{aligned}\mathbf{Q}_s &= \mathbf{P}_s \mathbf{W}_s^{-1} \mathbf{P}_{as} \\ \mathbf{Q}_g &= \mathbf{P}_g \mathbf{P}_{ag}\end{aligned}$$

For given appearance parameters $\mathbf{b}_a$, we can synthesize an image by calculating shape and gray-level vectors. We generate the shape-free patch from the gray-level vector, and we obtain landmark points from the shape vector. Finally, the shape-free patch is warped to the shape.

5.4.1 Choice of shape parameter weights

When concatenating the shape and gray-level parameters, we need to take into account the difference in units between the shape and gray-level models, so we multiply each shape parameter with a certain weight. Those weights are obtained by systematically displacing each shape parameter for the images in the training set, and measuring the effect of the displacement on the sample $\mathbf{g}$. The root-mean-square change in $\mathbf{g}$ per unit change in shape parameter b_{s_i} gives the weight w_{s_i} to be applied to that parameter.

A simple alternative is to set $\mathbf{W}_s = r\mathbf{I}$, where r^2 is the ratio of the total intensity variation to the total shape variation.

When implementing our appearance models we chose the other, simpler, alternative. So, all the shape parameters were multiplied with the scalar r, and calculating the shape vector $\mathbf{s}$ from the appearance parameters $\mathbf{b}_a$ reduced to

$$\mathbf{s} = \bar{\mathbf{s}} + \frac{1}{r}\mathbf{P}_s\mathbf{P}_{as}\mathbf{b}_a$$

5.5 All our models

All of the appearance models we built have in common that one doesn't need to think about the nature of the modeled objects, or about the choice of the landmark points. All of the data can simply be fed into this statistical model. The only data-specific aspects of the model concern the sizes of the objects in terms of translating and cropping, the paths used when visualizing shapes, and possibly the warping method used, as some data proved more difficult to warp than others.

We were therefore able to build a variety of face models, and we could simply choose which faces to include from the data set of 240 faces.

All faces

Our first complete model, incorporating shape, gray level, and appearance, was based on *all* faces in the faces data set. A total of 228 images were used to build the model (12 of the 240 original images proved to be unwarpable, cf. 4.2), and all facial expressions (neutral, smiling, "joker"), positions (frontal, oblique), and lighting effects (diffuse and source lighting) are included. For all the models, we present only the modes of variation in *appearance*. Figure 5.5 presents the first four modes, which, given the diversity of the original images, reveals a sundry collection of variance. The most significant mode of variation is the occurrence of oblique views, perhaps not surprisingly, as it indeed meets the eye with jarring effect. Other modes of variation seem to include wide and round vs. narrow and jutting chin structure, source-lighting, and upturned vs. downturned mouths. Again, it is impossible to guess the complete nature of a single mode of variation, but certainly some characteristics are more obvious than others.

Frontal faces

Figure 5.6 illustrates appearance models of our second complete model, which was based on all *frontal* face images. Therefore, oblique-angled poses are not longer included in the training set, and do not affect the modes of variation. Smiles, frowns, source lighting, and general shape are all still in evidence, however. We start to see the representation of more subtle variance, such as in the thickness of the lips, or the presence of facial hair.

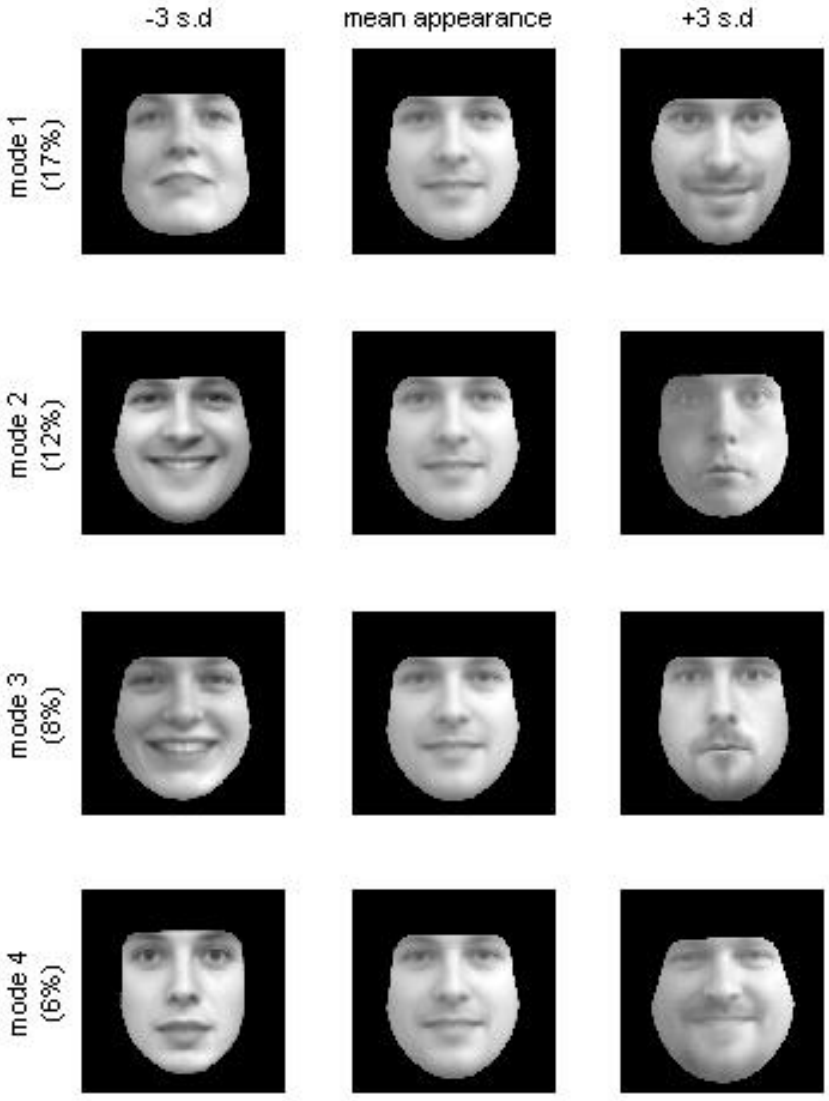

Figure 5.6: Appearance model of all *frontal* faces from the faces data set. All facial expressions and lighting conditions are represented.

Neutral faces

In our third model, we only those images with *neutral* expressions and diffuse lighting conditions. Figure 5.7 illustrates this model, whose modes of variation in appearance are becoming quite subtle. The first mode of variation seems to encompass the structure of the chin, as well as whether the features of the face turn upward or downward. In the second mode of variation, there seems to be a contrast of female vs. male characteristics.

Smiling faces

Our fourth model is similar to our neutral-faces model, above, except that we now kept only images of faces with *smiling* expressions. The first three modes of variation for this model can be seen in Figure 5.8. Overall shape of the face remains a significant factor in all three modes, and the quality of the smiles—toothy, tight-lipped, wide, narrow—also plays a role.

It is most interesting to note that the modes of variation—along with the people seen in the pairs of opposites—in the neutral set of faces in Figure 5.7 nearly exactly correspond to those in Figure 5.8, though the faces appear on opposite sides in each the two figures. We can calculate the modes of variation within a range of standard deviations from the mean, but this is done arbitrarily; in other words, the algorithm does not, for example, put small noses always on the left and large noses always on the right. So chins may be wide at -3 standard deviations in one model, while in another model, chins might be wide at $+3$. The similarity of the faces depicted in these two images is furthermore less surprising, given that the same 40 people are represented in both data sets. Correspondence is not perfect, because other factors are of course in play, such as the quality of the smile, and so on.

Cardiac cross sections

Our fifth complete model (see Fig. 5.9) was made from the cardiac data set, which comprised just 14 images. Still, the model did produce modes of variation, for example the overall shape (round/oblong), and the thickness of the chamber wall, which, as we learned previously, could be of significance for medical analysis.

Spines

Finally, our sixth complete model was based on the 20 spine images (Fig. 5.10). Gray-level normalization made it difficult for the vertebræthemselves to be seen in the images, so we have plotted the shapes atop the images. Even without having done so, the overall shape is very much apparent, with the first mode of variation showing obvious curvature as compared with the others.

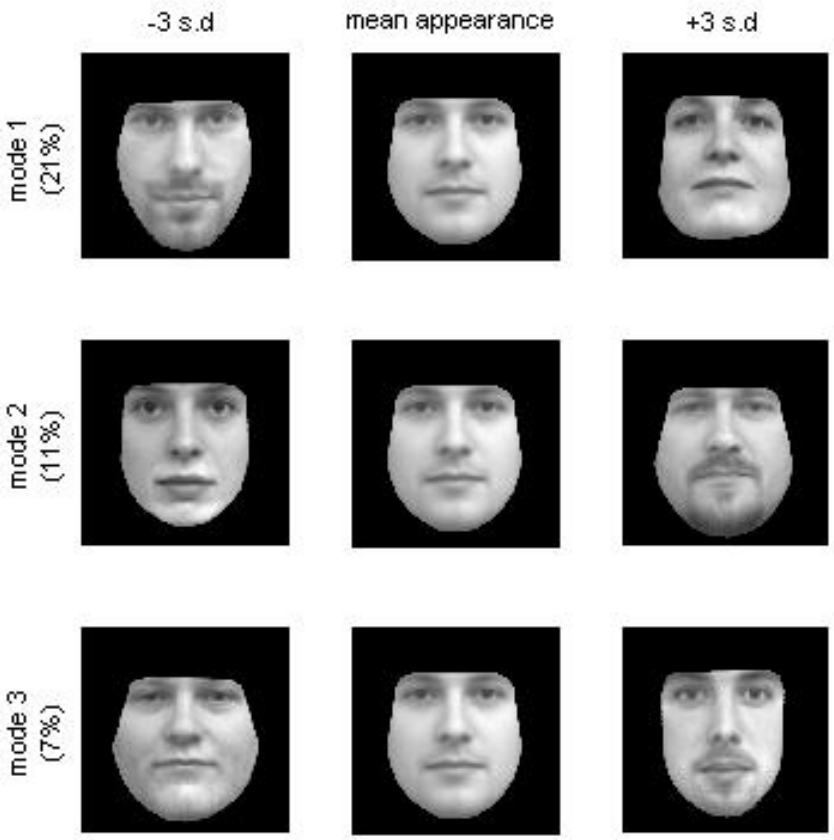

Figure 5.7: Appearance model of *neutral* faces and diffuse lighting conditions only. This model is based on 40 images in all.

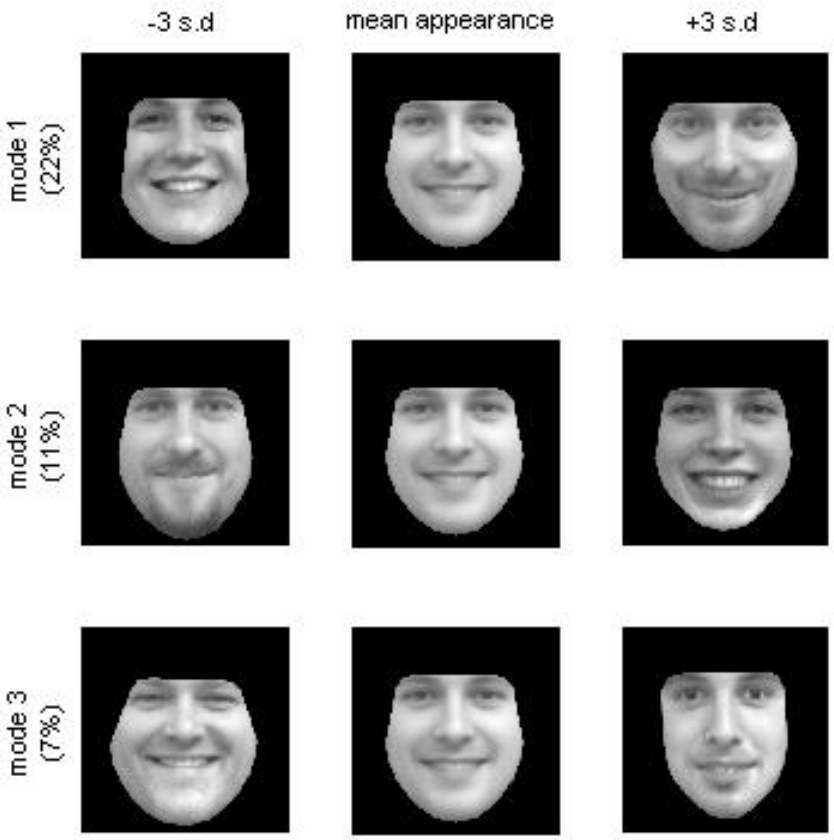

Figure 5.8: Appearance model of *smiling* faces only. This model is based on a total of 40 images.

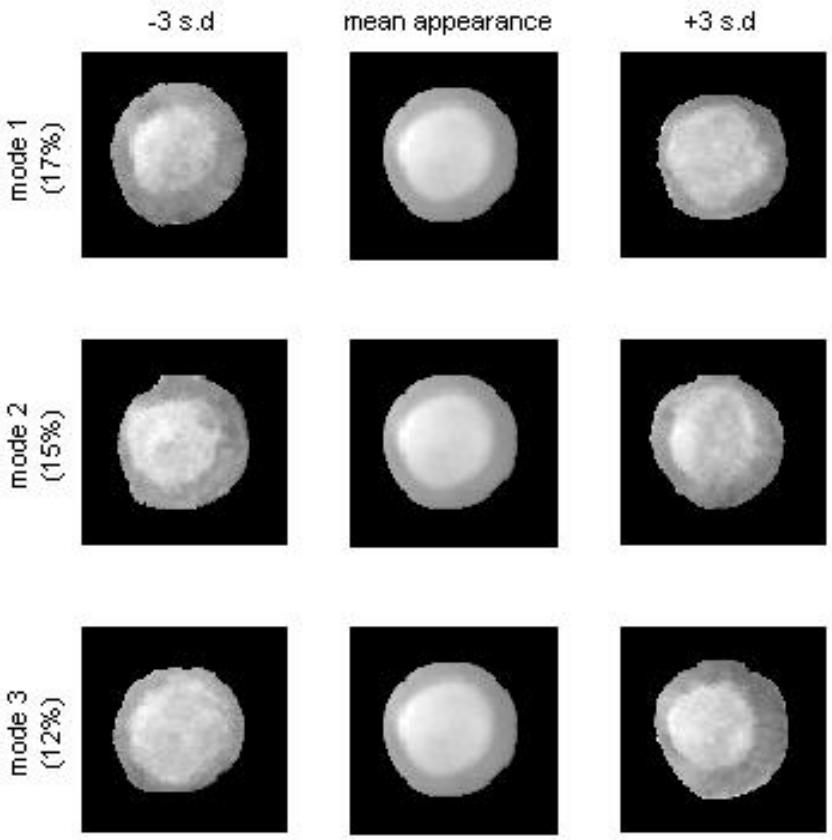

Figure 5.9: Appearance model for the cardiac cross sections, showing variations in both shape and gray level. As with previous figures, the middle column depicts the mean appearance, with left and right columns showing $\pm$ 3 s.d. for each of the three most significant modes of variation. Percentages cumulatively represent the portion of total variations in the set of images used to build the model.

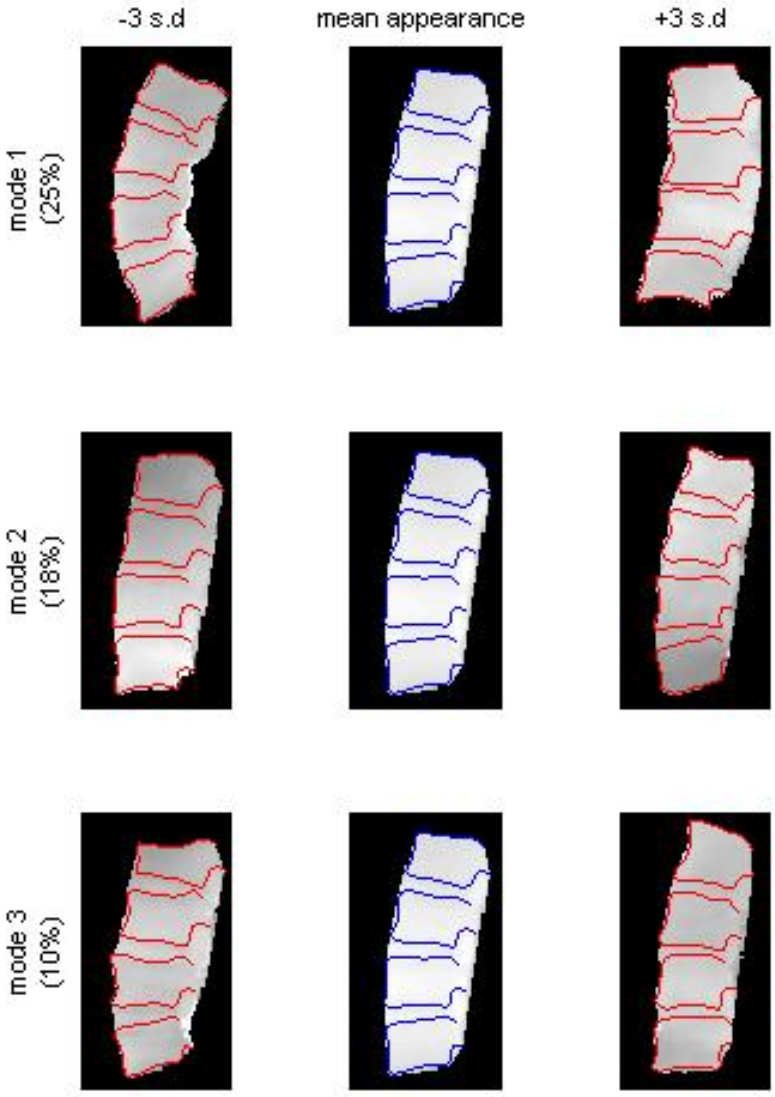

Figure 5.10: Spine appearance model with shape plotted on top, showing the first three modes of variation and ± 3 s.d.

Chapter 6

Manipulating Faces

6.1 Approximating faces

Having built shape, gray-level, and appearance models for a set of data, we can use those models to approximate images using just a subset of the all modes of variations. The images should be annotated with a set of landmark points.

For example, any shape **s** can be approximated as

$$\mathbf{s} \approx \bar{\mathbf{s}} + \mathbf{P}_s \mathbf{b}_s$$

where $\mathbf{P}_s$ is a sub-matrix of a matrix containing all modes of variation, and $\mathbf{b}_s$ is a projection of a parameter vector into a subspace of parameter space

$$\mathbf{b}_s = \mathbf{P}_s^T (\mathbf{s} - \bar{\mathbf{s}})$$

Similarly, any gray-level vector and any appearance vector can be approximated using the gray-level and appearance models.

We compared the two different ways of projecting. In both cases we started by creating a shape vector from the set of landmark points, and by creating a gray-level vector by sampling the image warped to the mean shape. We could then obtain the projected shape and gray-level parameters separately from the shape and gray-level vectors using the respective models. Alternatively, we could use the shape and gray-level vectors to create an appearance vector in the same way as when building the appearance model, and subsequently obtain projected appearance parameters from the appearance model.

We used approximation techniques on our faces data sets, but note that these techniques are in fact generic, and can be used in a variety of ways with other kinds of data.

In Figures 6.1 and 6.2, we see the comparison of the two methods. The number of parameters was always determined so that the model covered the same proportion of total variance for *each* model. We can immediately see that to cover a certain proportion of total appearance variance, we generally needed two-thirds of the modes in order to cover the same proportion of separated shape and gray-level

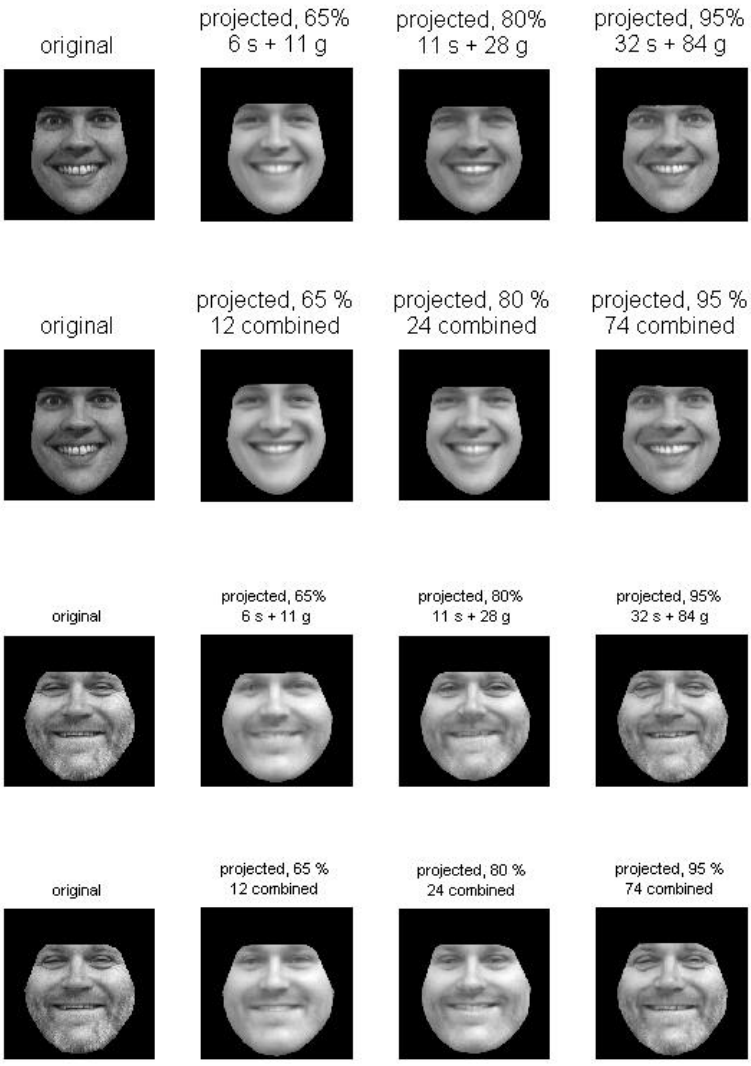

Figure 6.1: Approximating faces showing the comparison of two methods, using images taken from the training set. In rows 1 and 3, shape and gray-level models are used separately to recreate the original image. In rows 2 and 4, the shape and gray-level appearance vectors are combined to replicate the original. Efficiency of the combined method is markedly better, in that only 2/3 the number of modes are required to produce a result comparable to the separated shape/gray-level method.

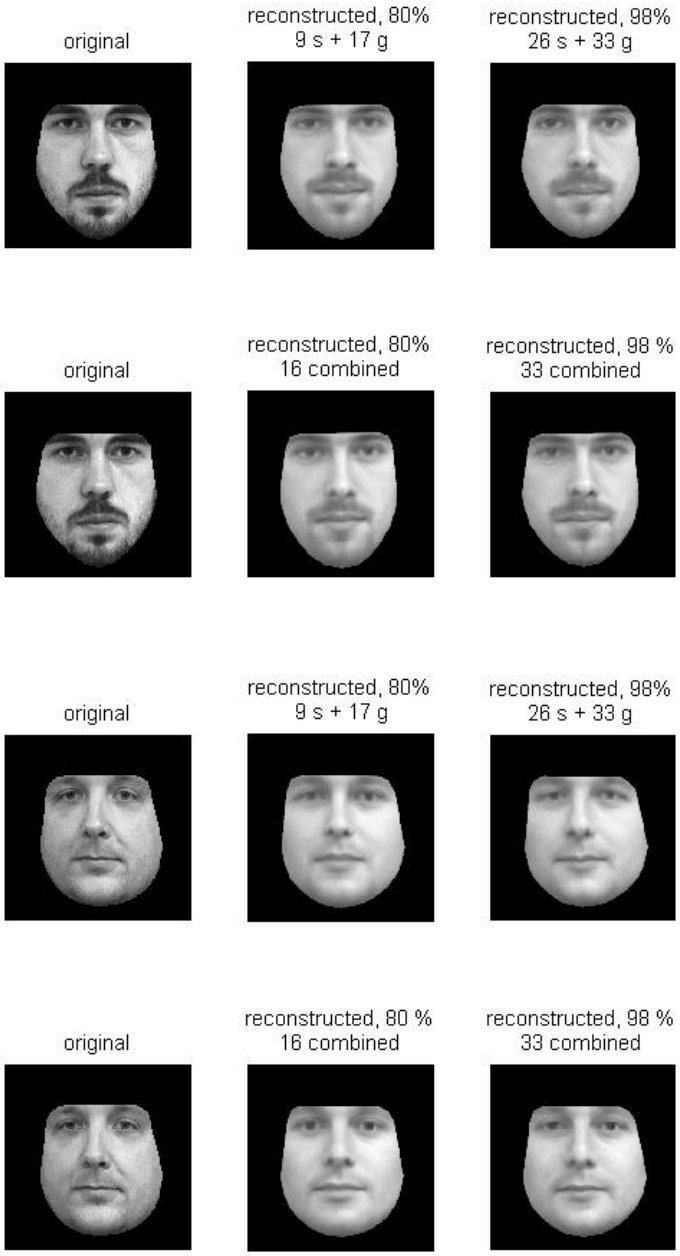

Figure 6.2: Approximating faces showing the comparison of two methods, using images not included in the training set. In rows 1 and 3, shape and gray-level models are used separately to recreate the original image. In rows 2 and 4, the shape and gray-level appearance models are combined to replicate the original. Both methods produce comparable results, though the latter method requires fewer than 2/3 the number of modes.

variance. This presents considerable savings in terms of the number of modes, with comparable results.

We evaluated the quality of the approximation subjectively, by simply looking at the images. The quantitative method that measures the quality of the fit can be implemented, but it would not provide a result of value, as we do not invert the gray-level normalization—the reason the resulting images are generally lighter than originals.

In fact, to provide a complete reconstruction, the appropriate pose (scale, translation, and rotation) should also be applied, and our algorithm does not provide pose-matching. Instead we pre-aligned the test images with the main shape for the model.

For Figure 6.1, the two faces were taken directly from the training set, so we were attempting to approximate faces that we already "knew": this could be used as a kind of face coder. We found the results quite satisfactory. For the approximations in Figure 6.2, we started with 40 neutral-face images. We made a training set of 37 of the face images, which left 3 for the purposes of testing. Our model was based on these 37 training images and their original landmark points. Approximation of the new test images was obviously less successful than in Figure 6.1, but given the small training set, it was not wholly unexpected, and we were satisfied with the result we did achieve.

6.2 Generating caricatures

Now begins a bit of fun for our project. We used our appearance model to generate caricatures and "anti-images" from original images, with some selected examples shown in Figure 6.3. For the caricatures, we generated images by *multiplying* the vector of parameters by a certain number, usually 2, sometimes 3. So in the parameter space, the resulting images are 2 (or 3) times further from the mean appearance than the original, but on the line connecting the mean and the original. Facial features were thereby exaggerated linearly. For example, if the original's eyes were lighter in color than the mean, the resulting caricature would feature very light eyes. If a mouth was slightly downturned compared with the mean, the caricature face would show a more exaggerated frown, and so on. The resulting images looked very much like hand-drawings of professional caricaturists, and we enjoyed them very much.

The anti-images are also the results of manipulating the appearance parameters, presenting the exact *opposite* of the original. If the original face was narrow, the anti-image would have a broad, round face. If the original had a downturned mouth, the anti-image would have a smile. Thin lips counter-matched full lips. Squinting eyes became round, open eyes. There were some surprising results as well, for example, original images with dark facial hair could result in either an anti-image with white facial hair, or no facial hair at all. One original image of a woman produced an anti-image that was quite masculine; this is possibly because the vast

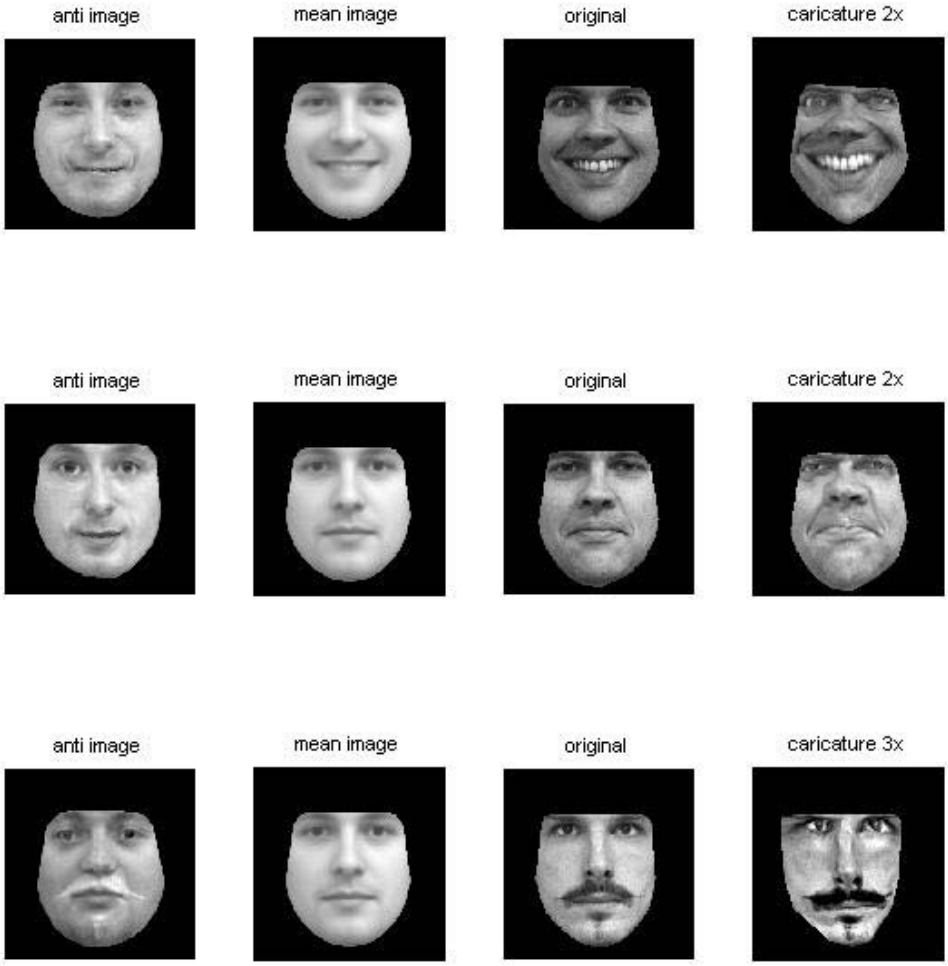

Figure 6.3: This and next page: Anti-images and caricatures. The mean appearance and original images are included for comparison. These images were produced using the neutral-face and smiling-face models.

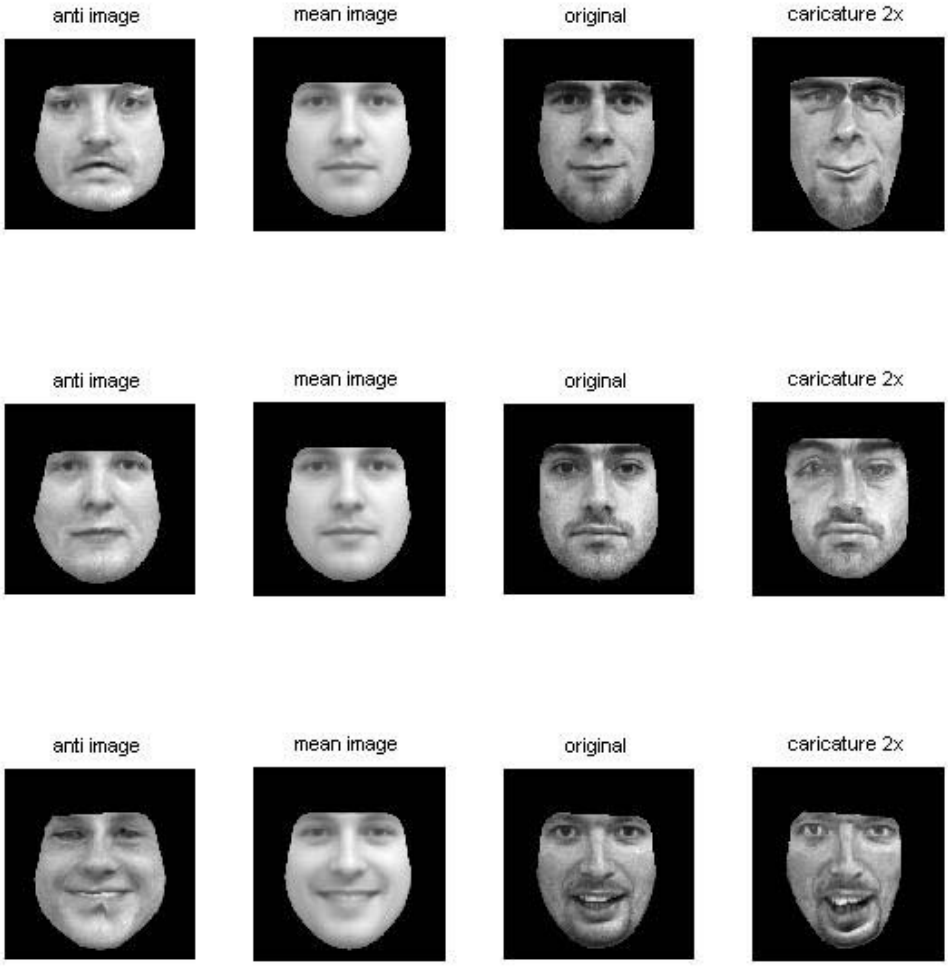

Figure 6.4: Anti-images and caricatures, continued. The mean appearance and original images are included for comparison. These images were produced using the neutral-face and smiling-face models.

majority of images used to build the appearance model were men, and our mean appearance was therefore decidedly masculine.

6.3 Forced smiles

We had images of 40 people with both neutral and smiling expressions, and so we decided to see if we could produce a smile on a neutral face using our model. We left five people (10 images) for testing our model, and used remaining 35 people (70 images) to build the model on. There is a variety of ways to model facial expressions. We implemented and tested two: mean smile and nearest-neighbor smile.

To add a mean smile to a neutral face, we looked at all of our faces in parameter space. We calculated the mean parameters for a neutral face and a mean parameter for a smile face. The difference between mean smile parameters and mean neutral parameters is a mean smile vector, and any neutral face can be forced to smile by adding the mean smile vector to its parameters. The mean smile vector is characteristic for an entire training set, and it makes all faces smile in the same way.

A basic idea behind the other method, nearest-neighbor smile, is that people with similar facial features probably smile in similar ways. To add the nearest-neighbor smile to a neutral face that is not from the same data set, we looked at the face in parameter space and found its nearest neutral neighbor from the training set. We then looked at the way that face smiled, that is, we found the smile vector for that particular face by subtracting the neutral expression parameters from the smile parameters. We than added the smile vector to our testing face.

Figure 6.5 illustrates some examples of applying the smile to neutral faces. We also included the original smile—the images of the real-life smiles belonging to the faces that we tried to make smile.

When testing our model, we noticed the rather broad variety of the nearest neighbor smiles, caused by the fact that the people in the original smile images smiled however they liked. In other words, the authors of the data set obviously didn't specify what kind of smile the person should make when being photographed.

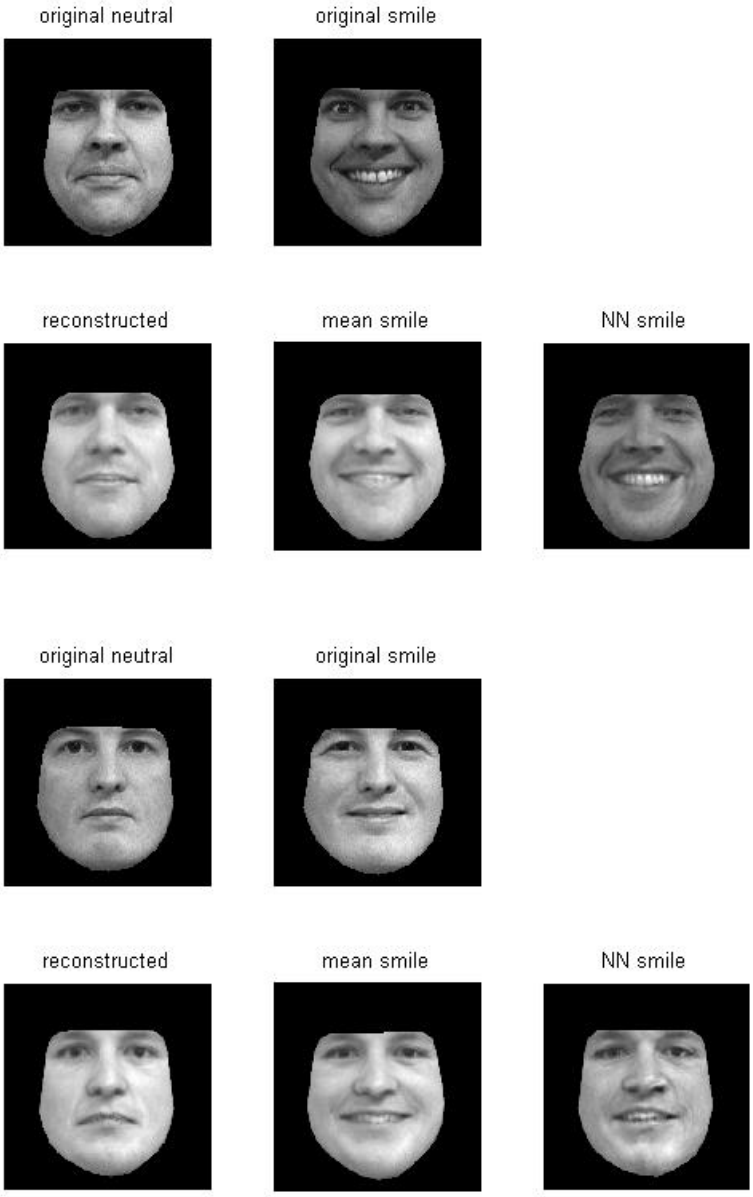

Figure 6.5: This and next page, forced smiles: applying a smile to neutral faces. The original images of the subjects actually smiling are included for comparison with our generated smiles.

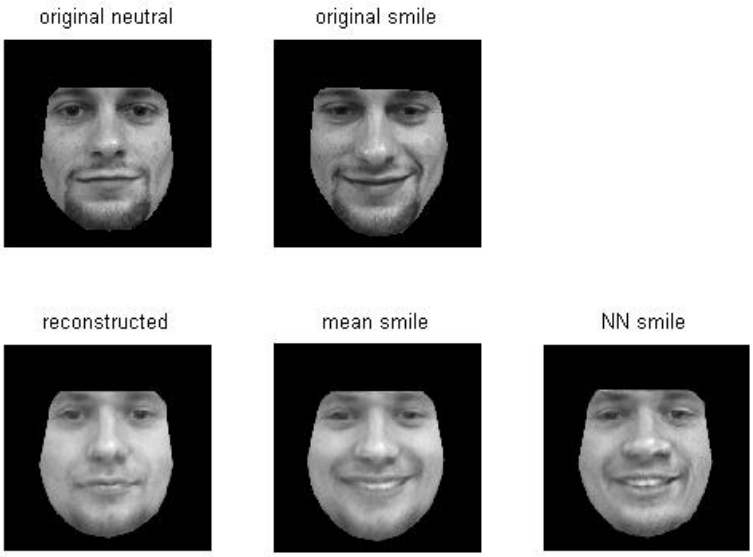

Figure 6.6: Forced smiles, continued. Applying a smile to neutral faces. The original images of the subjects actually smiling are included for comparison with the generated smiles.

Chapter 7

Active Appearance Models

Statistical appearance models, as presented in Chapter 5 can be used to search images for new instances of the objects they represent. This is active appearance modeling (AAM). In this chapter we will present the basic idea of the AAM search.

Although this idea is simple, after some initial implementation attempts it proved rather demanding (to implement)—we were dealing with large data, and processing times were lengthy. In some attempts the available memory ran out before the task could be completed. There are ways of manipulating data to deal with these problems, but we could not fit them into a four-week project. However, since we did examine the theory and implementation of AAM, we decided to include it in the report.

AAM search is an optimization method method based on two fundamental algorithms: part of the optimization common to all searches is learned in advance in a learning algorithm, and the optimization is finalized in case-specific iterative model refinement. We will introduce both algorithms here.

7.1 Learning

AAM search is treated as the optimization problem, that is, trying to minimize the difference between the actual image and the synthesized object delivered by a model. This difference $\delta\mathbf{g}$ is based on the difference between gray-level vectors

$$\begin{aligned} \delta\mathbf{g} &= \mathbf{g}_{\text{image}} - \mathbf{g}_{\text{model}} \\ &= \mathbf{g}_{\text{image}} - \mathbf{g}(\mathbf{b}_a) \end{aligned}$$

where $\mathbf{b}_a$ is vector of appearance parameters used to synthesize the image.

The dimensionality of the appearance models has been reduced by PCA, but it is generally still quite high, making this problem to appear as a cumbersome, highly dimensional optimization. Still, Cootes and Taylor [3] propose a simple and elegant AAM search solution.

The key observation is that each search is a similar optimization problem, and that it is possible to learn how to solve those problems in advance using the training

images. It is proposed that the spatial pattern of difference sample $\delta\mathbf{g}$ can be used to predict the needed correction of the model parameters $\delta\mathbf{b}_a$, which will lead to a better fit, i.e. which will minimize $\delta\mathbf{g}$. The simplest model we can arrive at constitutes a linear relationship

$$\delta\mathbf{b}_a = \mathbf{A}\delta\mathbf{g}$$

Cootes et al. (2001 [2]) show that this simplistic approximation suffices to produce a satisfactory result.

For a moment, let's consider the size of the matrix $\mathbf{A}$. The length of the parameter correction vector $\delta\mathbf{b}_a$ is the number of modes of variation, and it certainly does not exceed (and very probably is equal to) the number of example images, s. (It is actually $s-1$, but let's ignore the one.) The length of the gray-level error vector $\delta\mathbf{g}$ corresponds to the number of pixels sampled when building the appearance model, m, and this can be quite large. The matrix $\mathbf{A}$ must have size $s \times m$—a rather large matrix.

The matrix $\mathbf{A}$ needs to be estimated, and this is the core of the learning process. A set of experiments p is conducted, displacing the appearance parameters of the images in the training set and recording the effect of the change on the gray-level vector. Alternatively, images generated by the model can be used. Parameter changes are recorded in the $s \times p$ matrix $\delta\mathbf{B}$, and the changes in the gray-level vector are recorded in the $m \times p$ matrix $\delta\mathbf{G}$. The experimental results are then fed into the multivariate regression framework to obtain $\mathbf{A}$ from the relationship

$$\delta\mathbf{B} = \mathbf{A}\delta\mathbf{G}$$

Without going into detail about multivariate regression, let's just observe that this is an underdetermined system. The principal component regression can be a path toward estimating $\mathbf{A}$, which entails projecting the large $\mathbf{G}$ matrix into a certain subspace that captures the major part of the variation in $\mathbf{G}$.

What information does the matrix $\mathbf{A}$ carry, and is there a way of visualizing this information? The ith row of the matrix $\mathbf{A}$ is a length m vector corresponding to the change in the ith parameter. This row can be visualized as the image patch, and it gives the weights attached to different areas of the sampled patch—the areas that exhibit the largest variations when the ith parameter is changed have the largest weights.

Implementing the learning algorithm also raises some practical questions: How many displacements should one use, and how large should these displacements be? Should the parameters be displaced at once or separately, randomly or systematically? Some of the answers can be obtained only through experimentation, and the Master's thesis of Mikkel Stegmann [9] provided many valuable answers.

However, the learning algorithm alone was already too-big a bite for the four-week project period, and when we found we could not succeed with it to the degree we felt would be appropriate, we refocused our efforts back to building our appearance models and working with them in other ways.

In Figure 7.1 we show the result of our attempts to estimate matrix **A**. Both patches in the figure are the visualizations of the first row of matrix **A**, that row corresponding to the change in the first parameter. The patch on the left side of the figure was created using our normal model, with images size 280×280 pixels. We didn't use the regression method, and we can see how seriously undetermined matrix **A** is, despite the fact that the process of twice randomly displacing the parameters for each of the 40 training images, making 80 displacements in total, took more than an hour.

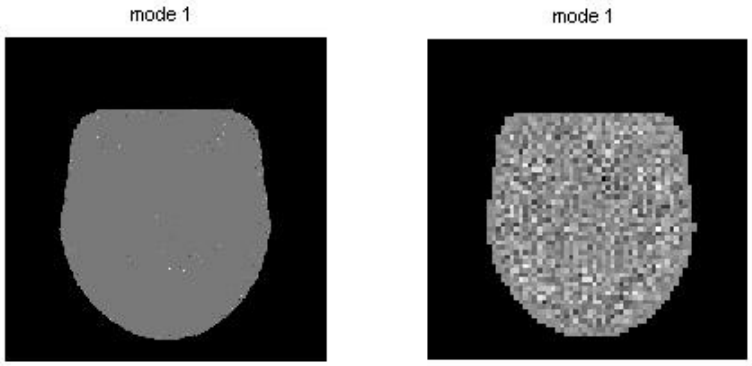

Figure 7.1: Estimating matrix **A**. Both patches are the visualizations of the first row of matrix **A**, the row corresponding to the change in the first parameter. The left patch was created using our normal model, with images size 280×280 pixels. The right patch was obtained after downsampling the images to a smaller size.

The right patch was obtained after downsampling the images to size 70×70 pixels, and changing the learning algorithm so that it systematically displaced each of the 39 parameters twice for each of the 40 images, resulting in 3120 displacements. The improvement is obvious, but we still didn't get the smooth resulting image as presented in the literature.

7.2 Iterative Model Refinement

Once obtained, the result of the learning algorithm produces a firm foundation for the optimization in AAM search. Given a method for predicting parameter correction needed to be made to achieve a better fit, we can construct an iterative search method.

1. Initially, the model object synthesized using the parameters $\mathbf{b}_a$ is placed over the image to be searched, and the image is sampled below the model object.

2. Error vector $\delta\mathbf{g} = \mathbf{g}_{\text{image}} - \mathbf{g}_{\text{model}}$, and error $|\delta\mathbf{g}|^2$ are evaluated.

3. The predicted displacement $\delta\mathbf{b}_a = \mathbf{A}\delta\mathbf{g}$ is computed.

4. The candidate parameter vector is found by $\mathbf{b}_{a\text{ new}} = \mathbf{b}_a + \delta\mathbf{b}_a$, using $k = 1$.

5. The object is synthesized using candidate parameters, the image is sampled below the object, and the error vector and error are calculated.

6. If the fit has improved, the candidate parameter can be accepted, otherwise $k = 1.5$, $k = 0.5$, $k = 0.25$, etc. should be tried until the best fit is found.

7. If an improvement has been made in the last iteration, return to step 2.

Just to restate, the material presented in this chapter covers the basic idea of AAM search, and we did not get into the details of dealing with the pose (scaling, translation, and rotation) of the objects across the image. Our algorithm could not take pose and overall gray-level into consideration, as we had planned on pre-aligning and pre-normalizing the testing images.

Chapter 8

Conclusions

Computer Science seems to encompass at least two schools of thought. In one, we try to make computers think and compute for us in ways that our feeble brains cannot. In the other, we try to make computers think like us, a process that is always more complicated, painstaking, and precise than one would intuitively expect. We found the same with our project. For example, active appearance models perform the same function that we do ourselves when we "discover" a new face on the street: in the blink of an eye, we match the stranger's appearance to the model of expectations we have assembled over the course of our life. If the parameters all seem to lie within the mean range, we might not look twice. However, if they do seem to vary beyond the mean, we might crane our neck for one or two more search iterations. We do this automatically, and we do it continually, all our lives.

AAM is one method for computer vision. Building statistical shape and appearance models from a set of annotated images allows us to represent both the mean shape and the appearance of objects, as well as their significant modes of variation. We can use AAM to locate similar objects in new images.

By the end of the project period, we found ourselves thoroughly impressed by the depth and breadth of this field. We realized that the scope of our project could easily accommodate a great many more aspects, which we wish we had the time to pursue. And we now also appreciate how the scope of this topic itself could be expanded to include more work with active appearance models, tracking and segmentation, and other applications.

We are happy with the amount of literature we managed to swallow during this period, the number of implementations we were able to make, and the amount of coverage we have written. We have limited the scope of our modeling procedures to images of the face, spine, and heart, but it would be nice to have the opportunity to make our own data: take photographs, for example, and do the annotation ourselves. We now know it is a process for the patient-of-heart, but the success of the model depends on it. We also think it would be fun to try our hand at 3-D appearance modeling, for example, modeling the shape and gray level of such 3-D surfaces as a human face—or play with specific facial-expressions models, such as

age progression, or emotion.

In all, this was an enjoyable and thoroughly fascinating topic for us, and we look forward to exploring it further one day.

Appendix A

Matlab code

By the end of the project, we found ourselves with quite a large collection of MATLAB functions. Only some data-specific functions used for visualization are not included here. This body of code also may also be found online at www.itu.dk/people/vedrana/AAM. We have grouped the functions into sections consistent with the rest of the report. Major functions are given first, with the smaller, helping functions presented at the end.

A.1 Preprocessing

All the manipulation of the data during preprocessing was done in a single `data` cell-array, so that a cell `data{i}` contains all information about *i*th image:

`data{i}{1}` contains matrix representation of the image.

`data{i}{2}` contains landmark point coordinates in $n \times 2$ matrix.

`data{i}{3}` contains additional information about landmark points, closed/open, inner/outer... This was not used.

`data{i}{4}` contains matrix representation of the image warped to the mean shape. We decided not to overwrite original image, but we could have done that too.

```
function data = preprocessing(data,size,warp_type)
% performs all the preprocessing of the data
% size is the new size of the images
%-----------------------------------------

tic
data = translate_and_crop(data,size);
[data, mean_s] = align_shapes(data);
data = mask_images(data);
data = warp_to_mean(data, mean_s,warp_type);
data = mask_warped(data, mean_s);
data = normalize_gray(data, mean_s);
toc
```

```
function data = translate_and_crop(data,size)
% translates and croppes images (and coresponding landmark points)
% to a desired size in such a way that the center of gravity of
% landmark points is ends the center of the image.
%---------------------------------------------------------------

m = length(data);

for i=1:m
    fprintf('... Translating and cropping image %g out of %g.\n',i,m);
    y_center = round(sum(data{i}{2}(:,1))/length(data{i}{2}(:,1)));
    x_center = round(sum(data{i}{2}(:,2))/length(data{i}{2}(:,2)));
    y_min = y_center - size(2)/2;
    x_min = x_center - size(1)/2;
    data{i}{1} = data{i}{1}(x_min:x_min+size(1)-1,y_min:y_min+size(2)-1);
    data{i}{2}(:,1) = data{i}{2}(:,1)-y_min+1;
    data{i}{2}(:,2) = data{i}{2}(:,2)-x_min+1;
end
```

```
function [data, mean_s] = align_shapes(data)
% perfomrs iterative alignment of all shapes
%-------------------------------------------

max_iter = 10;
m = length(data);
[y x] = size(data{1}{1});

% reading shape data
for i=1:m
    sh(:,:,i) = data{i}{2};
end

len = norm(mean(sh,3));
mean_s = fix(mean(sh,3));

% iterative shape alignment - finding the best mean
for iter=1:max_iter
    fprintf('... Aligning, iteration %g out of %g.\n',iter,max_iter);
    for i=1:m
        sh(:,:,i) = align(sh(:,:,i),mean_s);
    end
    new_mean_s = fix(align(mean(sh,3),mean_s));
    change(iter) = norm(new_mean_s-mean_s);
    mean_s = new_mean_s;
end
figure, plot(1:max_iter,change), axis([0 max_iter+1 -0.00005 inf])
xlabel('iteration number'), ylabel('change of the mean shape estimate')
mean_s = mean_s*len;
```

```
% applying the final alignment
for i=1:m
    fprintf('... Aligning image %g out of %g.\n',i,m);
    cp = data{i}{2};
    tform = cp2tform(cp,mean_s,'linear conformal');
    data{i}{1} = imtransform(data{i}{1},tform,'XData',[1 x],'YData',[1 y]);
    data{i}{2} = align(cp,mean_s);
end

function output = align(input,target)
% aligns landmark points of the input image with the target image
%---------------------------------------------------------------

tform = cp2tform(input,target,'linear conformal');
output = (fliplr(flipud(tform.tdata.T(1:2,1:2)))*input')'+...
        ones(size(input,1),1)*(tform.tdata.T(3,1:2));

function output = fix(input)

output = input/norm(input);
```

```
function data = mask_images(data)
% applies a mask to each image, leaving just a convex hull obtained
% from landmark points
%---------------------------------------------------------------

m = length(data);
[x y] = size(data{1}{1});

[X_matrix,Y_matrix] = coordinate_matrices(x,y);

for i=1:m
    fprintf('... Masking image %g out of %g.\n',i,m);
    im = data{i}{1};
    cp = data{i}{2};
    K = convhull(cp(:,1),cp(:,2));
    data{i}{1} = uint8(im).*uint8(inpolygon(X_matrix,Y_matrix,cp(K,1),cp(K,2)));
end
```

```
function data = warp_to_mean(data,mean_s,warp_type)
% warps an image to a shape free image
```

```
% warped images are added to data, original images are not overwriten
%--------------------------------------------------------------------’

m = length(data);

% reading data
for i=1:m
    cm(:,:,i) = data{i}{2};
end

% warping and adding warped images to database
for i=1:m
    fprintf('... Warping image %g out of %g.\n',i,m);
    data{i}{4} = im_warp(data{i}{1},data{i}{2},mean_s,warp_type);
end
```

```
function data = mask_warped(data,mean_s)
% aditional masking of warped images to eliminate posible unwanted border
% effect after the warping
%-------------------------------------------------------------------------

m = length(data);
[x y] = size(data{1}{1});

% reading data
for i=1:m
    cm(:,:,i) = data{i}{2};
end

% finding and applying mean shape mask
mean_mask = create_mask(mean_s,x,y);
for i=1:m
    fprintf('... Masking image %g out of %g.\n',i,m);
    data{i}{4} = data{i}{4}.*mean_mask;
end
```

```
function data = normalize_gray(data,mean_s)
% perfomrs iterative normalization of all shape-free patches
%-----------------------------------------------------------

max_iter = 10;
m = length(data);
```

```
[x y] = size(data{1}{1});

% find mean shape mask
in = find(create_mask(mean_s,x,y));

% sample inside the mask, and make initial normalization
a = [];
for i=1:m
    fprintf('... Reading image %g out of %g.\n',i,m);
    a = [a, fix(double(data{i}{4}(in)))];
end

% iterative normalization - finding the best mean
for iter=1:max_iter
    mean_a = fix(mean(a,2));
    for i=1:m
        a(:,i) = normalize(a(:,i),mean_a);
    end
    change(iter)=norm(mean_a-fix(mean(a,2)));
    fprintf('... Normalizing, iteration %g out of %g.\n',iter,max_iter);
end
figure, plot(1:max_iter,change), axis([0 max_iter+1 -0.1 inf]),
    xlabel('iteration number'), ylabel('change of the mean appearance estimate')
mean_a = fix(mean(a,2));

% appying the final normalization and returning to uint8
min_a = min(min(a));
max_a = max(max(a));
for i=1:m
    fprintf('... Normalizing image %g out of %g.\n',i,m);
    data{i}{4}(in) = uint8(255*(a(:,i)-min_a)/(max_a-min_a));
end

function output = normalize(input,target)

alpha = input'*target;
beta = sum(input)/length(input);
output = (input-beta)/alpha;

function output = fix(input)

output = (input-mean(input))/std(input);
```

A.2 Building the Models

All the information about shape, gray-level and appearance models is stored in a single `model` cell-array, so that a cell `data{1}` contains all information about shape model, cell `data{2}` contains all information about gray-level model and cell `data{3}` contains all information about appearance model:

`data{i}{1}` contains mean vector.

`data{i}{2}` contains a matrix composed of eigenvectors.

`data{i}{3}` contains a vector of eigenvalues.

`data{i}{4}` contains a vector of percentage of a total variation.

`data{i}{5}` contains a matrix composed of all data vectors.

The gray-level cell-array has a couple of additional information needed for reconstruction:

`data{2}{6}` contains indices of pixels inside the sampled area.

`data{2}{7}` contains original size of the image.

The appearance cell-array has additional information:

`data{3}{6}` contains a weighting factor.

```
function model = build_model(data)
% builds all three models
%---------------------------------------

model = build_shape_model(data);
model = build_gray_model(data,model);
model = build_combined_model(data,model);
```

```
function model = build_shape_model(data)

m = length(data);

% reading shape data
shapes = [];
for i=1:m
    shapes = [shapes, data{i}{2}(:)];
end

% building shape model
[mean_s,phi,lambda,perc_var] = calculate_model(shapes);

% updating model
model{1}{1} = mean_s;
model{1}{2} = phi;
model{1}{3} = lambda;
model{1}{4} = perc_var;
model{1}{5} = shapes;
```

```
function model = build_gray_model(data,model)
% builds gray-level model
%-------------------------------------------

m = length(data);
orig_size = size(data{1}{1});
mean_s = model{1}{1};

% find mean shape mask
in = find(create_mask(break_vector(mean_s),orig_size(1),orig_size(2)));

% sample inside the mask
grays = [];
for i=1:m
    grays = [grays, double(data{i}{4}(in))];
end

% building gray-level model
[mean_g,phi,lambda,perc_var] = calculate_model(grays);

% updating model
model{2}{1} = mean_g;
model{2}{2} = phi;
model{2}{3} = lambda;
model{2}{4} = perc_var;
model{2}{5} = grays;
model{2}{6} = in;
model{2}{7} = orig_size;
```

```
function model = build_combined_model(data, model)
% builds a combined model from the shape and gray model
%-----------------------------------------------------

% reading shape model and gray model
mean_s = model{1}{1};
phi_s = model{1}{2};
lambda_s = model{1}{3};
shapes = model{1}{5};
mean_g = model{2}{1};
phi_g = model{2}{2};
lambda_g = model{2}{3};
grays = model{2}{5};

% building combined model
weight = (sum(lambda_g)/sum(lambda_s))^0.5;
combined = [weight*phi_s'*(shapes-repmat(mean_s,1,size(shapes,2)));...
    phi_g'*(grays-repmat(mean_g,1,size(grays,2)))];
[mean_c,phi,lambda,perc_var] = calculate_model(combined);
```

```
% updating model
model{3}{1} = mean_c;
model{3}{2} = phi;
model{3}{3} = lambda;
model{3}{4} = perc_var;
model{3}{5} = combined;
model{3}{6} = weight;
```

```
function [mean_d, phi, lambda, perc_var] = calculate_model(data_matrix)
% performes principal component analysis on the data
%--------------------------------------------------------------------

mean_d = mean(data_matrix,2);
[phi,score,lambda] = princomp(data_matrix',’econ’);
perc_var = lambda*100/sum(lambda);
```

```
function model_variations(nom, max_var, model, warp_type, type)
% nom - number of modes displayed
% max_var - limits for mode variation
% type - (optional) type of data: faces, hearts, spines (’f’, ’h’, ’s’)
% first figure - shape variations of mean-gray image
% second figure - gray variations of shape-free image
% third figure - combined appearance variations
% eventualy fourth figure - shape variations
%--------------------------------------------------------------------

par = diag(max_var*ones(1,nom));
for figure_nr = 1:3
    figure,
    perc_s = model{figure_nr}{4};
    for mode=1:nom
        image = generate_image((-par(mode,:)),model,figure_nr,warp_type);
        subplot(nom,3,(mode-1)*3+1), im_show(image)
            if mode==1, title(sprintf(’-%d s.d’,max_var)), end
            ylabel(sprintf(’mode %d\n(%d%%)’,mode,round(perc_s(mode))))
        image = generate_image(0,model,figure_nr,warp_type);
        subplot(nom,3,(mode-1)*3+2), im_show(image)
            if mode==1
                if figure_nr==1 title(sprintf(’mean shape’))
                elseif figure_nr==2 title(sprintf(’mean gray’))
                elseif figure_nr==3 title(sprintf(’mean appearance’))
```

```
                end
            end
        image = generate_image(par(mode,:),model,figure_nr,warp_type);
        subplot(nom,3,(mode-1)*3+3), im_show(image)
            if mode==1, title(sprintf('+%d s.d',max_var)), end
    end
end

if nargin==5 shape_variations(nom,max_var,model,type), end


function image = generate_image(par,model,type,warp_type)

if type==1 image = generate_shape_image(par,model,warp_type);
elseif type==2 image = generate_gray(par,model);
elseif type==3 image = generate_combined(par,model,warp_type);
end
```

```
function shape_variations(nom, max_var, model, type)
% nom - number of modes displayed
% max_var - limits for mode variation
% type - (optional) type of data: faces, hearts, spines ('f', 'h', 's')
%-----------------------------------------------------------------------

par = diag(max_var*ones(1,nom));
figure
for mode=1:nom
    perc_s = model{1}{4};
    shape = generate_shape((-par(mode,:)),model);
    subplot(nom,3,(mode-1)*3+1), shape_plot(shape,'r',type)
        if mode==1, title(sprintf('-%d s.d',max_var)), end;
        ylabel(sprintf('mode %d\n(%d%%)',mode,round(perc_s(mode))))
    subplot(nom,3,(mode-1)*3+2), shape_plot(model{1}{1},'b',type)
        if mode==1, title('mean shape'), end;
    shape = generate_shape(par(mode,:),model);
    subplot(nom,3,(mode-1)*3+3), shape_plot(shape,'r',type)
        if mode==1, title(sprintf('+%d s.d',max_var)), end;
end
```

A.3 Generating Images

```
function image = generate_combined(parameters, model, warp_type)

% reading shape, gray and appearance models
mean_s = model{1}{1};
phi_s = model{1}{2};
mean_g = model{2}{1};
```

```
phi_g = model{2}{2};
in = model{2}{6};
orig_size = model{2}{7};
mean_c = model{3}{1};
phi_c = model{3}{2};
lambda = model{3}{3};
weight = model{3}{6};

% generating shape and shape-free gray image
t = length(parameters);
c = ((lambda(1:t)).^0.5).*parameters(:);

Q_s = phi_s*phi_c(1:size(phi_s,2),:)/weight;
Q_g = phi_g*phi_c(size(phi_s,2)+1:end,:);

s = mean_s + Q_s(:,1:t)*c;
g = mean_g + Q_g(:,1:t)*c;

gray = zeros(orig_size);
gray(in) = uint8(g);

% warping image to shape
image = im_warp(gray,break_vector(mean_s),break_vector(s),warp_type);
shape_mask = create_mask(break_vector(s),orig_size(1),orig_size(2));
image = uint8(image).*uint8(shape_mask);
```

```
function image = generate_shape_and_gray(param_s, param_g, model, warp_type)

% reading shape and gray models
mean_s = model{1}{1};
phi_s = model{1}{2};
lambda_s = model{1}{3};
mean_g = model{2}{1};
phi_g = model{2}{2};
lambda_g = model{2}{3};
in = model{2}{6};
o_size = model{2}{7};

% generating shape and shape-free gray image
s = generate_shape(param_s, model);
gray = generate_gray(param_g, model);

% warping image to shape
image = im_warp(gray,break_vector(mean_s),break_vector(s),warp_type);
shape_mask = create_mask(break_vector(s),o_size(1),o_size(2));
image = uint8(image).*uint8(shape_mask);
```

```
function shape = generate_shape(parameters, model)

% reading shape model
mean_s = model{1}{1};
phi = model{1}{2};
lambda = model{1}{3};

% generating shape
t = length(parameters);
shape = mean_s + phi(:,1:t)*(((lambda(1:t)).^0.5).*parameters(:));
```

```
function image = generate_gray(parameters, model)

% reading gray model
mean_g = model{2}{1};
phi = model{2}{2};
lambda = model{2}{3};
in = model{2}{6};
orig_size = model{2}{7};

% generating shape-free gray image
t = length(parameters);
g = mean_g + phi(:,1:t)*(((lambda(1:t)).^0.5).*parameters(:));
image = zeros(orig_size);
image(in) = uint8(g);
```

```
function image = generate_shape_image(parameters, model, warp_type)

% reading model
mean_s = model{1}{1};
o_size = model{2}{7};

% generating shape and main gray image
s = generate_shape(parameters, model);
gray = generate_gray(0, model);

% warping main gray image to shape
image = im_warp(gray,break_vector(mean_s),break_vector(s),warp_type);
shape_mask = create_mask(break_vector(s),o_size(1),o_size(2));
image = uint8(image).*uint8(shape_mask);
```

```
function show_reconstructed(test,indices,perc,model,warp_type);

J=length(perc);
for i=1:length(indices)
    index = indices(i);
    orig = test{index}{1};
    figure, subplot(2,J+1,1), im_show(orig), title('original')
            subplot(2,J+1,J+2), im_show(orig), title('original')
    for j=1:J
        per = perc(j);
        nom = modes_needed(per,model); % just for the sake of displaying it
        reco = reconstruct_shape_and_gray(test,index,per,model,warp_type);
        subplot(2,J+1,j+1), im_show(reco)
            title({'reconstructed,';sprintf('%2d%%',per);...
                    sprintf('%2d s + %2d g',nom(1),nom(2))})
        reco = reconstruct_combined(test,index,per,model,warp_type);
        subplot(2,J+1,J+j+2), im_show(reco),
            title({'reconstructed,';sprintf('%d %%',per);...
                    sprintf('%2d combined',nom(3))})
        end
    end
end

function par_c = project_combined(test,index,nom,model,warp_type)

shape = test{index}{2};
image = test{index}{1};

mean_s = model{1}{1};
phi_s = model{1}{2};
lambda_s = model{1}{3};
mean_g = model{2}{1};
phi_g = model{2}{2};
lambda_g = model{2}{3};
in = model{2}{6};
o_size = model{2}{7};
phi_c = model{3}{2};
lambda_c = model{3}{3};
weight = model{3}{6};

gray = im_warp(image,shape,break_vector(mean_s),warp_type);
g = gray(in);
b = [weight*phi_s'*(shape(:)-mean_s);phi_g'*(double(g)-mean_g)];
par_c = phi_c(:,1:nom(3))'*b./((lambda_c(1:nom(3))).^0.5);
```

```
function [par_s,par_g] = project_shape_and_gray(test,index,nom,model,warp_type)

shape = test{index}{2};
```

```
image = test{index}{1};

mean_s = model{1}{1};
phi_s = model{1}{2};
lambda_s = model{1}{3};
mean_g = model{2}{1};
phi_g = model{2}{2};
lambda_g = model{2}{3};
in = model{2}{6};
o_size = model{2}{7};

par_s = ((phi_s(:,1:nom(1)))'*(shape(:)-mean_s))./((lambda_s(1:nom(1))).^0.5);
gray = im_warp(image,shape,break_vector(mean_s),warp_type);
g = gray(in);
par_g = (phi_g(:,1:nom(2)))'*(double(g)-mean_g)./((lambda_g(1:nom(2))).^0.5);
```

```
function image = reconstruct_combined(test,index,perc,model,warp_type);

shape = test{index}{2};
image = test{index}{1};

nom = modes_needed(perc,model);
par_c = project_combined(test,index,nom,model,warp_type);
image = generate_combined(par_c, model,warp_type);
```

```
function image = reconstruct_shape_and_gray(test,index,perc,model,warp_type);

shape = test{index}{2};
image = test{index}{1};

nom = modes_needed(perc,model);
[par_s,par_g] = project_shape_and_gray(test,index,nom,model,warp_type);
image = generate_shape_and_gray(par_s, par_g, model,warp_type);
```

A.4 Manipulating Faces

```
function show_caricature_combined(victims,indices,perc,times,model,warp_type);

nom = modes_needed(perc,model);

for i=1:length(indices)
```

```
    try
        index = indices(i);
        shape = victims{index}{2};
        orig = victims{index}{1};
        [par_c] = project_combined(victims,index,nom,model,warp_type);
        anti = generate_combined(-1*par_c, model,warp_type);
        zero = generate_combined(0, model,warp_type);
        cari = generate_combined(times*par_c, model,warp_type);
        figure
        subplot(141), im_show(anti), title('anti image')
        subplot(142), im_show(zero), title('mean image')
        subplot(143), im_show(orig), title('original')
        subplot(144), im_show(cari), title('caricature')
    catch
        fprintf('...Gave up on image %2d\n',i);
    end
end
```

```
function show_caricature_shape_and_gray(victims,indices,perc,times,model,warp_type);

nom = modes_needed(perc,model);


for i=1:length(indices)
    try
        index = indices(i);
        shape = victims{index}{2};
        orig = victims{index}{1};
        [par_s,par_g] = project_shape_and_gray(victims,index,nom,model,warp_type);
        anti = generate_shape_and_gray(-1*par_s, -1*par_g, model,warp_type);
        zero = generate_shape_and_gray(0, 0, model,wapr_type);
        cari = generate_shape_and_gray(times*par_s, times*par_g, model,warp_type);
        figure
        subplot(141), im_show(anti), title('anti image')
        subplot(142), im_show(zero), title('mean image')
        subplot(143), im_show(orig), title('original')
        subplot(144), im_show(cari), title('caricature')
    catch
        fprintf('...Gave up on image %2d\n',i);
    end
end
```

```
function show_applied_smiles(testing,indices,training,perc,warp_type)

model = build_model(training);
[msv,par_neut,par_smile,n] = smile_parameters(model,training,perc,warp_type);

for i=1:length(indices)
    index=2*i-1;
    figure
    subplot(231), im_show(testing{index}{1}), title('original neutral')
    subplot(232), im_show(testing{index+1}{1}), title('original smile')

    reconstructed = reconstruct_combined(testing,index,perc,model,warp_type);
    subplot(234), im_show(reconstructed), title('reconstructed')
    [app_ms,app_NNs] = apply_smile(testing,index,model,msv,par_neut,...
            par_smile,n,warp_type);
    subplot(235), im_show(app_ms), title('mean smile')
    subplot(236), im_show(app_NNs), title('NN smile')
end
```

```
function [app_ms,app_NNs] = apply_smile(testing,index,model,msv,...
        par_neut,par_smile,n,warp_type);

param = project_combined(testing,index,n,model,warp_type);
app_ms = generate_combined(param + msv, model,warp_type);

for i=1:size(par_neut,2)
    dist(i) = norm(param - par_neut(:,i));
end

k = find(dist == min(dist));
app_NNs = generate_combined(param + par_smile(:,k) - par_neut(:,k),...
        model,warp_type);
```

```
function [msv,par_neut,par_smile,n] = smile_parameters(model,training,...
        perc,warp_type)

n = modes_needed(perc,model);

for i=1:length(training)/2;
    par_neut(:,i) = project_combined(training,2*i-1,n,model,warp_type);
    par_smile(:,i) = project_combined(training,2*i,n,model,warp_type);
end

msv = mean(par_smile,2) - mean(par_neut,2);
```

A.5 Active Appearance Models

```
function model = learning(data,model)

tic
l = length(data);
nom(3) = length(model{3}{3});
warp_type = 'l';
in = model{2}{6};

move = [0.5 -0.5];
%move = [0.5 -0.5 0.25 -0.25];

% initializing with empty matrices
D_g = [];
D_c = [];

for index=1:40
    c = project_combined(data,index,nom,model,warp_type);
    g = data{index}{1}(in);
    for k=1:nom(3)
        fprintf('Learning from image %g out of %g, changing parameter %g out of %g\n',...
            index,l,k,nom(3));
        for n = 1:length(move)
            try
                delta_c = zeros(nom(3),1); delta_c(k)=move(n);
                changed = generate_combined(c+delta_c,model,warp_type);
                delta_g = double(changed(in))-double(g);
                D_g = [D_g,delta_g(:)];
                D_c = [D_c,delta_c(:)];
            catch
                fprintf('... Gave up\n');
            end
        end
    end
    save('D_g','D_g')
    save('D_c','D_c')
end

A = D_c/D_g;
model{4}{1} = A;
model{4}{2} = nom;
toc
```

```
function show_weights(nom,model)

in = model{2}{6};
orig_size = model{2}{7};
```

```
w = zeros(orig_size);
A = model{4}{1};

for i=1:nom
    w(in) = fix(A(i,:));
    subplot(1,nom,i),
    im_show(w);
end

function o = fix(i)

o = uint8((i-min(i))*255/(max(i)-min(i)));
```

A.6 Helping Functions

```
function cp = break_vector(vector);
% breaks length 2n vector in size (n,2) matrix
%-------------------------------------------

cp = zeros(length(vector)/2,2);
cp(:) = vector;
```

```
function [X_mat,Y_mat] = coordinate_matrices(x,y)
% matrices needed for inpolygon function
%-------------------------------------------------

X_mat = [];
for i=1:y
    X_mat = [X_mat,i*ones(x,1)];
end

Y_mat = [];
for i=1:x
    Y_mat = [Y_mat;i*ones(1,y)];
end
```

```
function mask = create_mask(cp,size_x,size_y)
% creates a mask covering a convex hull of landmark points
%-----------------------------------------------------------

[X_matrix,Y_matrix] = coordinate_matrices(size_x,size_y);
K = convhull(cp(:,1),cp(:,2));
mask  = uint8(inpolygon(X_matrix,Y_matrix,cp(K,1),cp(K,2)));
```

```
function fix_axis

axis image, axis ij, box on %, axis off
```

```
function im_show(im)

imagesc(im), colormap gray, axis image, axis ij
```

```
function output = im_warp(input,cp,target_cp,warp_type)

% piecewise linear
if warp_type=='l'
    [x y] = size(input);
    tform = cp2tform(cp,target_cp,'piecewise linear');
    output = imtransform(input,tform,'XData',[1 y],'YData',[1 x]);
% tin-plate splines
elseif warp_type=='t'
    tpsInfo = calcTPSInfo([target_cp,cp]);
    output = deformImage(input,tpsInfo,1);
end
```

```
function n = modes_needed(perc,model)

if perc==100, for i=1:3, n(i) = length(model{i}{4}); end
else
    for i=1:3
        perc_var = model{i}{4};
        for k = 1:length(perc_var)
            tot(k) = sum(perc_var(1:k));
        end
        n(i) = min(find(tot >= perc));
    end
end
```

```
function shape_plot(coordinates,color,type)
% Takes in the coordinates of the shape in the form of
% length 2n vector, where n is the number of landmark points.
% Rows 1:n are x coordinates, and rows n+1:2n are y coordinates
%-------------------------------------------------------------
coordinates=break_vector(coordinates);

if type=='h'
    path{1} = [coordinates(1:33,:); coordinates(1,:)];
    path{2} = [coordinates(34:66,:); coordinates(34,:)];
elseif type=='f'
    path{1} = [coordinates(1:13,:)];
    path{2} = [coordinates(14:21,:); coordinates(14,:)];
    path{3} = [coordinates(22:29,:); coordinates(22,:)];
    path{4} = [coordinates(30:34,:)];
    path{5} = [coordinates(35:39,:)];
    path{6} = [coordinates(40:47,:); coordinates(40,:)];
    path{7} = [coordinates(48:58,:)];
elseif type=='s'
    path{1} = [coordinates(1:25,:)];
    path{2} = [coordinates(26:50,:)];
    path{3} = [coordinates(51:75,:)];
    path{4} = [coordinates(76:100,:)];
end

m = length(path);
hold on
for k=1:m
    plot(path{k}(:,1),path{k}(:,2),color)
end
hold off

fix_axis
if type=='h'
    axis([1 80 1 80])
elseif type=='f'
    axis([1 280 1 280])
elseif type=='s'
    axis([1 60 1 110])
end
```

```
function show_data(data,type)

for i=1:length(data)
    figure
        im_show(data{i}{1}), hold on
        shape_plot(data{i}{2}(:),'b',type)
        shape_plot(data{i}{2}(:),'b.',type)
```

```
        hold off
end
```

```
function show_data_warped(data)

m = length(data);

for i=1:m
    figure
        subplot(211)
        im_show(data{i}{1}), title(i)

        subplot(212)
        im_show(data{i}{4})
end
```

```
function show_faces(data,indices,sh_p)

for n=1:ceil(length(indices)/6)
    figure
    for j=1:6
        if (n-1)*6+j<=length(indices)
            index = indices((n-1)*6+j);
            subplot(2,3,j),
            im_show(data{index}{1}), title(index)
            if sh_p==1
                hold on, shape_plot(data{index}{2}(:),'b','f'),
                shape_plot(data{index}{2}(:),'b.','f'), hold off
            end
        end
    end
end
```

```
function show_faces_warped(data,indices,sh_p,mean)

for n=1:ceil(length(indices)/3)
```

```
    figure
    for j=1:3
        if (n-1)*3+j<=length(indices)
            index = indices((n-1)*3+j);
            subplot(2,3,j)
            im_show(data{index}{1}), title(index)
            if sh_p==1
                hold on, shape_plot(data{index}{2}(:),'b','f'),
                shape_plot(data{index}{2}(:),'b.','f'), hold off
            end
            subplot(2,3,j+3)
            im_show(data{index}{4})
            if mean~=0
                hold on, shape_plot(mean(:),'b','f'),
                shape_plot(mean(:),'b.','f'), hold off
            end

        end
    end
end
```

Bibliography

[1] F. L. Bookstein. Principal warps: Thin-plate splines and the decomposition of deformations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(6):567–585, 1989.

[2] T. Cootes, G. J. Edwards, and C. Taylor. Active appearance models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(6):681–685, 2001.

[3] T. Cootes and C. Taylor. Statistical models of appearance for medical image analysis and computer vision. In *Proc. SPIE Medical Imaging*, pages 1–14, Dept. Imaging Science and Biomedical Engineering, University of Manchester, U.K., 2001.

[4] T. Cootes and C. Taylor. Statistical models of appearance for computer vision. Dept. Imaging Science and Biomedical Engineering, University of Manchester, U.K., 2004.

[5] A. Lanitis, T. Cootes, and C. Taylor. Toward automatic simulation of aging effects on face images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(4):442–455, 2002.

[6] M. M. Nordstrøm, M. Larsen, J. Sierakowski, and M. B. Stegmann. The IMM face database: An annotated dataset of 240 face images. Informatics and Mathematical Modelling, The Technical University of Denmark, Richard Petersens Plads, Building 321, DK-2800 Kgs. Lyngby, Denmark, 2004.

[7] N. Sebe. Software decodes Mona Lisa's enigmatic smile. *New Scientist*, page 25, 2005.

[8] L. I. Smith. A tutorial on principal component analysis. February 26, 2002.

[9] M. B. Stegmann. Active Appearance Models: Theory, Extensions and Cases. Master's thesis, The Technical University of Denmark, Lyngby, 2000. IMM-EKS-2000-25.

[10] M. B. Stegmann. An annotated dataset of 14 cardiac MR images. 2002. Informatics and Mathematical Modelling, Technical University of Denmark, Richard Petersens Plads, Building 321, DK-2800 Kgs. Lyngby, Denmark.

[11] M. B. Stegmann, B. K. Ersbøll, and R. Larsen. FAME: A flexible appearance modelling environment, 2003.

[12] T. Vetter and S. Romdhani. ECCV 2004 Tutorial: Face recognition and modeling. Second Part. Pose and illumination invariant face modeling and recognition. Switzerland, 2004. http://informatik.unibas.ch.