

# Face Image Analysis using a Multiple Features Fitting Strategy

Inauguraldissertation

zur  
Erlangung der Würde eines Doktors der Philosophie  
vorgelegt der  
Philosophisch-Naturwissenschaftlichen Fakultät  
der Universität Basel  
von

Sami Romdhani

aus Brüssel, Belgien

Basel, 2005

Genehmigt von der Philosophisch-Naturwissenschaftlichen Fakultät

auf Antrag von

Prof. Dr. Thomas Vetter, Universität Basel, Dissertationsleiter  
Prof. Dr. Andrew Zisserman, University of Oxford, Korreferent

Basel, den 14.12.2004

Prof. Dr. Hans-Jakob Wirz, Dekan

## Abstract

The main contribution of this thesis is a novel algorithm for fitting a Three-Dimensional Morphable Model of faces to a 2D input image. This fitting algorithm enables the estimation of the 3D shape, the texture, the 3D pose and the light direction from a single input image.

Generally, the algorithms tackling the problem of 3D shape estimation from image data use only the pixels intensity as input to drive the estimation process. This was previously achieved using either a simple model, such as the Lambertian reflectance model, leading to a linear fitting algorithm. Alternatively, this problem was addressed using a more precise model and minimizing a non-convex cost function with many local minima. One way to reduce the local minima problem is to use a stochastic optimization algorithm. However, the convergence properties (such as the radius of convergence) of such algorithms, are limited. Here, as well as the pixel intensity, we use various image features such as the edges or the location of the specular highlights. The 3D shape, texture and imaging parameters are then estimated by maximizing the posterior of the parameters given these image features. The overall cost function obtained is smoother and, hence, a stochastic optimization algorithm is not needed to avoid the local minima problem. This leads to the Multi-Features Fitting algorithm that has a wider radius of convergence and a higher level of precision.

The new Multi-Feature Fitting algorithm is applied for such tasks as face identification, facial expression transfer from one image to another image (of different individuals) and face tracking across 3D pose and expression variations.

The second contribution of this thesis is a careful comparison of well known fitting algorithms used in the context of face modelling and recognition. It is shown that these algorithms achieve high run time efficiency at the cost of accuracy and generality (few face images may be analysed).

The third and last contribution is the Matlab Morphable Model toolbox, a set of software tools developed in the Matlab programming environment. It allows (i) the generation of 3D faces from model parameters, (ii) the rendering of 3D faces, (iii) the fitting of an input image using the Multi-Features Fitting algorithm and (iv) identification from model parameters. The toolbox has a modular design that allows anyone to build on it and, for instance, to improve the fitting algorithm by incorporating new features in the cost function.



# Contents

<b>Notations</b>	<b>7</b>
<b>1 Introduction</b>	<b>9</b>
1.1 Fitting Algorithms Characteristics	11
<b>2 Image synthesis with the 3D Morphable Model</b>	<b>15</b>
2.1 3D Morphable Model Construction	15
2.1.1 Motivations of the representation	15
2.1.2 Preprocessing Laser Scans	17
2.1.3 Dense Correspondences Computed by Optical Flow	17
2.1.4 Face Space Based on Principal Components Analysis	19
2.1.5 Segmented Morphable Model	20
2.2 Morphable Model to Synthesise Images	21
2.2.1 Shape Projection	21
2.2.2 Inverse Shape Projection	22
2.2.3 Illumination and Colour Transformation	23
2.2.4 Image Synthesis	24
<b>3 Compendium of Face Image Analysis Techniques</b>	<b>25</b>
3.1 Correspondence-based methods	26
3.2 Illumination-based methods	31
<b>4 Multi-Features Fitting</b>	<b>35</b>
4.1 Using multiple features	36
4.1.1 Parallelism with pattern classification	36
4.1.2 Multiple feature based posterior	37
4.1.3 Feature Characteristics	38
4.2 Image based features	40
4.2.1 Pixel colour feature	40
4.2.2 Edge Feature	43
4.2.3 Anchor points feature	50
4.2.4 Specular highlight feature	51
4.3 Model based features	54
4.3.1 Gaussian Prior probability features	54
4.3.2 Texture Constraints Feature	55
4.4 Features Summary	56
4.5 Robust Fitting	57
4.6 Multi-Features Fitting Algorithm	59
4.7 Fitting Results	62
4.8 Texture extraction and illumination correction	63

<b>5 Applications</b>	<b>67</b>
5.1 Identification and verification . . . . .	67
5.2 Automatic facial expression transfer . . . . .	71
5.3 Face tracking without anchor points . . . . .	74
<b>6 Conclusions</b>	<b>79</b>
<b>A Derivative</b>	<b>83</b>
A.1 Derivative of the Specular highlight cost function . . . . .	83
<b>B Matlab Morphable Model Toolbox</b>	<b>85</b>
B.1 Installation . . . . .	85
B.2 Morphable Model Toolbox Tour . . . . .	86
B.3 Documentation . . . . .	89
B.4 3D Face Generation and Rendering . . . . .	89
B.5 3D Face Fitting - An Implementation of MFF . . . . .	91
B.6 Data Structures . . . . .	92
B.6.1 Morphable model . . . . .	92
B.6.2 Rendering parameters . . . . .	93
B.6.3 Fitting parameters . . . . .	94
B.6.4 Cost functions parameters . . . . .	98
<b>C Reproducing figures and experiments</b>	<b>101</b>
<b>List of figures and tables</b>	<b>103</b>
<b>Curriculum Vitae</b>	<b>107</b>
<b>Bibliography</b>	<b>109</b>

# Notations

<b>a</b>	Vectors are denoted by lowercase bold letters.
$\mathbf{a}_i$	Element $i$ of the vector $\mathbf{a}$ .
<b>A</b>	Matrices are denoted by uppercase bold letters.
$\mathbf{A}_{M \times N}$	Matrix with $M$ rows and $N$ columns.
$\mathbf{A}^T$	Transpose of the matrix $\mathbf{A}$ .
$\mathbf{A}_{i,j}$	Element $(i, j)$ of the matrix $\mathbf{A}$ . This is a scalar.
$\mathbf{A}_{\cdot,j}$	Column vector formed by the column $j$ of the matrix $\mathbf{A}$ .
<b>I</b>	Identity matrix whose dimension depends on the context.
$\mathbf{1}_{M \times N}$	$M \times N$ matrix for which all elements are equal to one.
$\text{vec}(\mathbf{A})$	Vectorisation of the matrix $\mathbf{A}$ . If $\mathbf{A}$ is a $M \times N$ matrix, $\text{vec}(\mathbf{A})$ is a $MN \times 1$ column vector.
$\mathbf{a}^{(M)}$	If $\mathbf{a}$ is a $R \times 1$ column vector, $\mathbf{a}^{(M)}$ is a $M \times R/M$ matrix. It is assumed that $R/M$ is an integer value.
$\langle \mathbf{a}, \mathbf{b} \rangle$	Scalar product of vectors $\mathbf{a}$ and $\mathbf{b}$ .
$\mathbf{a} \times \mathbf{b}$	Vector product of vectors $\mathbf{a}$ and $\mathbf{b}$ .

## 3D Morphable Model Notations

$N_v$	Number of vertices in the model.
$N_t$	Number of triangles in the model.
<b>S</b>	$3 \times N_v$ Matrix with a 3D shape.
<b>T</b>	$3 \times N_v$ Matrix with an RGB texture (one colour per shape vertex).
$N_S$	Number of shape principal components.
$N_T$	Number of texture principal components.
$\alpha$	$N_S \times 1$ Column vector with the shape parameters. In the case of a segmented face, it is a $N_S \times 4$ matrix with one column per segment.
$\beta$	$N_T \times 1$ Column vector with the texture parameters. In the case of a segmented face, it is a $N_T \times 4$ matrix with one column per segment.
$\sigma_{S,i}$	Standard deviation of the shape principal component $i$ .
$\sigma_{T,i}$	Standard deviation of the texture principal component $i$ .
$(u, v)$	The reference frame on which the 3D shapes and the RGB textures are represented is denoted by $(u, v)$ . The model vertices are located at integer values of $(u, v)$ .

## Rendering Notations

$\rho$	Column vector of the 2D projection parameters.
$\iota$	Column vector of the light and colour transformation parameters.
$\gamma$	Imaging parameters, i.e. concatenation of $\rho$ and $\iota$ .

$\mathbf{R}$	$3 \times 3$ Rotation matrix.
$\mathbf{W}$	$3 \times N_v$ 3D Shape matrix in world coordinates.
$\mathbf{n}_i^v$	$3 \times 1$ Unit-length direction normal to the face surface in vertex $i$ in object centred coordinates.
$\mathbf{n}_i^{v,w}$	$3 \times 1$ Unit-length direction normal to the face surface in vertex $i$ in world coordinates.
$\mathbf{n}_j^t$	$3 \times 1$ Unit-length direction normal to the triangle $t$ of the triangle list.
$\mathbf{d}$	$3 \times 1$ Unit-length direction of a light source.
$\mathbf{t}_i^I$	Colour $\mathbf{t}_i$ with illumination effects.
$\mathbf{t}_i^C$	Colour $\mathbf{t}_i^C$ after the camera colour transformation.
$\mathbf{t}_i$	$3 \times 1$ Vector with the R, G, B albedo colour of vertex $i$ .
$\mathbf{p}(u, v; \alpha, \rho)$	$2 \times 1$ Vector valued function with the X-Y projection of the model vertex $(u, v)$ . For a 3D shape in the span of the model, this projection depends on the shape parameters $\alpha$ and on the projection parameters $\rho$ .
$\mathbf{p}^{-1}(x, y; \alpha, \rho)$	$2 \times 1$ Vector with the inverse projection of $\mathbf{p}(u, v; \alpha, \rho)$ : projection of a image frame point $(x, y)$ into the reference frame $(u, v)$ .
$\mathbf{t}(u, v)$	$3 \times 1$ Vector valued function with the R, G, B colour intensities of a facial texture expressed on the reference frame. For integer values of $(u, v)$ in the reference frame domain, its values are given by a texture matrix: $\mathbf{t}(u_i, v_i) = \mathbf{T}_{.i}$ . For non integer values, $\mathbf{t}(u, v)$ is obtained by interpolation of $\mathbf{T}$ using the triangle list.

### Fitting Notations

$I(x, y)$	R, G, B colour of the input image arranged in a $3 \times 1$ column vector.
$I^m(x, y)$	R, G, B colour of the model image arranged in a $3 \times 1$ column vector.
$f^x(I(x, y))$	Feature function that extracts feature $x$ from an input image. The superscript, $x$ , may be $c$ , $e$ , $a$ , $s$ , $p$ , or $t$ to denote, respectively, the pixel colour feature, the edge feature, the anchor feature, the specular highlight feature, the prior probability feature, and the texture constraint feature.
$\mathbf{e}^c$	$3N_c^f \times 1$ Residual vector obtained when fitting the pixel colour feature to the incenter of $N_c^f$ triangles.
$\mathbf{e}_i^c$	$3 \times 1$ Sub-vector of $\mathbf{e}^c$ with the R, G, B, colour residual of fitting triangle $i$ . In the case of fitting a grey-level image this is a scalar.
$\mathbf{e}^e$	$N_e^f \times 1$ Residual vector of the edge feature obtained when fitting $N_e^f$ model edge points. ( $\mathbf{e}_i^e$ is a scalar).
$\mathbf{e}^a$	$2N_a^f \times 1$ Residual vector of the anchor point feature obtained when fitting $N_a^f$ anchor points.
$\mathbf{e}^s$	$3N_s^f \times 1$ Residual vector of the specular highlight feature obtained when fitting $N_s^f$ model triangle overlapping a specular highlight.
$\mathbf{e}^t$	$3N_c^f \times 1$ Residual vector of the texture constrained feature obtained when $N_c^f$ triangles are fitted by the pixel colour feature.

# Chapter 1

## Introduction

Face image analysis, in the context of computer vision, is, in general, about acquiring a high-level knowledge about a facial image. Its purpose is to answer typical questions such as follows: Who is the individual whose face is photographed? What is his emotional state? Is he happy or sad? What is its age? In which direction is a person looking? In these cases the task of face image analysis is to infer some knowledge from a photograph. Another aspect of this field is about synthesising a novel image given a photograph, and, hence, to answer questions such as follows: How would look like a given person if its emotional state would change to sad or happy or if he would mimic someone else? How would the face of a given person appear in ten years time? The work presented in this thesis endeavours to answer some of the aforementioned questions and has the potential to answer many more.

A facial image conveys many pieces of information such as the identity of a person, the gender of a person, his expression, his age, the illumination environment in which the photograph was taken, and the direction the person is looking at, relative to the camera. These are (some of) the sources of variations of a facial image. The task of face image analysis is hence, given a face photograph, to infer these pieces of information and to synthesise a novel facial image in which some of them are modified, while keeping the other constant. The problem of face image analysis has then two parts: The *inference* part aiming at estimating semantic knowledge from a face photograph; and the *synthesis* part that uses this knowledge to generate a novel facial image.

A unified approach addressing these two objectives is to construct a generative face image model that codes *all* the sources of variation *separately* and *independently*. The face image model should be generative such as it is capable of synthesising a photo-realistic face image. This model is then used to address the inference problem by estimating the model parameters that, when used for synthesising, produce the same facial image as the input photograph. This scheme is called an *analysis by synthesis* framework. Naturally, reproducing exactly the same facial image is generally not feasible. Hence, the analysis task should estimate the parameters synthesising an image that resembles as much as possible to the given facial image.

Clearly, an analysis by synthesis framework is made of two components: a generative model and an analysing algorithm, called fitting algorithm. The generative facial image model used here, which was not developed as part of this thesis work, but introduced in [10], is the *3D Morphable Model* (3DMM). An overview of its fundamental concepts, its construction and use to render a face image is made in Chapter 2. The main novelty of this thesis is a new fitting algorithm, called Multi-Features Fitting (MFF) algorithm, explained in Chapter 4. In Chapter 5, the 3D morphable model and the multi-features fitting algorithm are used to answer two of the aforementioned questions: In Section 5.1, our framework is used to identify a facial photograph; and in Section 5.2, the expression of the facial photograph of one individual is transferred to another photograph of someone else.

The first section of Chapter 3 gives a review of the major fitting algorithms proposed in the literature. It will be seen that they all treat the fitting problem as an optimisation problem. The major difference between them lies in the cost function that is to be minimised and in the

approximations of the image formation process being made. A commonality of these algorithms is that they follow a top-down strategy: It is assumed that the facial images belong to a class for which a model has been constructed. Hence, they rely on acquired class-specific knowledge. A typical work-flow of a top-down approach follows.

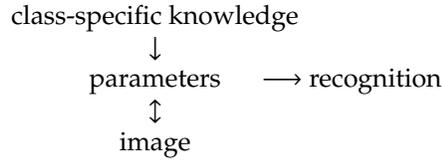


Figure 1.1: Top-down approach work-flow.

This is in contrast to earlier approaches in the field of computer vision, called bottom-up methods, which are based on estimating an internal 3D model from image data without knowledge of the object class nor of the recognition task at hand [9, 50]. This image analysis technique first extracts some features from the image, such as the contours, and use them to estimate a view-point invariant representation of the object analysed, on which, finally, recognition is performed. The work-flow of this process is outlined on Figure 1.2.



Figure 1.2: Bottom-up approach work-flow.

In the present work, we use a combination of top-down and bottom-up methods. Similarly to the top-down approach, we use a face specific model to drive the recognition to plausible result. However, in contrast to a purely top-down method, the image evidence used does not consist only on the image pixels but also on some features derived from the input image beforehand. Hence, the image analysis work-flow used in this work is depicted as follows.

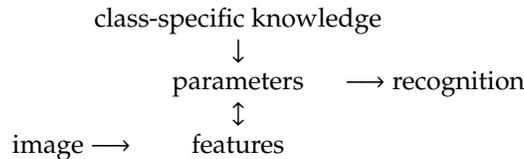


Figure 1.3: Integrated top-down/bottom-up approach work-flow adopted for the Multi-Features Fitting algorithm presented in this thesis.

I argue in this thesis that the convergence properties of a fitting algorithm improve if it uses a combination of top-down and bottom-up approaches, i.e. if not only the pixel intensity of the input image is used to drive the fitting but also some features, extracted from the image using a nonlinear process, such as the image edges or the location of the specular highlight. An example of this process is shown on Figure 1.4.

A major property of a cost function, which is to be optimised, is its convexity. A convex and smooth cost function has a single optimum. Hence, applied on this function, any optimisation algorithm would find this unique optimum that is also the global optimum. Thus, the image analysis algorithm would have the assurance to find the best description possible of an input image. In an analysis by synthesis approach, which is intrinsically a top-down method, the class-specific knowledge used is a model of the image formation process that is able to generate a photo-realistic image of an instance of the object studied. The task of the analysis algorithm is to invert this model: recover from an input image the model parameter able to instantiate it. However, as the image formation process is nonlinear, the cost function minimised during image analysis is non-convex. As a result, the image description yielded may not be optimal and can be very different from the real one. Thus, I developed a method that does not invert the full image

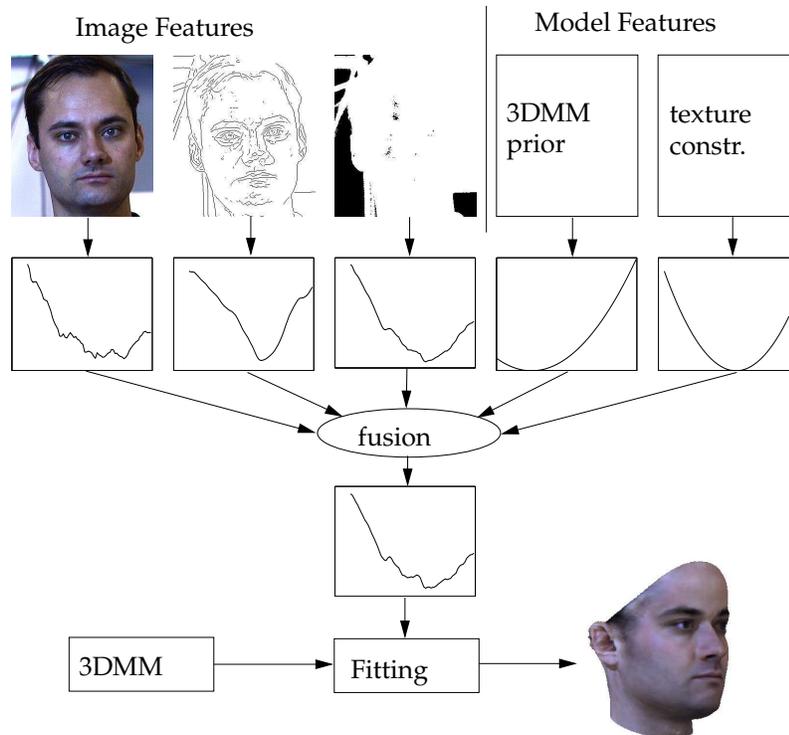


Figure 1.4: 3D Shape, texture and imaging parameters are estimated using the pixel intensity (page 40), the edges (page 43) and the specular highlights (page 51) detected in the input image, shown on the top row. Additionally, two model-based features are used: the 3D Morphable Model (3DMM) prior (page 54) and texture constraints (page 55). Second row: plots of each feature function cost along the azimuth angle. These cost functions are combined (page 37) yielding a smoother function that the one based on pixel intensity alone, which is, then, minimized.

formation process as one step, but rather uses an intermediate level of representation, called the feature level. The purpose of these features is to make the cost function used in the fitting better behaved, i.e. more convex, with fewer local minima.

An interesting problem of pattern analysis would be, given a set of images and an image formation model, to find automatically the intermediate features required to make the fitting of the image model to the input images through these features, a convex optimisation problem. This would require uncovering the features representations and the algorithm able to extract them from input images. This automatic feature selection problem is difficult due to several reasons. First, it is clearly apparent whether a given cost function is convex or not, but when it is non-convex, there is no mathematical tool to analyse how remote it is from a convex function on a given domain. Second, selecting a feature would require a feature space modelling the ensemble of features. The construction of such a feature space, general enough to be able to instantiate various types of features (such as edges, specular highlight or image constraints), has never, to the best of my knowledge, been attempted. Therefore, the selection of these features was based on image cues used in the field computer vision. I show in this work that using this ensemble of features yields a global cost function that is more convex and suffer less from the local minima problem.

## 1.1 Fitting Algorithms Characteristics

A fitting algorithm seeks the parameters of the image model that would produce a facial image as close to the input image as possible. These parameters explain the image and can be used for high-

level tasks such as identification. In order to analyse and compare different fitting algorithms, it is interesting understand their characteristics. Hereafter follows a list of their five major features.

- *Accuracy*: The accuracy, with which the face estimate approximates the ground truth, is crucial for the application that is to use the fitting results (and generally the level of accuracy required depends thereon). Accuracy of the fitting algorithm per se may be measured by several manners. The norm of the residual between the synthesised face obtained from the fitted model parameter and the input image is one choice. Its advantage is its straightforward and automatic computation that make it suitable for comparing fitting results. However, it must be used with care, as its value depends from image to image: A fitted dark image will have a lower residual norm than a brighter image even if its fitting is less accurate. Another disadvantage of this measure is that its value is not intuitive and has limited meaning for a human user. A qualitative measure, meaningful for a human user, is the visual comparison of the input photograph and the synthetic image rendered using the fitted parameters. If the model is a 3D model, then it is more informative to render a synthetic image at another view than the fitted image and to compare it with the same individual photographed from this novel view. This reveals how the fitting algorithm was able to estimate the third dimension. Obviously, this measure is not quantitative and, hence, cannot be used to compare fitting results obtained on an ensemble of face images. Another measure, which might be the one that offers the optimal compromise, is to test the fitting results on a specific quantitative application, such as identification, for instance. The advantage is that it can be easily and automatically recomputed in the same fashion, even across research lab. It's disadvantage is that it does not reflect the accuracy of the fitting algorithm per se, but rather the accuracy of the combination of the fitting and identification algorithms used. Identification experiments obtained with the fitting algorithm developed in this thesis are presented in Section 5.1 on page 67.
- *Efficiency*: The computational load allowed for the fitting algorithm is clearly dependent on applications. Security applications, for instance, require fast algorithms (i.e., near real time).
- *Robustness*: A good fitting algorithm should be robust to non-Gaussian residual. The assumption of normality of the difference between the image synthesised by the model and the input image is generally used for its simple least squares cost function. However, this assumption is frequently violated owing to the presence of accessories or artifacts (glasses, hair, specular highlight).
- *Automatic behaviour*: The fitting should require as little human intervention as possible, optimally with no initialisation.
- *Applicability*: The domain of application of a fitting algorithm is the type of facial images that are suitable to be fitted by an algorithm. There exist view-specific fitting algorithms that can be applied to facial images viewed from a specific direction (such as frontal face images, for instance). There are fitting algorithms able to recover ambient light only and that, generally, fail to fit faces lit by a directional light. There exist, even, person specific algorithms capable of fitting a small number of individuals. The domain of application of a fitting algorithm strongly depends on the type of generative face model it uses. Naturally, an optimal fitting algorithm should be applicable on any facial photographs.

An algorithm capable of any of the five aforementioned features is difficult to set up. An algorithm capable of *all* five features is the holy grail of model-based computer vision. The different methods proposed in the literature for addressing the fitting problem are summarised in Chapter 3. It will be seen that the fitting algorithms proposed so far trade off efficiency for accuracy: Several efficient algorithms have been proposed that works at interactive rates; however what is gained in speed is lost in accuracy and in applicability.

An interesting question facing anyone, having the task to design a fitting algorithm, concerns, then, the feature he should favour. Should he opt for an efficient algorithm at the expense of

accuracy or vice versa? One element that might help his decision is the fact that processor speed increases dramatically with time. Gray [35] shows that, since 1980, the processor speed (measured in MIPS, million instructions per second) has followed the Moore's Law [53] that predicted its double every year. Intel [45] reports that with the advent of nanotechnology, the Moore's Law will be sustained for at least the next decade. So, a fitting algorithm that now runs in 1 minute per face image, may run in less than a second in six years from now (and at 68 Hz in 12 years). On the other hand, what is certain is that the accuracy of a fitting algorithm stays constant over time. Therefore, in the design of the Multi-Features Fitting algorithm presented in this thesis, I favoured accuracy over efficiency.



## Chapter 2

# Image synthesis with the 3D Morphable Model

In this chapter, the 3D Morphable Model construction is reviewed. In the analysis by synthesis framework that we adopted, the synthesis part plays an important role and is therefore detailed in Section 2.2. The image synthesis is the method by which a facial image is produced from a set of model parameters. The model construction and the image synthesis were reported in the thesis of Volker Blanz [10].

### 2.1 3D Morphable Model Construction

The construction of a 3D morphable model requires a set of example 3D faces (e.g., laser scans). The morphable model used for this thesis was constructed with 200 laser scans acquired by a *Cyberware* 3030PS laser scanner. The construction is performed in three steps: First, the laser scans are preprocessed. This semi automatic step aims to remove the scanning artifacts and to select the part of the head that is to be modelled (from one ear to the other and from the neck to the forehead). In the second step, the correspondences are computed between one scan chosen as the reference scan, and each of the other scans. Then a principal components analysis is performed to estimate the statistics of the 3D shape and texture of the faces.

#### 2.1.1 Motivations of the representation

The model should be generic and code all the sources of variations to allow synthesis and analysis of any facial photograph. The coding of these sources of variation separately and independently is important because of two reasons. Firstly, the analysis of a face image yields, then, directly the high-level knowledge desired as the value of the model parameters. If the parameters are independent, they are not affected by one another. For instance, for the task of identification, it is paramount that the identity parameter stay constant, even if the illumination, the pose or the expression of the individual vary on input photographs. Secondly, after the analysis of a facial image, one or several sources of variation may be modified to generate a new facial image that retains certain characteristic of the input photograph, while changing others. For instance, in the case of an expression transfer application, a new facial image is produced with a new expression of the same person in the input photograph. This novel expression could come from another facial photograph of another individual.

A 3D representation enables the accurate modelling of any illumination and pose as well as the separation of these variations from the rest (identity, expression, etc.). The generative model must be able to synthesise images from any individual. In a morphable model, the identity variation is modelled by making linear combinations of faces of a small set of persons. In this chapter, we show

why linear combinations yield a realistic face only if the set of example faces is in correspondence. A good generative model should be restrictive in the sense that unlikely faces should be rarely instantiated. To achieve this the probability density function of human faces is learnt and used as a prior probability to synthesise faces.

### Three-Dimensional Representation

Each individual face can generate a variety of images. This huge diversity of face images makes their analysis difficult. In addition to the general differences between individual faces, the appearance variations in images of a single individual can be separated into the following four sources.

- Pose changes can result in dramatic changes in images. Due to occlusions different parts of the object become visible or invisible. Additionally, the parts seen in two 2D views change their spatial configuration relative to each other.
- Illumination changes influence the appearance of a face even if the pose of the face is fixed. Positions and distribution of light sources around a face have the effect of changing the brightness distribution in the images, the locations of attached shadows and specular reflections. Additionally, cast shadows can generate prominent contours in facial images.
- Facial expressions, an important tool in human communication, are another source of variations in images. Only a few facial landmarks that are directly coupled with the bony structure of the skull, such as the interocular distance or the general position of the ears, are constant in a face. Most other features can change their spatial configuration or position via articulation of the jaw or muscle action (e.g. a moving eyebrows, lips or cheeks).
- In the long term, a face changes because of ageing, changing a hairstyle, or use of makeup or accessories.

The isolation and explicit description of all these different sources of variations must be the ultimate goal of a face analysis system. For example, it is desirable that the parameters that code the identity of a person are not perturbed by a modification of pose. In an analysis by synthesis framework this implies that a face model must account for each of these variations independently by explicit parameters.

The main challenge for the design of such systems is to find or choose a description of these parameters that allows the appropriate modelling of images on the one hand and gives a precise description of an image on the other.

Some of the sources of variations, such as illumination and pose, obey the physical laws of nature. These laws reflect constraints derived from the three-dimensional geometry of faces and the interaction of their surfaces with light. They are optimally imposed by a 3D representation, which was therefore chosen for the morphable model.

On the other hand, there are additional regularities between faces that are not formulated as physical laws but can be obtained by exploiting the general statistics of faces. These methods are also denoted as learning from examples. It is expected that learning schemes that conform or incorporate the physical constraints are more successful in tasks such as generalising from a single image of a face to novel views or to different illumination conditions.

As a result, the 3D morphable model uses physical laws to model pose and illumination as well as statistical methods to model identity and expression. As we see in the next two sections, these statistical methods require the faces to be put into correspondence.

### Vector Space of Faces

In the previous section, we saw that a facial image under any illumination and viewed from any angle can be obtained using a 3D face representation. The model must be able to generate facial image of any individual as well. One way of generating any face is by making a face space that

spans the identity variations and sampling from it. As the face surface is represented as a discrete set of points, the face space is a vector space, and a face must be vectorised. Then, assuming that the dimension of this vector space is finite, a new face can be produced by a linear combination of existing vectorised faces.

**Correspondence.** Then, the question is, how to vectorise a face. As an example, let us consider the case of 2D face images; the same argument is applied to 3D faces. If the raw textures were vectorised simply by concatenating the pixel of a face image, as was done by Turk and Pentland [68], already the mean of two textures would not appear as a face texture. As the edges of these two textures are very unlikely to fall on the same pixels, the result would be blurry and present double edges and such artifacts. Hence, facial images in their pixel representation do not form a vector space. For correct image modelling and synthesis, it is not sufficient to consider only image intensities, it is also necessary to consider the spatial locations of object features. That is, the correspondence between images has to be established. Then, face images must be sampled on a set of feature points common to all faces. This set of points is called the reference frame. Only a separate addition of the shape sampled in the reference frame and texture, also sampled in the common reference frame, satisfies the vector space requirements. Hence, shape alone forms a vector space, and texture alone forms another vector space. The face vector space is the combination of these two vector spaces.

The utilisation of correspondence for image modelling was proposed by several authors [8, 25, 37, 47, 69]; for a review see Beymer and Poggio [7]. The common feature of these methods is that they all derive their facial model, used for the analysis and synthesis of face images, from separate texture and shape vector spaces. It should be noted that some of these approaches do not extract the correspondence of *physical* points: Lanitis *et al.* [47], for instance, put into correspondence the 2D occluding contour, which varies from pose to pose. In contrast, our approach puts in correspondence 3D points that are equivalent across object instances.

The optical flow algorithm used to compute correspondences between faces is reviewed in Section 2.1.3.

**Face Statistics.** In the previous paragraph, a vector space of faces was introduced. To use this space appropriately, an orthogonal basis and a probability distribution over this space must be defined. The probability distribution over the face space is important for the robustness of the fitting algorithm and required to yield a maximum a posterior estimate. Assuming a Gaussian distribution, principal component analysis is used to derive the orthonormal basis of the shape and texture vector space and to estimate the mean and covariance matrix from a set of exemplar faces in Section 2.1.4.

## 2.1.2 Preprocessing Laser Scans

Preprocessing laser scans is a semi automatic step that aims to remove the scanning artifacts and select the part of the head that is to be modelled. It is performed in three consecutive stages.

1. Holes are filled, and spikes (i.e., scan artifacts) are removed using an interactive tool.
2. The faces are rigidly aligned in 3D using the 3D-3D absolute orientation method [40].
3. The parts of the head that we do not wish to model (e.g., the back of the head behind the ears, the hair area, and the region underneath the throat) are trimmed.

## 2.1.3 Dense Correspondences Computed by Optical Flow

To automatically compute dense point-to-point correspondences between two 3D laser scans of faces, an optical flow algorithm is used. The scans are recorded by a laser scanner, measuring

the radius (i.e., depth),  $r$ , along with the colour  $R, G, B$  of faces. Optical flow is computed on a cylindrical representation,  $I(h, \phi)$ , of the coloured 3D scans:

$$I(h, \phi) = [r(h, \phi), R(h, \phi), G(h, \phi), B(h, \phi)]. \quad (2.1)$$

Correspondences are given by a dense vector field  $\mathbf{v}(h, \phi) = [\Delta h(h, \phi), \Delta \phi(h, \phi)]$ , such that each point of the first scan,  $I_1(h, \phi)$ , corresponds to the point  $I_2(h + \Delta h, \phi + \Delta \phi)$  on the second scan. A modified optical flow algorithm [12] is used to estimate this vector field.

**Optical flow on grey-level images** Many optical flow algorithms (e.g. [6], [43], [49]) are based on the assumption that objects in an image sequence  $I(x, y, t)$  conserve their brightness as they move across the images at a velocity  $(v_x, v_y)^T$ .

$$\frac{dI}{dt} = v_x \frac{\partial I}{\partial x} + v_y \frac{\partial I}{\partial y} + \frac{\partial I}{\partial t} = 0 \quad (2.2)$$

For a pair of images,  $I_1$  and  $I_2$ , taken at two discrete moments, the temporal derivatives,  $v_x, v_y$  and  $\frac{\partial I}{\partial t}$ , in Equation (2.2), are approximated by finite differences  $\Delta x, \Delta y$ , and  $\Delta I = I_2 - I_1$ . If the images are not from a temporal sequence but show two different objects, corresponding points can no longer be assumed to have equal brightnesses. Still, optical flow algorithms may be applied successfully.

A unique solution for both components of  $\mathbf{v} = (v_x, v_y)^T$  from Equation (2.2) can be obtained if  $\mathbf{v}$  is assumed to be constant on each neighbourhood  $R(x_0, y_0)$ , and the following expression [6, 49] is minimised at each point  $(x_0, y_0)$ .

$$E(x_0, y_0) = \sum_{x, y \in R(x_0, y_0)} \left[ v_x \frac{\partial I(x, y)}{\partial x} + v_y \frac{\partial I(x, y)}{\partial y} + \Delta I(x, y) \right]^2 \quad (2.3)$$

A  $5 \times 5$  pixel neighbourhood  $R(x_0, y_0)$  is used. In each point  $(x_0, y_0)$ ,  $\mathbf{v}(x_0, y_0)$  can be found by solving a  $2 \times 2$  linear system (see Blanz and Vetter [13] for details). To deal with large displacements  $\mathbf{v}$ , the algorithm of Bergen and Hingorani [6] employs a coarse-to-fine strategy using a Gaussian pyramid of down-sampled images: With the gradient-based method described above, the algorithm computes the flow field on the lowest level of resolution and refines it on each subsequent level.

**Generalisation to 3D coloured surfaces** For processing 3D laser scans  $\mathbf{I}(h, \phi)$ , Equation (2.3) is replaced by

$$E = \sum_{h, \phi \in R} \left\| v_h \frac{\partial \mathbf{I}(h, \phi)}{\partial h} + v_\phi \frac{\partial \mathbf{I}(h, \phi)}{\partial \phi} + \Delta \mathbf{I}(h, \phi) \right\|^2 \quad (2.4)$$

$$\text{with a norm } \|\mathbf{I}(h, \phi)\|^2 = w_r r^2 + w_R R^2 + w_G G^2 + w_B B^2 \quad (2.5)$$

Weights  $w_r, w_R, w_G$ , and  $w_B$  compensate for variations in the radius data and the red, green, and blue texture components; and they control the overall weighting of shape versus texture information. The weights are chosen heuristically. The minimum of Equation (2.4) is again given by a  $2 \times 2$  linear system (see [13]).

Correspondences between scans of different individuals, who may differ in overall brightness and size, are improved by using Laplacian pyramids (band-pass filtering) rather than Gaussian pyramids (low-pass filtering). Additional quantities, such as Gaussian curvature, mean curvature, or the surface normal, may be incorporated in  $\mathbf{I}(h, \phi)$  to improve the results. To obtain reliable results even in regions of the face with no salient structures, a specifically designed smoothing and interpolation algorithm [13] is added to the matching procedure on each level of resolution.

### 2.1.4 Face Space Based on Principal Components Analysis

We mentioned that the correspondences enable the formulation of a face space. The face space is constructed by putting a set of  $M$  example 3D laser scans into correspondence with a reference laser scan. This introduces a consistent labelling of all  $N_v$  3D vertices across all the scans. The shape and texture surfaces are parameterised in the  $(u, v)$  reference frame, where one pixel corresponds to one 3D vertex (Figure 2.1). The 3D position in Cartesian coordinates of the  $N_v$  vertices of a face scan are arranged in a shape matrix,  $\mathbf{S}$ ; and their colour in a texture matrix,  $\mathbf{T}$ .

$$\mathbf{S} = \begin{pmatrix} x_1 & x_2 & \cdots & x_{N_v} \\ y_1 & y_2 & \cdots & y_{N_v} \\ z_1 & z_2 & \cdots & z_{N_v} \end{pmatrix}, \quad \mathbf{T} = \begin{pmatrix} r_1 & r_2 & \cdots & r_{N_v} \\ g_1 & g_2 & \cdots & g_{N_v} \\ b_1 & b_2 & \cdots & b_{N_v} \end{pmatrix} \quad (2.6)$$

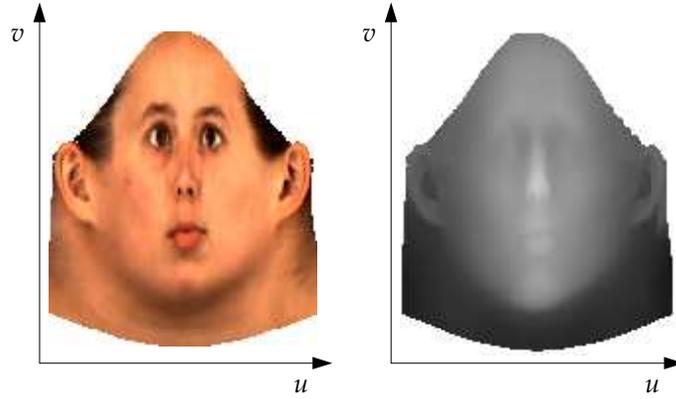


Figure 2.1: Texture and shape in the reference space  $(u, v)$ .

Having constructed a linear face space, we can make linear combinations of the shapes,  $\mathbf{S}_i$ , and the textures,  $\mathbf{T}_i$  of  $M$  example individuals to produce faces of new individuals.

$$\mathbf{S} = \sum_{i=1}^M \alpha_i \cdot \mathbf{S}_i, \quad \mathbf{T} = \sum_{i=1}^M \beta_i \cdot \mathbf{T}_i \quad (2.7)$$

Equation (2.7) assumes a uniform distribution of the shapes and the textures. We know that this distribution yields a model that is not restrictive enough: For instance, if some  $\alpha_i$  or  $\beta_i$  are  $\gg 1$ , the face produced is unlikely. Therefore, we assume that the shape and the texture spaces have a Gaussian probability distribution function. Principal component analysis (PCA) is a statistical tool that transforms the space such that the covariance matrix is diagonal (i.e., it decorrelates the data). PCA is applied separately to the shape and texture spaces. We describe the application of PCA to shapes; its application to textures is straightforward. After subtracting their average,  $\bar{\mathbf{S}}$ , the exemplars are arranged in a data matrix  $\mathbf{A}$  and the eigenvectors of its covariance matrix  $\mathbf{C}$  are computed using the singular value decomposition [58] of  $\mathbf{A}$ .

$$\bar{\mathbf{S}} = \frac{1}{M} \sum_{i=1}^M \mathbf{S}_i, \quad \mathbf{a}_i = \text{vec}(\mathbf{S}_i - \bar{\mathbf{S}}), \quad \mathbf{A} = [\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_M] = \mathbf{U}\mathbf{\Lambda}\mathbf{V}^T \quad (2.8)$$

$$\mathbf{C}_A = \frac{1}{M} \mathbf{A}\mathbf{A}^T = \frac{1}{M} \mathbf{U}\mathbf{\Lambda}^2\mathbf{U}^T$$

The component  $\text{vec}(\mathbf{S})$  vectorised  $\mathbf{S}$  by stacking its columns. The  $M$  columns of the orthogonal matrix  $\mathbf{U}$  are the eigenvectors of the covariance matrix  $\mathbf{C}_A$ , and  $\sigma_i^2 = \frac{\lambda_i^2}{M}$  are its eigenvalues, where the  $\lambda_i$  are the elements of the diagonal matrix  $\mathbf{\Lambda}$ , arranged in decreasing order. Now, instead of representing the data matrix  $\mathbf{A}$  in its original space, it can be projected to the space spanned by

the eigenvectors of its covariance matrix. Let us denote by the matrix  $\mathbf{B}$  this new representation of the data, and by  $\mathbf{C}_B$  its covariance matrix.

$$\mathbf{B} = \mathbf{U}^T \mathbf{A} \quad \mathbf{C}_B = \frac{1}{M} \mathbf{B} \mathbf{B}^T = \frac{1}{M} \mathbf{\Lambda}^2 \quad (2.9)$$

The second equality of the last equation is obtained because  $\mathbf{U}^T \mathbf{U} = \mathbf{I}$  and  $\mathbf{V}^T \mathbf{V} = \mathbf{I}$ , where  $\mathbf{I}$  is the identity matrix with the appropriate number of elements. Hence the projected data are decorrelated, as they have a diagonal covariance matrix. Hereafter, we denote by  $\sigma_{S,i}$  and  $\sigma_{T,i}$  the variances of, respectively, the shape and the texture vectors.

There is a second advantage in expressing a shape (or texture) as a linear combination of shape principal components, namely dimensionality reduction. It is demonstrated in [28], that the subspace, spanned by the columns of an orthonormal matrix  $\mathbf{X}$ , with  $N$  dimensions that minimises the mean squared difference between a data vector  $\mathbf{a}$ , sampled from the same population as the column vectors of the matrix  $\mathbf{A}$ , and its reconstruction,  $\mathbf{X} \mathbf{X}^T \mathbf{a}$ , is the one formed by the  $N$  eigenvectors having largest eigenvalues:  $\mathbf{X} = [\mathbf{U}_{\cdot,1}, \dots, \mathbf{U}_{\cdot,N}]$ , where  $\mathbf{U}_{\cdot,i}$  denotes the  $i^{\text{th}}$  column of  $\mathbf{U}$ . On average the decrease of the mean squared difference follow the curve of the variance that is plotted on Figure 2.2. It can be seen that for the 50 the decrease is polynomial and for higher principal components it is exponential.

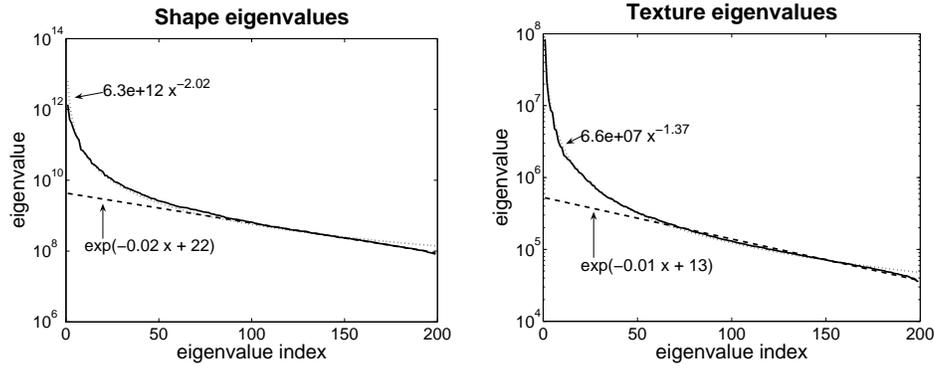


Figure 2.2: Semi logarithmic plot of the shape and texture eigenvalues. For each plot a polynomial (dashed) line and an exponential (dotted) line are fitted and their equation printed.

Let us denote  $\mathbf{U}_{\cdot,i}$ , the column  $i$  of  $\mathbf{U}$ , and the principal component  $i$ , reshaped into a  $3 \times N_v$  matrix, by  $\mathbf{S}^i = \mathbf{U}_{\cdot,i}^{(3)}$ . The notation  $\mathbf{a}_{m \times 1}^{(n)}$  [52] folds the  $m \times 1$  vector  $\mathbf{a}$  into an  $n \times (m/n)$  matrix.

Now, instead of describing a novel shape and texture as a linear combination of examples, as in Equation 2.7, we express them as a linear combination of  $N_S$  shape and  $N_T$  texture principal components.

$$\mathbf{S} = \bar{\mathbf{S}} + \sum_{i=1}^{N_S} \alpha_i \cdot \mathbf{S}^i, \quad \mathbf{T} = \bar{\mathbf{T}} + \sum_{i=1}^{N_T} \beta_i \cdot \mathbf{T}^i \quad (2.10)$$

The third advantage of this formulation is that the probabilities of a shape and a texture are readily available from their parameters.

$$p(\mathbf{S}) \sim e^{-\frac{1}{2} \sum_i \frac{\alpha_i^2}{\sigma_{S,i}^2}}, \quad p(\mathbf{T}) \sim e^{-\frac{1}{2} \sum_i \frac{\beta_i^2}{\sigma_{T,i}^2}} \quad (2.11)$$

### 2.1.5 Segmented Morphable Model

As mentioned, our morphable model is derived from statistics computed on 200 example faces. As a result, the dimensions of the shape and texture spaces,  $N_S$  and  $N_T$ , are limited to 199. This might not be enough to account for the rich variations of individualities present in mankind. Naturally,

one way to augment the dimension of the face space would be to use 3D scans of more persons but they are not available. Hence we resort to another scheme: We segment the face into four regions (nose, eyes, mouth and the rest) shown on Figure 2.3 and use a separate set of shape and texture coefficients to code them [12]. This method multiplies by four the expressiveness of the morphable model. The results presented in the Chapter 5 are based on a segmented morphable model with  $N_S = N_T = 100$  for all segments. In this thesis, we denote the shape and texture parameters by  $\alpha$  and  $\beta$  when they can be used interchangeably for the global and the segmented parts of the model. When we want to distinguish them, we use, for the shape parameters,  $\alpha^s$  for the global model (full face) and  $\alpha^{s_1}$  to  $\alpha^{s_4}$  for the segmented parts (the same notation is used for the texture parameters).

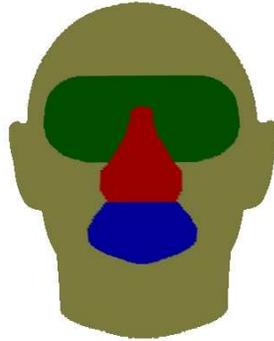


Figure 2.3: Image showing the four segments of the segmented morphable model: the nose, the eyes, the mouth and the rest.

## 2.2 Morphable Model to Synthesise Images

One part of the analysis by synthesis loop is the synthesis (i.e., the generation of accurate face images viewed from any pose and illuminated by any condition). This process is explained in this section.

### 2.2.1 Shape Projection

To render the image of a face, the 3D shape must be projected to the 2D image frame. This is performed in two steps. First, a 3D rotation and translation (i.e. a rigid transformation) maps the object-centred coordinates,  $\mathbf{S}$ , to a position relative to the camera in world coordinates.

$$\mathbf{W} = \mathbf{R}_\gamma \mathbf{R}_\theta \mathbf{R}_\phi \mathbf{S} + \mathbf{t}_w \mathbf{1}_{1 \times N_v} \quad (2.12)$$

The angles  $\phi$  and  $\theta$  control in-depth rotations around the vertical and horizontal axis, and  $\gamma$  defines a rotation around the camera axis;  $\mathbf{t}_w$  is a 3D translation. A projection then maps a vertex  $i$  to the image plane in  $(x_i, y_i)$ . We typically use two projections: the perspective and the weak perspective projection.

$$\text{Perspective : } \begin{cases} x_i = t_x - f \frac{\mathbf{W}_{1,i}}{\mathbf{W}_{3,i}} \\ y_i = t_y + f \frac{\mathbf{W}_{2,i}}{\mathbf{W}_{3,i}} \end{cases} \quad \text{Weak perspective : } \begin{cases} x_i = t_x - f \mathbf{W}_{1,i} \\ y_i = t_y + f \mathbf{W}_{2,i} \end{cases} \quad (2.13)$$

where  $f$  is the focal length of the camera, which is located in the origin; and  $(t_x, t_y)$  defines the image-plane position of the optical axis. The negative sign in front of the X component of the world coordinate exists for legacy reasons.

For ease of explanation, the shape transformation parameters are denoted by the vector  $\rho = [f \ \phi \ \theta \ \gamma \ t_x \ t_y \ \mathbf{t}_w^T]^T$ , and  $\alpha$  is the vector whose elements are the  $\alpha_i$ . The projection of the vertex  $i$  to the image frame  $(x, y)$  is denoted by the vector valued function  $\mathbf{p}(u_i, v_i; \alpha, \rho)$ . This function is clearly continuous in  $\alpha, \rho$ . To provide continuity in the  $(u, v)$  space as well, we use a triangle list and interpolate between neighbouring vertices, as is common in computer graphics. Note that only  $N_{vv}$  vertices, a subset of the  $N_v$  vertices, are visible after the 2D projection (the remaining vertices are hidden by self-occlusion). We call this subset the domain of the shape projection  $\mathbf{p}(u_i, v_i; \alpha, \rho)$  and denote it by  $\Omega(\alpha, \rho) \in (u, v)$ .

In conclusion, the shape modelling and its projection provides mapping from the parameter space  $\alpha, \rho$  to the image frame  $(x, y)$  via the reference frame  $(u, v)$ . However to synthesise an image, we need the inverse of this mapping, detailed in the next section.

## 2.2.2 Inverse Shape Projection

The shape projection aforementioned maps a  $(u, v)$  point from the reference space to the image frame. To synthesise an image, we need the inverse mapping: An image is generated by looping on the pixels  $(x, y)$ . To know which colour must be drawn on that pixel, we must know where this pixel is mapped into the reference frame. This is the aim of the inverse shape mapping explained in this section.

The inverse shape projection,  $\mathbf{p}^{-1}(x, y; \alpha, \rho)$ , maps an image point  $(x, y)$  to the reference frame  $(u, v)$ . Let us denote the composition of a shape projection and its inverse by the symbol  $\circ$ ; hence,  $\mathbf{p}(u, v; \alpha, \rho) \circ \mathbf{p}^{-1}(x, y; \alpha, \rho)$  is equal to  $\mathbf{p}(\mathbf{p}^{-1}(x, y; \alpha, \rho); \alpha, \rho)$ , but we prefer the former notation for clarity. The inverse shape projection is defined by the following equation, which specifies that under the same set of parameters the shape projection composed with its inverse is equal to the identity.

$$\begin{aligned} \mathbf{p}(u, v; \alpha, \rho) \circ \mathbf{p}^{-1}(x, y; \alpha, \rho) &= (x, y) \\ \mathbf{p}^{-1}(x, y; \alpha, \rho) \circ \mathbf{p}(u, v; \alpha, \rho) &= (u, v) \end{aligned} \quad (2.14)$$

Because the shape is discrete, it is not easy to express  $\mathbf{p}^{-1}$  analytically as a function of  $\mathbf{p}$ , but it can be computed using the triangle list: The domain of the plane  $(x, y)$  for which there exists an inverse under the parameters  $\alpha$  and  $\rho$ , denoted by  $\Psi(\alpha, \rho)$ , is the range of  $\mathbf{p}(u, v; \alpha, \rho)$ . Such a point of  $(x, y)$  lies in a single visible triangle under the projection  $\mathbf{p}(u, v; \alpha, \rho)$ . Therefore, the point in  $(u, v)$  under the inverse projection has the same relative position in this triangle in the  $(u, v)$  space. This process is depicted in Figure 2.4.

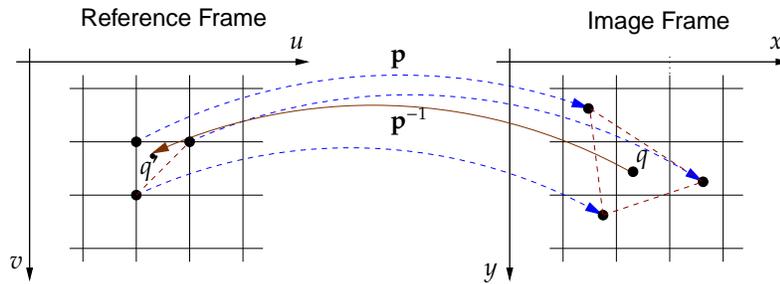


Figure 2.4: Inverse shape function  $\mathbf{p}^{-1}(x, y; \alpha, \rho)$  maps the point  $q$  (defined in the  $(x, y)$  coordinate system), onto the point  $q'$  in  $(u, v)$ . This is done by recovering the triangle that would contain the pixel  $q$  under the mapping  $\mathbf{p}(u, v; \alpha, \rho)$ . Then the relative position of  $q$  in that triangle is the same as the relative position of  $q'$  in the same triangle in the  $(u, v)$  space.

### 2.2.3 Illumination and Colour Transformation

**Ambient and Directed Light** We simulate the illumination of a face using an ambient light and a directed light. The effects of the illumination are obtained using the standard Phong model, which approximately describes the diffuse and specular reflection on a surface [31]; see [13] for further details. The parameters of this model are the intensity of the ambient light ( $L_r^a, L_g^a, L_b^a$ ), the intensity of the directed light ( $L_r^d, L_g^d, L_b^d$ ), its direction ( $\theta_l$  and  $\phi_l$ ), the specular reflectance of human skin ( $k_s$ ), and the angular distribution of the specular reflections of human skin ( $\nu$ ). For clarity, we denote by the vector  $\mathbf{t}_i$  the  $i^{\text{th}}$  column of the matrix  $\mathbf{T}$ , representing the RGB colour of the vertex  $i$ . When illuminated, the colour of this vertex is transformed to  $\mathbf{t}_i^l$ .

$$\mathbf{t}_i^l = \begin{pmatrix} L_r^a & 0 & 0 \\ 0 & L_g^a & 0 \\ 0 & 0 & L_b^a \end{pmatrix} \cdot \mathbf{t}_i + \begin{pmatrix} L_r^d & 0 & 0 \\ 0 & L_g^d & 0 \\ 0 & 0 & L_b^d \end{pmatrix} \cdot (\langle \mathbf{n}_i^{\nu,w}, \mathbf{d} \rangle \cdot \mathbf{t}_i + k_s \cdot \langle \mathbf{r}_i, \mathbf{v}_i \rangle^\nu \cdot \mathbf{1}_{3 \times 1}) \quad (2.15)$$

The first term of this equation is the contribution of the ambient light. The first term of the last parenthesis is the diffuse component of the directed light and the second term is its specular component. To take account of the attached shadows, these two scalar products are lower bounded to zero. To take account of the cast shadows, a shadow map is computed, using standard computer graphics techniques [31]. The vertices in the cast shadow are illuminated by the ambient light only.

In Equation 2.15,  $\mathbf{d}$  is the light direction in Cartesian coordinates, which can be computed from its spherical coordinates by:

$$\mathbf{d} = \begin{pmatrix} \cos(\theta_l) \cdot \sin(\phi_l) \\ \sin(\theta_l) \\ \cos(\theta_l) \cdot \cos(\phi_l) \end{pmatrix} \quad (2.16)$$

The normal of the vertex  $i$ ,  $\mathbf{n}_i^{\nu,w}$ , is expressed in world coordinates. World coordinates of a normal are obtained by rotating the normal from the object centred coordinates to the world coordinates.

$$\mathbf{n}_i^{\nu,w} = \mathbf{R}_\gamma \mathbf{R}_\theta \mathbf{R}_\phi \mathbf{n}_i^\nu \quad (2.17)$$

The normal of a vertex in object centred coordinates,  $\mathbf{n}_i^\nu$ , is defined as the unit-length mean of the normals of the triangles connected to this vertex (i.e. the triangles for which this vertex is one of the three corners).

$$\mathbf{n}_i^\nu = \frac{\sum_{j \in \mathcal{T}_i} \mathbf{n}_j^t}{\|\sum_{j \in \mathcal{T}_i} \mathbf{n}_j^t\|} \quad (2.18)$$

where  $\mathcal{T}_i$  is the set of triangle indexes connected to the vertex  $i$ . The normal of a triangle is determined by the unit-length cross product of the vectors formed by two of its edges. If  $\mathbf{s}_{i_1}$ ,  $\mathbf{s}_{i_2}$ , and  $\mathbf{s}_{i_3}$  are the Cartesian coordinates of the three corners of the triangle  $j$  (these indexes are given by the triangle list), then its normal is:

$$\mathbf{n}_j^t = \frac{(\mathbf{s}_{i_1} - \mathbf{s}_{i_2}) \times (\mathbf{s}_{i_1} - \mathbf{s}_{i_3})}{\|(\mathbf{s}_{i_1} - \mathbf{s}_{i_2}) \times (\mathbf{s}_{i_1} - \mathbf{s}_{i_3})\|} \quad (2.19)$$

In the Equation (2.15),  $\mathbf{v}_i$  is the viewing direction of the vertex  $i$ , which is the direction between the vertex  $i$  and the camera centre. The camera centre is at the origin of the world coordinates.

$$\mathbf{v}_i = -\frac{\mathbf{W}_{:,i}}{\|\mathbf{W}_{:,i}\|} \quad (2.20)$$

The vector  $\mathbf{r}_i$  in Equation (2.15) is the direction of the reflection of the light coming from the direction  $\mathbf{d}$ . As the graph in Figure 4.14 on page 52 shows, it is computed as follows:

$$\mathbf{r}_i = 2 \cdot \langle \mathbf{n}_i^{\nu,w}, \mathbf{d} \rangle \mathbf{n}_i^{\nu,w} - \mathbf{d} \quad (2.21)$$

**Colour Transformation** Input images may vary a lot with respect to the overall tone of colour. To be able to handle a variety of colour images as well as grey level images and even paintings, we apply gains  $g_r, g_g, g_b$ , offsets  $o_r, o_g, o_b$ , and a colour contrast  $c$  to each channel [12]. This is a linear transformation that yields the definitive colour of a vertex, denoted by  $\mathbf{t}_i^C$ , by multiplying the RGB colour of a vertex after it has been illuminated,  $\mathbf{t}_i^I$ , by the matrix  $\mathbf{M}$  and adds the vector  $\mathbf{o} = [o_r, o_g, o_b]^T$ .

$$\mathbf{t}_i^C = \mathbf{M} \cdot \mathbf{t}_i^I + \mathbf{o}, \text{ where} \quad (2.22)$$

$$\mathbf{M} = \begin{pmatrix} g_r & 0 & 0 \\ 0 & g_g & 0 \\ 0 & 0 & g_b \end{pmatrix} \cdot \left[ \mathbf{I} + (1 - c) \begin{pmatrix} 0.3 & 0.59 & 0.11 \\ 0.3 & 0.59 & 0.11 \\ 0.3 & 0.59 & 0.11 \end{pmatrix} \right] \quad (2.23)$$

For brevity, the illumination and colour transformation parameters are regrouped in the vector  $\boldsymbol{\iota}$ . Hence the illuminated and colour corrected texture depends on the coefficients of the texture linear combination regrouped in  $\boldsymbol{\beta}$ , on the light parameters  $\boldsymbol{\iota}$ , and on  $\boldsymbol{\alpha}$  and  $\boldsymbol{\rho}$  used to compute the normals and the viewing direction of the vertices required for the Phong illumination model.

Similar to the shape, the colour of a vertex  $i$ ,  $\mathbf{t}_i^C$ , is represented on the  $(u, v)$  reference frame by the vector valued function  $\mathbf{t}^C(u_i, v_i; \boldsymbol{\beta}, \boldsymbol{\iota}, \boldsymbol{\alpha}, \boldsymbol{\rho})$ , which is extended to the continuous function  $\mathbf{t}^C(u, v; \boldsymbol{\beta}, \boldsymbol{\iota}, \boldsymbol{\alpha}, \boldsymbol{\rho})$  by using the triangle list and interpolating.

## 2.2.4 Image Synthesis

Synthesising the image of a face is performed by mapping a texture from the reference to the image frame using an inverse shape projection.

$$I^m(x_j, y_j; \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\rho}, \boldsymbol{\iota}) = \mathbf{t}^C(u, v; \boldsymbol{\beta}, \boldsymbol{\iota}, \boldsymbol{\alpha}, \boldsymbol{\rho}) \circ \mathbf{p}^{-1}(x_j, y_j; \boldsymbol{\alpha}, \boldsymbol{\rho}) \quad (2.24)$$

where  $j$  runs over the pixels that belong to  $\Psi(\boldsymbol{\alpha}, \boldsymbol{\rho})$  (i.e., the pixels for which a shape inverse exist, as defined in Section 2.2.2).

## Chapter 3

# Compendium of Face Image Analysis Techniques

The previous chapter detailed the 3D Morphable Model, a mathematical formulation of the full image formation process. It takes into account most of the sources of facial image variations (flexible shape deformation, varying albedo, 3D rotation, and directed lights). A face recognition algorithm is intrinsically a method that is inverting this image formation process, i.e. inverting Equation (2.24), thereby separating the identity from the imaging parameters. The first algorithm that inverted the full image formation process, and that made the least assumptions, treating the problem in its full complexity, is the Stochastic Newton Optimisation (SNO) [10, 12, 13]. It casted the task as an optimisation problem estimating all the model parameters (shape, texture, rigid transformation and illumination). The only assumption made is that the pixels are independent and identically distributed with a residual normally distributed. This is a difficult and computationally expensive optimisation problem, as the cost function is non convex. To avoid the local minima problem, the optimisation was made stochastic: At each iteration of the fitting algorithm, the cost function and its derivatives are evaluated on a very small set of points (40) that are randomly chosen. This stochasticity introduces a perturbation on the derivatives that minimises the risk of locking on a local minimum. The price to pay is a computationally expensive algorithm with several thousands iterations.

Over the past years, many face recognition methods have been presented. Having computational efficiency and tractability as main goal, these methods restrain the domain of applicability and make several assumptions. It is interesting to analyse the assumptions made by each of these methods and their limitations in light of the 3DMM.

**Eigenfaces** Since the beginning of the nineties, statistical method for face recognition are extensively used. It started with Kirby and Sirovich [46] and Turk and Pentland [68], who applied a principal component analysis (PCA) on facial images. With their technique, a facial image was coded with a low number of coefficients by projecting it to the orthogonal subspace obtained by PCA. These coefficients could then be used for identification. This method was also applicable to video compression, as one face image, encoded in the low-dimensional principal component subspace, could be reconstructed to the original image space. The problem was that this image was blurry for reasons explained in Section 2.1.1. In this method, the different sources of variations of faces are not separated: they are all modelled by one set of parameters that also encodes identity. As a result, it suffers from a limited generalisation capabilities across viewpoints and light directions and limited identification performances.

### 3.1 Correspondence-based methods

**Point Distribution Model and Active Shape Model** To remove the blurriness artifact of the Eigenface approach, Craw and Cameron [25] aligned the face images and segmented the face part from the background image. Then they applied PCA on the textures put in coarse correspondence by sampling them on a reference frame defined by a few landmarks. Cootes *et al.* [19, 23] represented, then, the 2D face shapes as a sparse set of feature points that corresponded to one another across a face ensemble. Applying PCA on this set of points yielded a Point Distribution Model (PDM).

Though it can be argued that there is no major conceptual difference between the PDM's of the early nineties and the 3D shape model of the 3DMM (detailed in Chapter 2), as both models compute statistics on a set of points in correspondence, yet there are three major differences between the two models: (i) The use of 3D rather than 2D. (ii) The 3DMM is a dense model including several thousands vertices whereas the PDM uses a few tens. (iii) As implemented by Cootes *et al.*, the PDM includes landmarks on the contour between the face and the background. This contour depends on the pose of the face on the image and on the 3D shape of the individual. Thus the landmarks on it do not represent the same physical points, and hence should not be put in correspondence across individuals. The authors of the PDM argue that it is capable of handling  $\pm 20^\circ$  out of the image plane rotation. However, the 2D shape variation induced by this 3D rigid transformation is encoded in the 2D shape parameters, resulting in shape parameters not independent of the face pose. This reduces the identification capabilities of this model.

The algorithm used to fit a PDM to an image is called the Active Shape Model (ASM) and its first version appeared in Cootes and Taylor [21]. This version used the edge evidence of the input image to adjust the 2D translation, 2D scale, image plane rotation and shape parameters of the model. Each iteration of this fitting algorithm included two steps: First, each model point was displaced in the direction normal to the shape contour, toward the strongest edge, and with an amplitude proportional to the edge strength at the point. This yielded a position for each model point that would fit better the image. The second step was to update the 2D pose and shape parameters reflecting the new position of the model points. To increase the likeliness of the resulting shape, hard limits were put on the shape parameters. This method proved itself not robust enough to deal with complex objects, where the model points do not necessarily lie on strong edges. Therefore, the second version of the ASM [22] modelled the grey-levels along the normal of the contour at each model point. Then, during model search, a better position for a model point was given by the image point, along the normal of the contour, that minimised the distance to its local grey-level model. An exhaustive search for this minimiser was performed that evaluated all the points along the normal within a given distance of the model point. A PCA model was used to model a grey-level profile along the normal of a contour point. Then the distance minimised during fitting is the norm of the reconstruction error of a grey-level profile obtained from a point along the normal evaluated as a potential minimiser. A predicament of the local grey-level models, as they were implemented in the ASM, is the fact that one half of the pixels modelled are outside the face area, in the background area, and hence could change randomly from image to image.

**Active Appearance Model** The first identification experiment on facial images fitted by an ASM was made by Lanitis *et al.* [48]. After fitting an image by the ASM, the grey texture enclosed within the face contour was sampled on the reference frame defined by the mean shape (and called shape-free texture) and modelled by a PCA model. The features used for identification were the shape and texture coefficients of the shape and texture models.

Continuing in this direction, researchers started to use not only the pixels along the normal of the landmark points but rather the full texture in the face area to drive the parameter fitting algorithm. The main motivation was that, as the algorithm would use more information, the fitting would converge faster to a more accurate minimum more robustly. First, Gleicher [34], with the Image Difference Decomposition (IDD) algorithm, then Sclaroff and Isidoro [65], with

the Active Blobs, and Cootes *et al.* [20], with the Active Appearance Model (AAM), used the full texture error to compute, linearly, an update of the model parameters. The texture error,  $\delta \mathbf{t}$ , is the difference between the texture extracted from the input image and sampled using the shape parameters and the model texture.

$$\delta \mathbf{t} = I(x, y) \circ \mathbf{p}(u, v; \boldsymbol{\alpha}) - \mathbf{t}(u, v; \boldsymbol{\beta}) \quad (3.1)$$

The aim of the fitting algorithm is to estimate the shape and texture model parameters that minimise the square of the norm of the texture error.

$$\min_{\boldsymbol{\alpha}, \boldsymbol{\beta}} \|\delta \mathbf{t}\|^2 \quad (3.2)$$

For efficiency reasons, the texture difference is projected onto a constant matrix, which yields the shape and texture model parameters update.

$$\begin{pmatrix} \delta \boldsymbol{\alpha} \\ \delta \boldsymbol{\beta} \end{pmatrix} = \mathbf{A} \cdot \delta \mathbf{t} \quad (3.3)$$

Then the next estimate is obtained by adding the update to the current estimate.

$$\boldsymbol{\alpha} \leftarrow \boldsymbol{\alpha} + \delta \boldsymbol{\alpha} \quad \text{and} \quad \boldsymbol{\beta} \leftarrow \boldsymbol{\beta} + \delta \boldsymbol{\beta} \quad (3.4)$$

This algorithm would be a gradient descent optimisation algorithm, if the matrix relating the texture error to the model update, the matrix  $\mathbf{A}$ , was the inverse of the Jacobi matrix. Assuming  $\mathbf{A}$  to be constant, is equivalent to assuming that the model Jacobi matrix is constant and hence that the rendering model is linear. We have seen that the sources of nonlinearities of the rendering model are multiple: (i) The out of the image plane rotation and the perspective effects induce a nonlinear variation of the shape points projected onto the image plane. (ii) The modification of the light source direction produce a nonlinear variation in pixel intensities. (iii) The warping of the texture using the shape parameters is nonlinear as well. As the face model on which this fitting algorithm is applied, is 2D, allow only small out of the image plane rotation, and does not model directed light sources, the first two sources of nonlinearities could be limited. Thus, the authors showed that this fitting was effective on facial images with no directed light and with small pose variations. However, it does not produce satisfactory results on the full 3D problem addressed in this thesis.

The constant matrix relating the texture error to the model update, the matrix  $\mathbf{A}$ , was first computed by a regression using texture errors generated from training images and random model parameters displacement. Then, it was estimated by averaging Jacobian matrix obtained by numerical differentiation on typical facial images. A problem with these two approaches is that the pixels outside the face area are sampled to form the training texture error. Thus, the quality of the estimate obtained on an input image depends on the resemblance of the background of this image to the one of the images used for computing the matrix  $\mathbf{A}$ .

To enlarge the domain of application of the Active Appearance Model to faces viewed from any azimuth angle, Cootes *et al.* [24] introduced the multi-view AAM. It is constituted of five AAMs, each trained on facial images at different pose (front, left and right side views and left and right profile). So, the fitting was also constituted of five constant Jacobi matrices. This is an ad-hoc solution addressing one of the limitations of a 2D model that was not pursued afterwards.

**Inverse Compositional Image Alignment algorithm** As aforementioned, for efficiency reasons, the AAM treats the matrix relating the texture error to the model parameter update as constant. This is based on the assumption that the Jacobi matrix (that should be recomputed at each iteration) is well approximated by a constant matrix. However, this matrix is not constant owing to the warping of the texture by the shape. Baker and Matthews [1] introduced the Inverse Compositional Image Alignment (ICIA) algorithm, that also uses a constant Jacobi matrix, but the matrix is shown, to a first order, to be constant. The fixedness of the updating matrix is not assumed anymore.

This was achieved by a modification of the cost function to minimise. Instead of optimising Equation (3.2), the following cost function is minimised.

$$\min_{\delta\alpha} \|\mathbf{t}(u, v; 0) \circ \mathbf{p}(u, v; \delta\alpha) - I(x, y) \circ \mathbf{p}(u, v; \alpha)\|^2 \quad (3.5)$$

To clarify the notations, in what follows, the dependency on the frame coordinates  $(x, y)$  and  $(u, v)$  is not explicit anymore; only the dependency on the model parameters is left. A first Taylor expansion of the term inside the norm of the cost function yields:

$$\min_{\delta\alpha} \|\mathbf{t}(0) \circ \mathbf{p}(0) + \mathbf{t}(0) \left. \frac{\partial \mathbf{p}}{\partial \alpha} \right|_{\alpha=0} \delta\alpha - I \circ \mathbf{p}(\alpha)\|^2 \quad (3.6)$$

Differentiating this cost function with respect to the shape parameter update, equating to zero, and rearranging the terms, yields the expression of the parameter update:

$$\delta\alpha = \left[ \mathbf{t}(0) \left. \frac{\partial \mathbf{p}}{\partial \alpha} \right|_{\alpha=0} \right]^\dagger (I \circ \mathbf{p}(\alpha) - \mathbf{t}(0) \circ \mathbf{p}(0)) \quad (3.7)$$

where the notation  $^\dagger$  denotes the pseudo-inverse of a matrix. This derivation leads to a Gauss-Newton optimisation [33]. In a least squares optimisation, the Hessian is a sum of two terms: the Jacobi matrix transposed and multiplied by itself and the purely second derivative terms. In a Gauss-Newton optimisation, the Hessian is approximated by its first term only. The reason is that in a least square optimisation (such as  $\min_x \sum_i e_i^2$ ), the second derivative term of the Hessian of the cost function is the sum of the Hessian matrices of the elements,  $\frac{\partial^2 e_i}{\partial x^2}$ , multiplied by their residual,  $e_i$ . When the residuals  $e_i$  are small, the approximation is adequate. Hence, a Gauss-Newton optimisation algorithm is more efficient than a Newton algorithm, as the second derivatives are not computed. It is not surprising that the ICIA algorithm is equivalent to a Gauss-Newton optimisation, as it was derived using a first order Taylor approximation.

The essence of the ICIA algorithm, is that the Jacobi matrix,  $\mathbf{t}(0) \left. \frac{\partial \mathbf{p}}{\partial \alpha} \right|_{\alpha=0}$  does not depend on the current estimate of the model parameters, neither on the input image. It is then constant throughout the fitting algorithm. This constancy is not assumed but derived from a first order expansion. The update is, then, not added to the current estimate as in the case of the AAM fitting algorithm, but composed with the current estimate.

$$\mathbf{p}(u, v; \alpha) \leftarrow \mathbf{p}(u, v; \alpha) \circ \mathbf{p}^{-1}(u, v; \delta\alpha) \quad (3.8)$$

Baker *et al.*, in [1], do not make the distinction between the reference and the image frames. A consequence of this, is that they require the set of warps to be closed under inversion. This leads them to a first order approximation of the inverse shape projection (called inverse warping in their nomenclature):  $\mathbf{p}^{-1}(x, y; \alpha) = \mathbf{p}(u, v; -\alpha)$ . This does not agree with the identity defining the inverse shape projection of Equation (2.14): As shown in Figure 3.1, a point from  $(u, v), q'$ , is mapped under  $\mathbf{p}(u, v; \alpha)$  to  $q$  in  $(x, y)$ . Hence, to agree with the identity, this point  $q$  must be warped back to  $q'$  under  $\mathbf{p}^{-1}(x, y; \alpha)$ . So, the displacement in  $q$  which should be inverted is the one from  $q'$ . However, in Baker *et al.* [1], the displacement function  $\mathbf{p}$  is inverted at the point  $q$ , leading to the point  $b$ , instead of  $q'$ . This is due to the fact that the distinction between the two coordinates systems is not made. This approximation is less problematic for a sparse-correspondence model as used by Baker for which the triangles are quite large (see Image (b) of Figure 2 of [1]), because the chances that both  $q$  and  $q'$  fall in the same triangle are much higher than in our dense correspondence model for which the triangles are much tinier. When  $q$  and  $q'$  fall in the same triangle, then their displacements are similar to a first order approximation, due to the linear interpolation inside triangles, and the error made during composition is small.

The improvement of ICIA over AAM is that, as the updating matrix is derived mathematically from the cost function and not learnt over a finite set of examples, the algorithm is more accurate and requires less iterations to converge. The fact that the Jacobi matrix is constant is the first

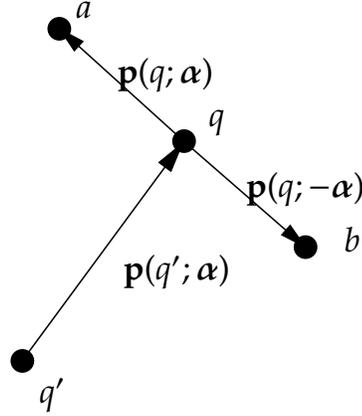


Figure 3.1: First order approximation of the inverse shape projection defined by Baker and Matthews in [1].

factor of the efficiency of the ICIA algorithm. The second factor is the fact that only the shape parameters are iteratively updated. The texture parameters are estimated in a single step after the recovery of the shape parameters. This is achieved by making the shape Jacobi matrix  $\mathbf{t}(0) \frac{\partial \mathbf{p}}{\partial \alpha} \Big|_{\alpha=0}$  orthogonal to the texture Jacobi matrix, by projecting out the shape Jacobi matrix onto the texture Jacobi matrix. It is therefore called the *project out* method. This induces a perturbation on the shape Jacobi matrix. However, if the texture model has few components (Baker and Matthews use less than ten components), then the error is small.

ICIA is an efficient algorithm. However, its domain of application is limited. It is a fitting algorithm for 2D AAM, hence cannot handle out of the image plane rotation and directed light. After discussion with Simon Baker, it also appears that it achieves its best performance when fitting images of individuals used for training the model. On novel subjects, the performance and accuracy are comparable to the AAM fitting algorithm. The ICIA is hence a person-specific fitting algorithm.

**ICIA applied to the 3DMM** As part of this thesis work, the ICIA fitting algorithm was adapted to use the 3DMM in [63]. Several modifications of the original algorithm had to be made in order to obtain accurate results. The first one was to use the precise inverse shape projection defined in Section 2.2.2, which makes the distinction between the reference and the image frames. The use of this inverse shape projection leads to the definition of the following cost function:

$$\|\mathbf{t}(u, v; \Delta\beta) \circ \mathbf{p}^{-1}(x, y; \gamma^d) \circ \mathbf{p}(u, v; \gamma^d + \Delta\gamma) - \mathbf{t}^{-1}(I(x, y) \circ \mathbf{p}(u, v; \gamma); \beta)\|^2 \quad (3.9)$$

where  $\gamma$  is the vector formed by the concatenation of the shape parameters,  $\alpha$ , and the rigid transformation parameters,  $\rho$ . The derivatives are precomputed at the parameters  $\gamma^d$ . Projecting out the shape Jacobi matrix onto the texture Jacobi matrix would significantly perturb the shape update, hence the project out method was not used. Thus, the texture parameters,  $\beta$ , are also iteratively updated. The texture parameter update is denoted by the vector  $\Delta\beta$ . This required the definition of the inverse texture transformation:

$$\mathbf{t}^{-1}(\mathbf{t}(u_i, v_i); \beta) = \mathbf{t}(u_i, v_i) - \sum_{k=1}^{N_T} \beta_k \cdot \mathbf{T}_{\cdot, i}^k \quad (3.10)$$

where  $\mathbf{T}_{\cdot, i}^k$  denotes the  $i^{\text{th}}$  column of the matrix  $\mathbf{T}^k$ , i.e. the deviation of the RGB colour of vertex  $i$  along the principal component  $k$ . This definition was chosen for the texture inverse because, then, a texture composed with its inverse, under the same set of parameters is equal to the mean texture:  $\mathbf{t}^{-1}(\mathbf{t}(u_i, v_i); \beta) = \bar{\mathbf{T}}_{\cdot, i}$ , see Equation ( 2.10 on page 20).

This algorithm is not as efficient as the original ICIA, but it is more accurate and its domain of applicability is also wider: It is able to fit input facial images at any pose and of any individual. This was demonstrated in the identification experiments reported in [60]. However, this algorithm does not handle directed light sources. There is a second predicament of this algorithm. In an implementation of ICIA, the parameters at which the derivatives are computed  $\gamma^d = [\alpha^{d^T} \rho^{d^T}]^T$  must be selected. A natural choice for the shape parameters is  $\alpha^d = 0$ . The selection of  $\rho^d$  is not as trivial, because the derivatives of the shape projections are computed in a particular image frame set by  $\theta^d$  and by  $\phi^d$ . Therefore, the two rotation angles should be close to their optimal values (depending on the input image). Hence, a set of Jacobians is computed for a series of different directions. During the iterative fitting, the derivatives used are the ones closest to the current estimation of the angles. Note that, at first, this approach might seem very close to the View-based approach [24, 54, 55]. The difference is, however, fundamental. In this approach, the extraneous (rotation) parameters are clearly separated from the intrinsic (identity, i.e.  $\alpha, \beta$ ) parameters. They are, however, convolved with one another in the View-based approach.

**2D+3D Active Appearance Model** Recently, Xiao *et al.* [70] extended the ICIA algorithm, originally developed for to the fitting of 2D AAM, to the fitting of a 2D+3D AAM. The aim of this fitting algorithm is to recover the 3D shape and appearance of a face very efficiently (more than 200 fps). They argued that the difference between a 2D AAM and the 3DMM is that the 3DMM codes, additionally to the (X,Y) coordinates, the Z coordinate for each shape vertex. We will see, shortly, that there is an additional major difference between these two models. Xiao *et al.* [70] showed that, in fact, any 2D projection of a 3D shape in the span of a 3DMM can also be instantiated by a 2D AAM, but at the expense of using more parameters. The 2D AAM requires up to 6 times more parameters than the 3DMM to model the same phenomenon. A weak perspective projection model was used to demonstrate this. This property would not hold for a perspective projection. They also showed that such a 2D AAM would be capable of instantiating invalid shapes, which is natural, as 3D transformations projected to 2D are not linear in 2D. The conclusion is that it is possible to fit facial images with non frontal poses with a 2D AAM trained on frontal pose. To ensure the validity of the shape estimated and to increase the efficiency of the algorithm, Xiao *et al.* impose the constraint that the 2D shape is a legitimate projection of a 3D shape modelled by a 3DMM. Thus, 3DMM shape model parameters and weak projection parameters are required to exist such as they produce a 2D shape equal to the one estimated by the 2D fitting algorithm. This is implemented as a soft constraint by augmenting the original ICIA cost function by a term proportional to the discrepancy between the 2D AAM estimated shape and the projection of the 3DMM shape. This seems to be a rather inefficient solution as, here, two shapes have to be estimated: The 2D AAM shape as well as the 3DMM shape and the projection parameters. The 3DMM model is only used to ensure the validity of the 2D AAM shape. The 2D AAM shape is used to warp the shape-free texture onto the image frame.

Xiao *et al.* [70] argued that the difference between a 2D AAM and the 3DMM is that the 3DMM codes depth information for each vertex and the 2D AAM does not. There is another major difference. The 3DMM is a dense model whereas the AAM is a sparse shape model. 76000 vertices are modelled by the 3DMM and a few tens by the AAM. This dense sampling enables the 3DMM to separate the texture from the illumination, thereby estimating a texture free of illumination effects. This is because, the shading of a point depends mostly on the normal of this point and on its reflectance properties. (The self-cast shadow term of the illumination depends also on the full 3D shape of the object.) Therefore, to accurately model the illumination, it is required to accurately model the normals. The normal of a point depends on the local 3D shape in a neighbourhood of this point. Hence, the 3D surface is required to be densely sampled in order to permit an accurate computation of the normals. Thus, it is not possible for a sparse 3D shape model to accurately separate the shading from the texture. It would then be difficult to relight a facial image with a different lighting configuration, as it is possible with the dense 3DMM, or to obtain high identification rates on a face recognition application across illumination.

As demonstrated in [70], the 2D+3D ICIA algorithm recovers accurately the correspondence

between the model 3D vertices and the image. These vertices are located on edges (eyebrows, eyes, nostrils, lips, contour). The edge features are hence used, through the input image gradient, to drive the fitting. Although, the correspondences are recovered, it does not imply that the estimated Z values of the landmarks is close to the Z value of the corresponding physical points on the face surface. In a single 2D image the only depth information is contained in the lighting. To estimate accurately the 3D shape of a surface, the lighting must be estimated and its 3D shape recovered using a reflectance model. For example, in a frontal image, the only clue about the distance between the nose tip and, say, the lips, is in the shading of the nose. Failing to take the shading into account in a fitting algorithm, as it is done by the 2D+3D AAM fitting algorithm, results in an imprecise 3D shape.

As the original ICIA fitting algorithm, the 2D+3D ICIA fitting is person-specific: it is able to fit accurately only individual within the training set. Its main domain of application is real-time 3D face tracking.

**Linear Shape and Texture Fitting algorithm** LiST [61] is a 3DMM fitting algorithm developed during the course of this doctoral work that addresses the same problem as the SNO algorithm in a more efficient manner by use of the linear parts of the model. (A fitting is 5 times faster than with the SNO algorithm.) It is based on the fact that if the correspondences between the model reference frame and the input image are estimated, then fitting the shape and rigid parameters is a simple bilinear optimisation that can be solved accurately and efficiently. To obtain a bilinear relationship between the 2D vertices projection and the shape and rigid parameters, a weak-perspective projection is used. One way of estimating these correspondences is by the use of a 2D optical flow algorithm similar to the one of Section 2.1.3. Hence an optical flow algorithm is applied to a model image synthesised using the current model parameters and the input image. These correspondences are then used to update the shape and rigid parameters. The texture and illumination parameters can also be recovered efficiently using the correspondences: The input image is sampled at the location given by the correspondences and first the illumination parameters are recovered using a Levenberg-Marquardt optimisation [58], while keeping the texture parameters constant. This optimisation is fast as only a few parameters need to be estimated. Then using the estimated light parameters, the light effect of the extracted texture is inverted, yielding an illumination-free texture used to estimate the texture parameters. The texture parameters are recovered by inverting a linear system of equations. It is hence efficient and provide an accurate estimate.

A drawback of this algorithm is that the 3D shape is estimated using correspondence information only, not using the shading, as it is done in the SNO algorithm or in the Multi-Features Fitting algorithm presented in Chapter 4.

## 3.2 Illumination-based methods

The methods described in the previous section aimed at recovering the shape and appearance of a face using a correspondence based model. The illumination model used was very crude: only a scalar global illumination parameter multiplying the texture was used. The methods described in this section use a Lambertian illumination model and estimate the 3D shape of faces from the shading. These techniques fall in the category of shape-from-shading algorithms. Several images of the same face under different light sources direction are required to recover the 3D shape. Correspondences between the pixels of these images are not estimated, they, rather, are assumed given by a manual alignment preprocessing step. Hence, the correspondence-based and the illumination-based methods are complementary: they each address one part of the problem. Only the 3DMM method addresses both problems in an unified approach.

A general comment about these methods is that they all assume a crude illumination model: Faces are assumed to have a Lambertian reflectance function. The intensity of a pixel is, then, the product of the albedo by the cosine of the angle between the light direction and the normal of the physical point imaged. The cast shadows and specular highlight are hence not modelled

contrasting with the phong illumination model chosen in this work. A selection of the most prominent techniques in this field is presented hereafter.

**The Illumination Cone** It was shown by Belhumeur and Kriegman [5] that the set of all  $n$ -pixel images of an object with a convex shape (i.e. with no cast shadows), at one pose, and with Lambertian reflectance, under arbitrary combinations of point light sources, forms a convex polyhedral cone in the image space  $\mathbb{R}^n$ . The convex cone is completely defined by the set of normals of the object.

Previous shape-from-shading approaches required the light source position to be accurately known. One of the improvement of the illumination cone, is that the illumination cone can be estimated without knowledge of light source locations. If three images of the object at the same pose, and illuminated from different directions, *with no pixel in attached nor cast shadow*, are available, then, the illumination cone can be estimated precisely by a singular value decomposition. However, there is only one light source direction for which no pixel is in shadow: the viewing direction. To address this issue, Georgiades *et al.* [32] proposed an iterative algorithm that treats some pixels as invalid and does not use them in the estimation of the cone. Seven images (and not three) are then required as input. The constraint that the recovered surface must be integrable is used.

To model pose variation, a view-based approach is used. The space of appearance variation of an individual across pose and illumination variations is the union of the illumination cones acquired at different poses.

This algorithm is applied to an identification task. A set of illumination cones is computed for every individual in the gallery set. Hence, at least seven images per pose are required for the subjects in the gallery set. Then identification of a test image is performed by computing the distance to all illumination cones. Minimising this distance yields the identity of the subject in the test image.

**The 9D Illumination Subspace** All the face recognition methods that model lighting effects assume a single directed light source. Ramamoorthi and Hanrahan [59], and Basri and Jacobs [3] presented an efficient way of taking into account an infinite number of light source directions, assuming the object is Lambertian and convex (no cast shadows). This method is based on the fact that the Lambertian positive cosine law is a low-pass filter of light.

Both lighting and Lambertian reflectance can be described as functions on the surface of a sphere. The intensity of (distant) light is function of its direction and can be represented as an environment map [26] for which one pixel gives the intensity of light in one direction. Lambertian reflectance is a function of the direction of the surface normal. According to Lambert's law, the intensity,  $i$ , reflected by a point is the product of its albedo,  $\rho$ , and the sum over all light directions,  $\mathbf{u}$ , of the intensity of the light in a direction,  $l(\mathbf{u})$ , times the positive dot product of the light direction and the normal,  $\mathbf{n}$ , of the point. In the continuous case, the sum is replaced by an integral.

$$i = \rho \int_{\text{sphere}} \max(0, \mathbf{u}^T \cdot \mathbf{n}) \cdot l(\mathbf{u}) \cdot d\mathbf{u} \quad (3.11)$$

In fact, the integral can be seen as a convolution over  $\mathbf{u}$  of the kernel,  $\max(0, \mathbf{u}^T \cdot \mathbf{n})$ , with the signal  $l(\mathbf{u})$ . As the integral is over the surface of a sphere, it is convenient to represent the lighting function  $l$  and the positive dot product function in spherical harmonics. Then the Funk-Hecke theorem [36] may be used, which states that the convolution on the sphere of two signals is equivalent to the multiplication of their spherical harmonics coefficients. It turns out, that the positive dot product, which can also be called half cosine kernel, acts as a low pass filter as the magnitude of its spherical coefficients decrease exponentially. Hence a second order spherical harmonic approximation of the half cosine already accounts for 99.22% of its energy. A second order spherical harmonic expansion of the lighting function involves only nine coefficients. Then nine basis image can be computed from a dense normal field of the points of a face surface. A

face image illuminated by any environment map, can then be synthesised by computing a linear combination of these nine basis images. The coefficients of the linear combination are the ones of the expansion into spherical harmonics of the environment map.

This algorithm was applied to face recognition by Basri and Jacobs [4]. Their method is quite demanding in terms of input, as it requires a 3D scan of each individual in the gallery set. This set of 3D scans also have to be labelled: The location of some features such as eye corners, mouth corners and nose tip must be manually set. A test image must be similarly labelled. This labelling is used to fit a 3D model of the gallery set to the test image. After model registration, the set of normals can be projected to the image frame. This assigns to each pixel a normal. This normal is correct if the identity of the 3D model is the same as the one of the test image. Then using the spherical harmonic representation, the illumination coefficients of the test image environment maps are computed. Using these coefficients, the texture is back-projected to the image frame and the discrepancy between this synthetic image and the test image is computed. This is performed for each individual in the gallery. Identification is made by assigning to the test image the identity of the 3D model with minimal back-projection error.

We considered using this illumination model for the 3DMM. Using it, we could easily and efficiently use a multitude of directed light sources. However, this method is not capable of rendering cast shadows nor specular highlight. Therefore, we favoured the phong illumination model with one light source.



## Chapter 4

# Multi-Features Fitting

We have seen in the previous chapter that, to date, the fitting algorithm that estimates the 3D face shape and albedo the most accurately, is the Stochastic Newton Optimisation algorithm [13]. Often it converges to a minimum close enough to the global minimum, such as the recovered shape and texture approximate well those of the photographed individual. However, in many cases, it converges to a local minimum far from the global one, yielding an unrealistic face, as seen on Figure 4.1.

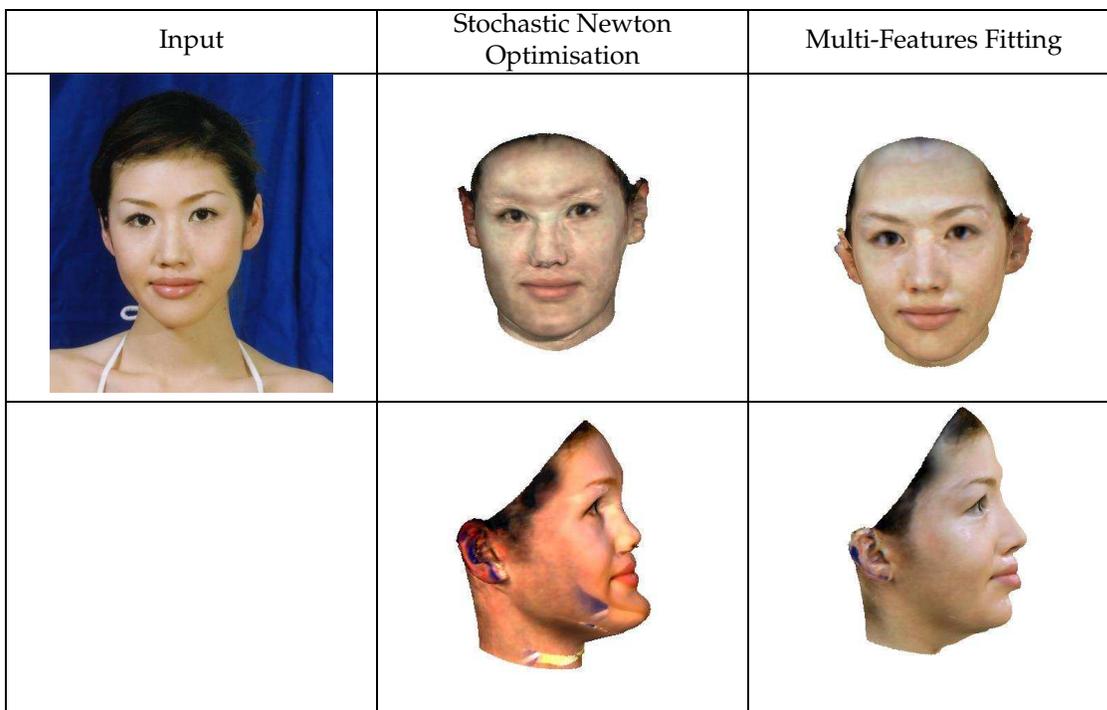


Figure 4.1: Example of poor fitting yielded by the Stochastic Newton Optimisation algorithm. Top row: fitting results obtained with the Stochastic Newton Optimisation (middle row) and with the Multi-Features Fitting (right row) algorithms using as input the photograph on the left. Bottom row: Novel view of the fitting results of the top row, with the texture extracted from the input image (Section 4.8 on page 63).

The contribution of this thesis is a fitting algorithm that is more robust to the local minima problem. It is inspired from the field of pattern classification in which stronger classifiers are constructed from multiple weaker classifiers. An overview of this theory is given in the next section which will serve as a basis for the construction of a robust fitting algorithm.

## 4.1 Using multiple features

### 4.1.1 Parallelism with pattern classification

The problem of pattern classification is to decide which of  $K$  classes an input pattern,  $\mathbf{v}$ , belongs to. This may be achieved by estimating the posterior probability  $p_k$  that  $\mathbf{v}$  belongs to the class  $k$ , for  $k = 1, \dots, K$ . We denote by  $\mathbf{p}$  the vector of the posterior probabilities of the  $K$  classes. In essence, in a probabilistic framework, the pattern classification problem is to estimate a function that maps a set of patterns  $\{\mathbf{v}\}$  to the posterior probabilities  $\mathbf{p}$ . Based on this mapping, the optimum decision can be taken by, what is called, a classifier.

The estimation of the mapping function is computed using a set of example patterns for which their class is known. This set of examples is called a training set and the function estimation is called learning. There exist a number of different approaches to learn this mapping from the training set ranging from neural networks to support vector machines. Additionally, various mappings can be computed by alteration of the training set or by modification of the set of measurements. Hence, different classifiers can be computed for the same application. Even though, these classifiers could have similar long-term performances, they each have their own strengths and weaknesses. Maximising their joined strengths and repelling their deficiencies is the endeavour of *classifier combination*.

There exists several techniques to combine classifiers. An overview of the most related approach to the current work follows.

- In a *cascade of classifiers*, a chain of classifiers is built with the purpose of adapting the classification computational cost to the input pattern. The individual classifiers making the chain are activated on demand: as long as a decision with a given confidence is not reached, the pattern is given to the next classifier in the chain. If a classifier in the chain reach a decision, then the classification stops and the decision is returned. The individual classifiers are then arranged by increasing discriminative power and increasing computational cost: The cheapest and least perfect answer is given by the first classifier. If this classifier is not discriminative enough, then the pattern is passed on to the second classifier. The advantage of this technique is to make a combined classifier as powerful as the best classifier of the cascade but at a reduced computational expense. This technique is applied to the detection of faces by a reduced set expansion of a Support Vector Machine in [62].
- In an *voting* scheme, the different classifier are seen as having equal competence and equal rights. Then, the classifiers are seen as experts and we seek to consolidate their decision. The Dempster's Rule provides a way to combine the decisions of the experts based on the theory of evidence [2]. This rule is explained in [64]. The Dempster's Rule says that if two independent experts yield the probabilities  $\mathbf{p}_k^1$  and  $\mathbf{p}_k^2$  for the class  $k$ , then the combined probability is:

$$\mathbf{p}_k = \frac{\mathbf{p}_k^1 \cdot \mathbf{p}_k^2}{\sum_{k=1}^K \mathbf{p}_k^1 \cdot \mathbf{p}_k^2} \quad (4.1)$$

Classification learns a mapping from a space of pattern to a class label. There is an analogy with the fitting problem which aims to provide a mapping between a 2D face image and the model parameters. The pattern classification problem is to find the class that maximises the posterior probability given the training set; whereas the fitting problem is to find the model parameters that maximise the posterior probability given the 3D Morphable Model. In a voting scheme several classifiers are used to gather evidence that are then combine using the Dempster's Rule. Similarly, a fitting algorithm could gather features (instead of evidence) and use them to maximise the posterior probability. The motivations behind combining classifiers is that sub-classifiers could focus on a restrained set of evidence (in the case of the voting scheme) or a restricted sub-space of the class label (in the case of the hierarchical classifier) and then be combined together to obtain a classifier that has better performance than if a single classifier was used. Similarly, a fitting

algorithm could be more robust to the imaging condition or to the local minima problem if it was based on multiple features. So instead of maximising  $p(x|I)$  as does the Stochastic Newton Optimisation algorithm, we maximise  $p(x|f^1(I), \dots, f^N(I))$ , where the feature function  $f^i(I)$  extracts a specific feature from the image. In the case of the pixel colour feature, for instance, the feature function is, naturally, the identity.

### 4.1.2 Multiple feature based posterior

A multiple features fitting algorithm aims at maximising the posterior probability of the model parameters given not only the input image but also different features of it. To make the notations more readable, we derive the formula using two features,  $f^1$  and  $f^2$ , and making their dependence on the input image  $I$  implicit. Using the Bayes rule and denoting by  $\theta$  the ensemble of model parameters, we have on the one hand:

$$p(\theta, f^1, f^2) = p(\theta) \cdot p(f^1, f^2|\theta) = p(\theta) \cdot p(f^1|\theta) \cdot p(f^2|f^1, \theta), \quad (4.2)$$

and on the other hand:

$$p(\theta, f^1, f^2) = p(\theta|f^1, f^2) \cdot p(f^1, f^2). \quad (4.3)$$

Combining these two equations yields:

$$p(\theta|f^1, f^2) = \frac{p(f^1|\theta) \cdot p(f^2|f^1, \theta) \cdot p(\theta)}{p(f^1, f^2)}. \quad (4.4)$$

If the features are independent, this equation becomes:

$$p(\theta|f^1, f^2) = \frac{p(f^1|\theta) \cdot p(f^2|\theta) \cdot p(\theta)}{p(f^1) \cdot p(f^2)}. \quad (4.5)$$

The aim of the multiple feature fitting algorithm is to maximise the above equation with respect to the model parameters,  $\theta$ . If  $f^1$  is the identity, as in the case of the pixel colour feature for instance, then multiplying the equation by its probability function will not influence the optimum  $\theta$ . Hence the term  $p(f^1)$  may collapse in the above equation. If we use a deterministic algorithm for the second feature, such as the *canny edge detector*, in the case of an edge based feature, then  $p(f^2) = 1$ . As a result this equation is simplified to:

$$p(\theta|f^1, f^2) = p(f^1|\theta) \cdot p(f^2|\theta) \cdot p(\theta). \quad (4.6)$$

Maximising this equation is equivalent to minimising the negative of its logarithm:

$$-\ln p(\theta|f^1, f^2) = -\ln p(f^1|\theta) - \ln p(f^2|\theta) - \ln p(\theta). \quad (4.7)$$

In this thesis, I call a *cost function* the negative of the logarithm of a probability function, as it is to be minimised.

As aforementioned, the use of combined classifiers, in the pattern classification field, stems from the fact that each classifier has different strengths and their combination maximises the overall performances. For example, an optimal cascaded classifier is constructed with a chain of classifiers such as the classifier  $i$  achieves its maximum discriminations on the patterns that are poorly discriminated by the first  $i - 1$  classifiers. Similarly, for the fitting problem, optimally, the different features cost functions should have local minima in different areas of the parameter space. Then, the combination of the multiple cost functions would have fewer local minima and a smoother behaviour on the parameter space as it is shown in Figure 4.2.

As it is shown on Figure 4.2, one local minimum would persist in the combined cost function, if this local minimum is consistent across all the different feature cost functions. If the features are independent this is not likely to happen. To make this even less likely to happen, a good strategy is to use as many features as possible.

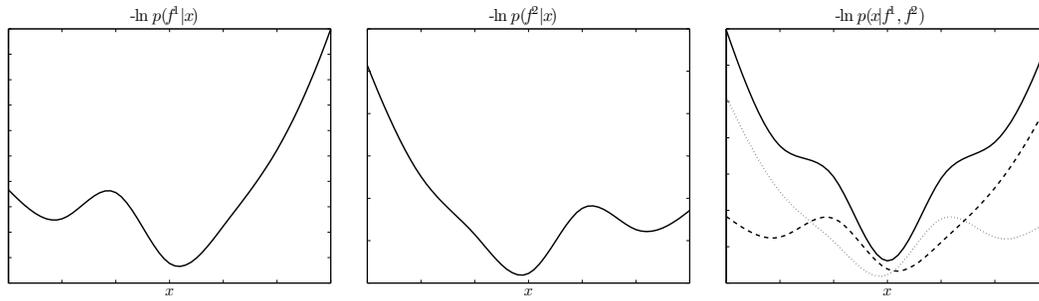


Figure 4.2: Two feature cost functions with, each, two minima. The last plot shows the multiple feature cost function (plain line) with one single minimum yielded by the addition of the two weak cost functions (dashed lines).

### 4.1.3 Feature Characteristics

It is shown in the previous section, that using different features provides a fitting algorithm more robust to local minima. The features that are used in the fitting algorithm can be categorised with respect to some characteristics detailed here.

**Image based vs. model based.** The last term of Equation (4.6) is the prior that enforces the fitting results to be a likely face. The prior probability of the model parameters is given by the Principal Component Analysis, that assumes the shape and the texture to have a Gaussian probability model. However, this assumption is suspicious as it appears that some faces with a high probability according to the PCA model are rather unlikely faces. For instance, the texture shown in Figure 4.3 is obtained by a linear combination of principal components whose coefficients are all within 2 standard deviations; and it clearly appears that this texture is unlikely. This texture is actually yielded by fitting an image illuminated by multiple light sources. As the fitting was done with only one light source, the light was poorly estimated, and rubs off on the estimated texture. This problem happens because the PCA model is not restrictive enough. In order to



Figure 4.3: Example of unlikely texture for which the PCA coefficients are all within 2 standard deviation. This texture was obtained by fitting an image with multiple light sources. The asymmetry of the texture is a residual from the illumination inversion algorithm and stems from the fact that the attached shadows were not properly recovered (because this light source was not well estimated). Thus the texture is affected by a lighting artifact and re-renderings at other illumination conditions would be poor.

reduce this problem, additional features are used that restrict the model parameters such as they generate more realistic faces. These features depend only on the model parameter and not on the input image, hence they are called *model based features*. On the other hand, the features that do depend on the input image are called *image based features*.

**Convexity.** A convex cost function has one stable state, the one and only global minimum. A descent optimisation algorithm, initialised anywhere in the parameter space, always recovers the global minimum of a convex cost function. Convexity is, hence, an important characteristic of a cost function. It is sometimes possible to prove that a cost function is convex, especially for the model based cost functions. However, it is not possible to assess the convexity of image based cost functions, in general, as they do depend on the input image; for these, changing the input image, modifies the behaviour of the cost function.

Comparing cost function only on their convexity property is not informative as two non-convex cost functions might have a different density and number of local minima. Therefore, we use the concept of domain of convergence. The *domain of convergence* of a cost function is the domain that includes the global minimum and on which this function is convex. It is defined as the domain around the global minimum on which the Hessian of the function is positive definite. The smaller the domain of convergence, the more non-convex is the cost function.

Finding the domain of convergence analytically for an image based cost function is not possible as an input image cannot be expressed analytically (an input image is merely a series of discrete pixels). Hence, the domain of convergence can only be estimated for a given input image. In a high-dimensional parameter space, estimating the domain of convergence is computationally expensive. What is tractable is to estimate the degree of non-convexity by introspecting the analytical form of a cost function and using some conjectures about input images. This is what we propose to do in the present work. We will hence grade cost functions as highly non-convex, non-convex or convex.

**Holistic vs. sparse.** A holistic feature involves the whole set of model vertices, whereas a sparse feature includes a limited set of model vertices. The cost function of a holistic feature depends on all the vertices of the model. So, the model parameter update, given at one iteration of the fitting algorithm, seeks to reduce the fitting error on the whole face area. For this feature type, the set of possible corresponding pixels for one model vertex is the entire set of pixels in the input image.

On the other hand, a sparse cost function involves a limited number of vertices. The set of image pixels that these vertices correspond to, is also restricted. This subset of image pixels is extracted by the feature preprocessing function  $f^i(I)$ . There are two types of sparse features: Either the vertices in the cost function are localised on a small region of the face, and then the purpose of the feature is to refine the 3D shape estimate in this area (the specular highlight feature is an example of sparse localised feature); or the model vertices used in the cost function are scattered throughout the face area, in which case, the purpose of the feature is to put these vertices efficiently in correspondence with the image (an example of such a feature is the edge based feature). Hence, the set of correspondence on the whole face area is rather coarse, as only information about a limited number of points is used. This type of feature is used at an early stage of the fitting to rapidly put in coarse correspondence the model with the image and, afterwards, holistic and localised features are used to refine the correspondence and the shape estimate.

**Biased vs non-biased** As seen in Section 4.1.2, the purpose of an image-based feature is to extract some information about the input image. It is then, this extracted information that is fitted to the same feature of the model. However it may happen, that the image feature and the model feature do not correspond to one another. For instance, in the case of the textured edge feature, the image edges are extracted by a canny edge detector. The model edges, on the other hand, are set by a human expert, independently of the input image (Figure 4.9 on page 47). It may happen that these edges do not correspond exactly to one another: Their might be missing edges in the image, or the canny edge detector might yield spurious edges due to lighting. In such a situation, we say that this image-based feature introduces a bias on the estimate. Despite this lack of accurate correspondence, a biased feature might still be used to bring the model in coarse correspondence with the image. As a result, a biased feature should be used in the early stages of the fitting but not in the latest stages.

In the next section, the different features used within this thesis are detailed.

## 4.2 Image based features

As mentioned in Section 4.1.3, we distinguish two types of features: Image based feature and model based features. In this section the image based features are detailed, leaving the model based ones for the next.

A good image based feature adheres to the two following criteria:

- The cost function of an ideal feature is convex. If it is not convex, as is the case for an image based feature (see previous section), it should have a large domain of convergence.
- A feature should be robustly extracted from any input image. This means that a feature based on the edges, for instance, should extract always the same intrinsic edges, located on the same facial features, independently of the input image, and should not extract artifact edges produced by shadows. For a feature based on the specular highlights, the robustness criteria means that the preprocessing algorithm should, for instance, detect the highlights on the face skin and not on the glasses.

### 4.2.1 Pixel colour feature

Fitting the pixels colour aims to recover the correspondence, the 3D shape, and the texture by matching the colour of the face image generated by the model to the colour of the input face image.

The correspondences are estimated using the edges *implicitly*: Edge information is convolved with the texture. Using the edges to recover the correspondence can only be done by a combined estimation of the texture and the shape. This is one source of the non-linearities of the cost function.

The second cue used to estimate the 3D shape is the shading. The shape from shading part of the cost function is also non-linear as the directed light source and the texture are unknown. As a result, the shape, the texture and the illumination condition are to be estimated together.

#### Pixel colour cost function

The feature used to fit the pixel colour,  $f^c$ , is simply the input image.

$$f^c(I(x, y)) = I(x, y) \quad (4.8)$$

Assuming the pixels to be independent, the probability of the input image given the model parameters is the product of the probabilities of each pixel given the model parameters.

$$-\ln p(I(x, y)|\theta) = -\sum_i \ln p(I(x_i, y_i)|\theta) \quad (4.9)$$

Assuming that the difference between the synthetic image and the input image is Gaussian and the variance is the same for all pixels, yields the following probability (with  $I^m(x, y)$  given in Equation 2.24 on page 24):

$$p(I(x_i, y_i)|\theta) = \exp\left(-\frac{1}{2 \cdot \sigma^2} \cdot (I(x_i, y_i) - I^m(x_i, y_i))^2\right) \cdot \frac{1}{\sqrt{2\pi\sigma^2}} \quad (4.10)$$

Hence, the pixel feature cost function, for one pixel, is formulated as follows:

$$-\ln p(I(x_i, y_i)|\theta) \propto \frac{1}{2}(I(x_i, y_i) - I^m(x_i, y_i))^2 \quad (4.11)$$

The cost function,  $C^c$ , can then be written as the inner product of the error vector  $\mathbf{e}^c$ .

$$C^c = \frac{1}{2} \mathbf{e}^{cT} \cdot \mathbf{e}^c, \text{ with } \mathbf{e}_i^c = I(x_i, y_i) - \mathbf{t}^c(u, v; \beta, \alpha, \gamma) \circ \mathbf{p}^{-1}(x_i, y_i; \alpha, \rho) \quad (4.12)$$

As mentioned in Section 2.2.2, the inverse shape mapping  $\mathbf{p}^{-1}(x_i, y_i; \boldsymbol{\alpha}, \boldsymbol{\rho})$  cannot be expressed analytically easily. However, composing Equation (4.12) on its right with  $\mathbf{p}(u, v; \boldsymbol{\alpha}, \boldsymbol{\rho})$ , using Equation 2.14 on page 22, and sampling the reference frame instead of the image frame, yields a cost function that depends on the forward shape mapping, which is expressed analytically.

$$\mathbf{e}_i^c = I(x_i, y_i) \circ \mathbf{p}(u_i, v_i; \boldsymbol{\alpha}, \boldsymbol{\rho}) - \mathbf{t}^c(u_i, v_i; \boldsymbol{\beta}, \boldsymbol{\alpha}, \boldsymbol{\gamma}) \quad (4.13)$$

Note that this error function is formally not the same as the one of Equation (4.12), as this one is sampled on the reference frame and the former one, on the image frame.

The plot of the pixel colour cost function along the azimuth rotation angle of the example image shown on Figure 4.4, is shown on Figure 4.5. It is clear, on this plot, the pixel colour cost function suffers from many local minima.

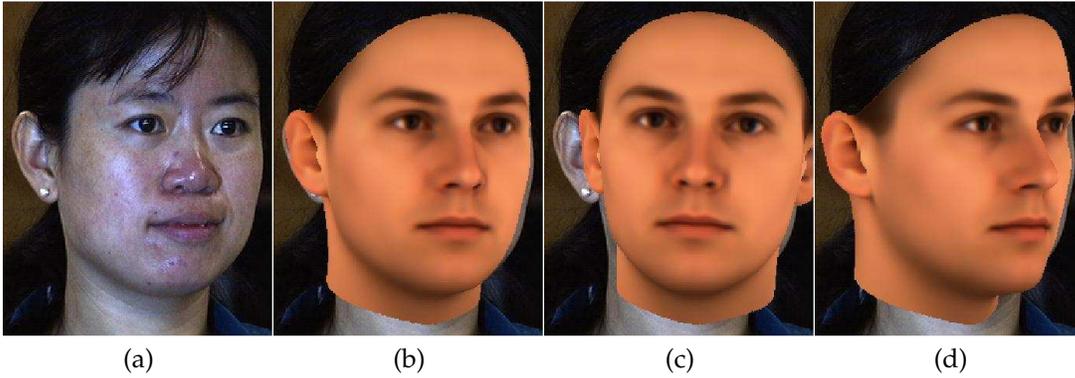


Figure 4.4: (a) Example of an input image on which the features cost functions are computed. The cost functions are computed using the initial values for the shape, texture and illumination parameters shown in the rendering (b). They are plotted for variations of the azimuth angle of  $\pm 20^\circ$  shown in the rendering (c) and (d).

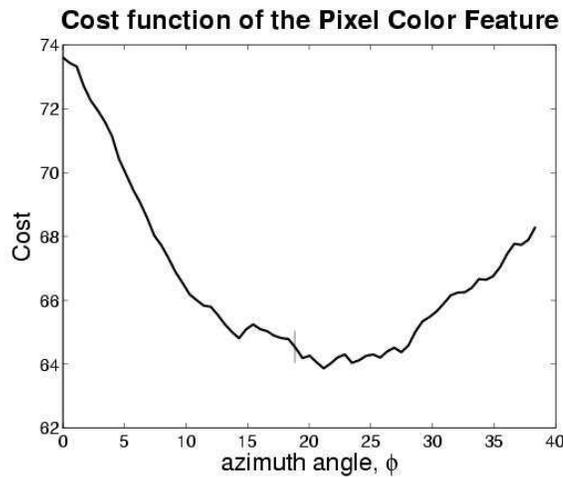


Figure 4.5: Plot of the pixel colour cost function along variations of the azimuth angle of  $\pm 20^\circ$  around the optimum. The cost function is evaluated on the input image shown on Figure 4.4 and for variations of the azimuth angle also shown on this figure. The vertical line is positioned at the value of the azimuth angle at the minimum reached by the fitting algorithm detailed in Section 4.6.

## Derivatives

In this section, the derivative of the pixel colour cost function with respect to a shape parameter is worked out. In the previous section, it was mentioned that the cost function is sampled on the reference frame. Sampling it at one vertex is not the wisest choice as computing the derivative of the unit-length normal at one vertex with respect to a shape parameter is involved and computationally expensive. A better choice is to compute it at the incenter of a triangle.

Differentiating Equation (4.12) with respect to the shape parameter  $j$ , yields:

$$\frac{\partial C^c}{\partial \alpha_j} = \frac{\partial \mathbf{e}^c}{\partial \alpha_j} \cdot \mathbf{e}^c \quad (4.14)$$

$\mathbf{e}^c$  is the vector of residuals evaluated at the incenters of the triangles sampled among the visible triangles.  $\mathbf{e}_i^c$  is the triplet of RGB residual at the incenter of the triangle  $i$ . Its derivative is computed using Equation (4.12) and 2.22 on page 24.

$$\frac{\partial \mathbf{e}_i^c}{\partial \alpha_j} = \frac{\partial I}{\partial x} \cdot \frac{\partial \mathbf{p}^x}{\partial \alpha_j} + \frac{\partial I}{\partial y} \cdot \frac{\partial \mathbf{p}^y}{\partial \alpha_j} - \frac{\partial \mathbf{t}_i^c}{\partial \alpha_j} \quad (4.15)$$

The first two terms of this equation is the part of the derivative that recovers the correspondence using the gradient of the input image. The interesting part of this derivative is the last term. This is the shape from shading part of the fitting algorithm. It is expressed using Equation 2.15 on page 23.

$$\frac{\partial \mathbf{t}_i^c}{\partial \alpha_j} = \mathbf{M} \cdot \mathbf{L}_{dir} \cdot \left( \frac{\partial \langle \mathbf{n}_i^t, \mathbf{d} \rangle}{\partial \alpha_j} \mathbf{t}_i + k_s \frac{\partial \langle \mathbf{r}_i, \mathbf{v}_i \rangle^v}{\partial \alpha_j} \mathbf{1}_{3 \times 1} \right) \quad (4.16)$$

where  $\mathbf{L}_{dir}$  is the diagonal matrix, with the directed light intensities on its diagonal,  $L_r^d, L_g^d, L_b^d$ . It is a simple matter of calculus to differentiate the diffuse and the specular highlight scalar products of this expression.

## Implementation Details

When implementing the pixel colour section of the fitting algorithm, the following issues are important.

- **Fitting the triangle centre.** The expression of the derivative of the normal at a triangle's corner with respect to the a shape parameter is much more complex than the one of the normal at a triangle centre. Therefore, it is the triangles centre that are fitted and not the model vertices. Hence, a shape and texture model expressed in the triangle centre must be computed. This is done by averaging the shape and texture principal components of the three corners of a triangle (in the `mf_f_o_i` MMT Function (See Appendix B).
- **Selection of the triangles to be fitted.** Opposed to the Stochastic Newton Algorithm, the triangles that are fitted are sampled once at the beginning of an optimisation process (a stage, as defined in Section 4.6) and not at each iteration. All the visible triangles are not fitted because the computational time of the algorithm would be prohibitive. Hence, the triangles that are fitted must be selected. The selection function must not be arbitrary, as it has a major influence on the fitting result. If, for example, the triangles are selected with a uniform probability on the visible triangles, then very few vertices will be sampled in the interesting regions of the face (nose, lips, and eyes) as most of the vertices are located on the cheeks, neck and forehead. Therefore, the fitting parameter `sel_coef` specifies the proportion of vertices sampled on the different face regions. Hence, a uniform probability is defined on each of these regions. This uniform probability is then multiplied by a factor that accounts for the surface of a triangle in the image frame (using the current fitting parameters). The triangles that constitute the pixel intensity cost function are then sampled using this probability distribution function.

- **Number of triangles.** The number of triangles that form the cost function is also an important parameter. The more triangles are included, the better will be the fitting, and the less overfitting will occur. However, the computational time for each iteration would be increased. I found out that using 1000 triangles was a good compromise.
- **Cast shadow and vertex visibility update.** Ideally, the vertices in the cast shadow and the visibility of each vertex should be recomputed after each iteration, using the current light direction and pose parameters. However, doing so would substantially increase the computational load. Hence the cast shadows and the visibility are computed only before each stage of the algorithm (Section 4.6).

### 4.2.2 Edge Feature

The aim of the fitting algorithm is to estimate the 3D shape. This is achieved by recovering correspondences with the 3D shape model and estimating the parameter of this shape model. So, estimating the correspondence is paramount. Edge feature is precisely suited to this problem, as it can be used to recover correspondence between the edges of the model face image and the ones of the input image more robustly than using the pixel intensity alone.

Using the pixel intensity alone, any vertex of the model could theoretically be set in correspondence with any pixel of the input image. However, the number of possible correspondence for edges is much more limited as the edge pixels constitute a sub-set of the set of pixels with a much lower cardinality (three orders of magnitude). As a result, the number of possible solution is considerably reduced. Hence, we say that the edge feature is a sparse feature.

The cost function that fits the edges gives a direct constraint on the shape. As opposed to the pixel colour feature that conceals shape information in the image gradient and in the shading, the edge reveals explicit shape information. As a result, the Jacobi matrix of the edge feature is more constant across the parameter space than the one of the pixel colour feature. Hence, it is reasonable to expect that the domain of convergence of the edge feature is wider than the one of the pixel colour feature.

The plot of the cost function along the azimuth rotation angle of the example image shown in Figure 4.4, is shown on Figure 4.6. It is much smoother than the plot of the pixel colour feature shown in Figure 4.5 and has only one minimum.

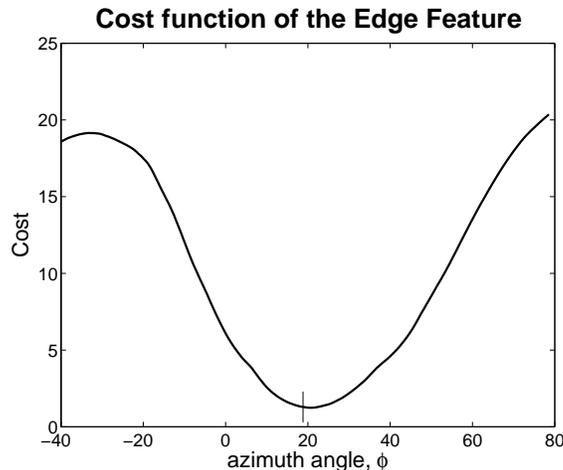


Figure 4.6: Plot of the edge cost function along variations of the azimuth angle of  $\pm 60^\circ$  around the optimum. The cost function is evaluated on the input image shown on Figure 4.4. The vertical line is positioned at the value of the azimuth angle recovered at the end of the fitting.

### Edge cost function

A deterministic algorithm is used to find the edges such as the *canny edge detector*. It provides a binary edge map for the input image:

$$f^e(I(x, y)) = \text{canny}(I(x, y)) \quad (4.17)$$

Each image edge pixel is assigned an index  $j = 1, \dots, J$  and their 2D positions are referenced by  $\mathbf{q}_j^e$ . I denote by the vector  $\mathbf{q}_{k(i)}^e$ , the 2D coordinates of the input image edge points in correspondence with the edge point of the model image  $\mathbf{p}_i$ . Here, the index  $i$  runs over all the visible model edge points. I assume, for now, that the mapping between the input image and the model image edge points,  $k(i)$ , is known. It is explain hereafter how it is estimated. If the edge points are independent, and identically normally distributed over the image, then the probability of the edge map, given the shape model parameter  $\alpha$  and the rigid transformation parameters  $\rho$  is

$$p(f^e(I)|\alpha, \rho) = \prod_i \exp\left(-\frac{\|\mathbf{q}_{k(i)}^e - \mathbf{p}_i\|^2}{2\sigma^2}\right) \frac{1}{2\pi\sigma^2} \quad (4.18)$$

Under these assumptions, the edge feature cost function is

$$-\ln p(f^e(I)|\alpha, \rho) \propto C^e = \mathbf{e}^{eT} \cdot \mathbf{e}^e, \text{ with } \mathbf{e}_i^e(\alpha, \rho) = \|\mathbf{q}_{k(i)}^e - \mathbf{p}_i\| \quad (4.19)$$

In this cost function, I have omitted the terms that do not depend on the parameters, and hence that have no influence on the position of the minimum in the parameter space.

One observation can be made on the edge based cost function in Equation (4.19) comparing it with the pixel colour cost function in Equation (4.12). The edge based function involves only 2D distances. As a result, the cost function to be optimised is not a composition of a shape term and a texture term, as in the pixel colour feature. Rather, the edge cost function includes only a shape term. Hence, we may expect its degree of non-convexity to be lower than the pixel colour cost function.

So, now the question is, how to estimate the mapping  $k(i)$ ? The same rule as the Iterative Closest Point (ICP) algorithm is used: The model edge point  $i$  is set in correspondence with the input image edge point closest to it:

$$k(i) = \arg \min_{j=1, \dots, J} \|\mathbf{q}_j^e - \mathbf{p}_i\| \quad (4.20)$$

Similar the ICP algorithm, fitting the edges involves, at each iteration, the following two steps: First the correspondence mapping,  $k(i)$ , is computed. Then, given this mapping, the model parameters are updated such as the cost function of Equation (4.19) is reduced. However, performing these two steps separately is not optimal, as modifying the mapping  $k(i)$  alters the minimum. Hence, it is desirable, to update  $k(i)$  along with the parameters  $\alpha$  and  $\rho$ . Levenberg-Marquardt ICP (LM-ICP) is an algorithm, proposed by Fitzgibbon [30], addressing this problem. The difference between the Fitzgibbon algorithm and this one, is that in [30], only the rigid parameters were estimated. Here, not only the rigid parameters  $\rho$  are estimated but also the non-rigid parameters  $\alpha$ .

The trick is to use the Chamfer Distance Transform (CDT) [14]. It is defined as the mapping  $D(\mathbf{x})$  that associates a point of the image space,  $\mathbf{x}$ , with the distance to the closest edge point:

$$D(\mathbf{x}) = \min_{j=1, \dots, J} \|\mathbf{q}_j^e - \mathbf{x}\| \quad (4.21)$$

The Chamfer Distance Transform at the image point,  $\mathbf{x}$ , in essence, incorporates two pieces of information: The closest edge point to  $\mathbf{x}$ , which, according to the ICP algorithm, is set in correspondence with  $\mathbf{x}$ ; and the distance between these two points. Hence, it replaces both the mapping  $k(i)$  and the distance  $\|\mathbf{q}_{k(i)}^e - \mathbf{p}_i\|$ . As a result, using the Chamfer Distance Transform, the edge cost function is transformed to:

$$\mathbf{e}_i^e(\alpha, \rho) = D(\mathbf{p}_i(\alpha, \rho)) \quad (4.22)$$

The edge cost function is computed simply by sampling the Chamfer Distance Transform at the position where the edge model points are projected under the current shape and projection parameters.

Note that the CDT depends only on the input image, not on the model parameters. Hence, it can be computed only once, at the beginning of the fitting algorithm. An efficient implementation of the computation of  $D(\mathbf{x})$  is proposed in [29], whose complexity is  $O(2wh)$ , where  $w$  and  $h$  are the width and height of the input image. An example of Chamfer Distance Transform is shown on Figure 4.7.

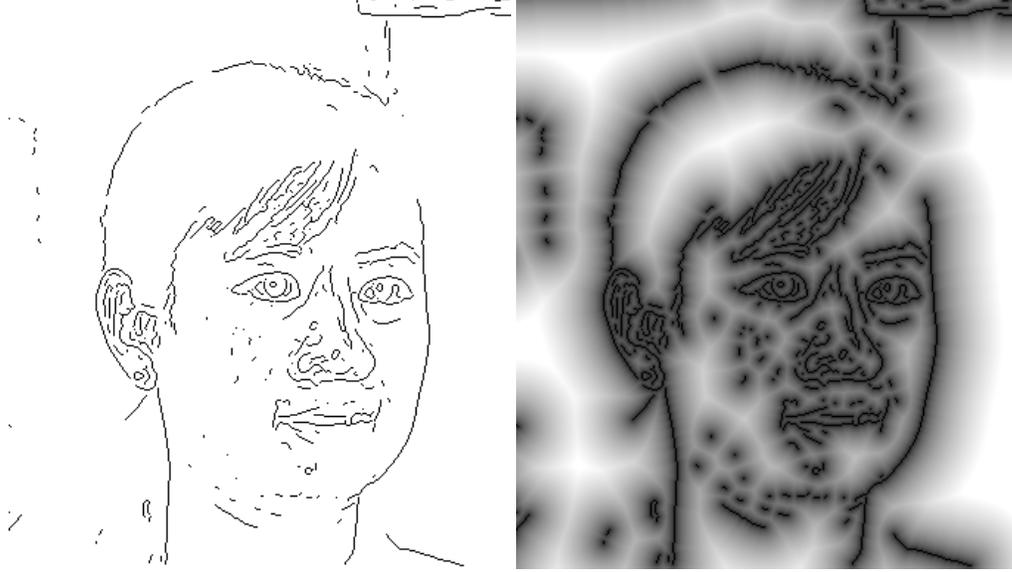


Figure 4.7: Edge map of the input image shown in Figure 4.4 and its Chamfer Distance Transform.

### Derivatives

Differentiating the edge cost function of Equation (4.22) with respect to the shape parameter  $j$  is straightforward and yields:

$$\frac{\partial e_i^e}{\alpha_j} = \frac{\partial D}{\partial x}(\mathbf{p}_i) \cdot \frac{\partial \mathbf{p}_i^x}{\partial \alpha_j} + \frac{\partial D}{\partial y}(\mathbf{p}_i) \cdot \frac{\partial \mathbf{p}_i^y}{\partial \alpha_j} \quad (4.23)$$

The advantage of the LM-ICP algorithm is that the derivatives of the CDT encode both the derivative of the mapping function  $k(i)$  and the one of the distance between corresponding points. As opposed to the ICP algorithm, in which the derivative do not depend on the mapping and, hence, regard it as being constant for each update of the parameter. As a result, in the LM-ICP algorithm, used here, the parameter update takes into account the variation of the mapping function. This is the reason why this scheme is called 'soft correspondence'. The derivatives of the CDT for the example of Figure 4.7 are shown on Figure 4.8.

### Textured edges

I mentioned in the previous section how to obtain the input image edge map (using the *canny edge detector*) and how to derive a cost function from it (using the Chamfer Distance Transform). However, I did not address yet, how to compute the model edges. This is the purpose of this section.

An edge result from a change in texture and not from shading induced by lighting. Hence, edges are independent of the lighting. In this work, I distinguish between two type of edges:

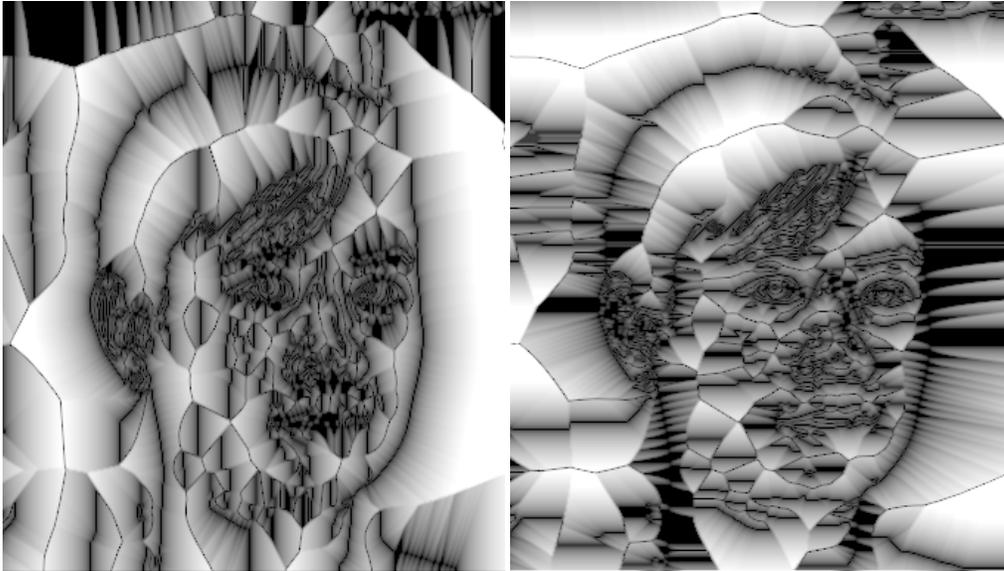


Figure 4.8: Derivatives along  $x$  (first image) and along  $y$  (second image) of the Chamfer Distance Transform shown in Figure 4.7. The images shown are actually the absolute value of the derivatives.

Firstly, edges marking the border between two facial features, such as the edges around the eyebrow marking the border between the eyebrow and the forehead on the upper part and the eye on the lower part. These edges are called *textured edges*. Secondly, edges located at the border between the face and a non-face part of the image. These edges are called *Contour edges* and will be addressed at the next section.

Textured edges do not move on the face surface as the pose of the face changes (only their visibility state can change). As, they do not depend on the lighting either, they are independent on imaging conditions. Hence, their location is constant on the reference frame. As the face surface of the 3D Morphable Model is densely sampled by 3D vertices, we may treat as model edges a constant subset of model vertices. The subset of model vertices chosen as textured edges is shown on the texture map of the average head on Figure 4.9.

Note that, generally, there is no one-to-one mapping between the model edge points and the image edge points: Some edge points could be not detected in the image. These are called *missed edges*. Additionally, artifact edges due to shading could be detected. These problems introduce a lack of correspondence between image and model edges. This is specially the case for textured edges, which are constant and set manually. Hence, the textured edge cost function may be biased. Therefore, it is used at the initial stage of the fitting algorithm to get the major face features in correspondence, but not at final stage.

### Contour edges

Computing contour edges is more involved than computing textured edges. As opposed to the textured edges, the vertices of the contour edges do depend on the rigid parameters. Hence, the set of contour vertex indexes vary with the pose of the face. Contour edges are defined as the set of points in the image plane that are on the border between the face area and the non-face area. There exists another definition for the contour: The contour of a 3D model projected on an image plane can also be defined as the set of vertices of the 3D model whose normal is orthogonal to the direction of the viewing vector. The viewing vector of a vertex connects the camera centre to the vertex. However, we do not use this definition as it yields contours that could be difficult to fit. As an example, consider the model image shown on the left of Figure 4.11. Some of the vertices on the upper part of the ridge of the nose would be considered on the contour according to the

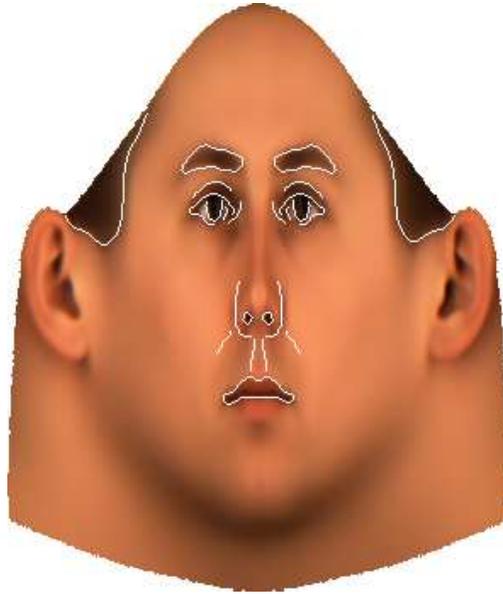


Figure 4.9: The textured edges on the face surface are shown on this figure in white on the texture map of the average head.

second definition, but not according to the first definition. These vertices are marked with a black line on Figure 4.10. As the norm of the image gradient on these contour pixels is small, these pixels would not be detected as edges by any edge detector.



Figure 4.10: The pixels marked by a black line are contours according to the normal-based definition but they would be difficult to fit as they would not be detected as edge points.

The fitting of contour edges proceeds in two steps: First, the model vertices on the contour for the current rigid parameters are selected, then these vertices are fitted to the input edges in the same way as for textured edges, i.e. using the Chamfer Distance Transform. The following algorithm is used for the selection of model contour edges:

1. *Vertex map*: First the vertex map is constructed. A vertex map is a rendering of the face, but instead of setting the R, G, B colour at one pixel a vertex index is set. The vertex whose index is set at one pixel, is the one that is projected nearest to this pixel.
2. *Binary vertex map*: The vertex map is converted to a binary format by thresholding the vertex map to one. The binary vertex map has, hence, a value of one for any pixel in the face area of the image, and zero, for any pixel outside the face area.

3. *Erosion and subtraction*: The morphological operation of erosion is applied to the binary vertex map, using as structuring element a disk of radius of one pixel. Then this eroded binary image is subtracted to the binary vertex map obtained at step 2. The resulting binary image is the contour map. Each pixel on this contour map set to one is on the contour of the face. Sampling the vertex map (obtained at step 1.) on these pixels yields the indexes of vertex on the contour.
4. *Contour vertex indexes*: Some of the contour vertices yielded by the previous step are not correct. This is because the 3D Morphable Model does not model the entire skull. It only models the points ranging from one ear to the other and from the top of the forehead to the neck. As a result, some parts of its contour, on the lower part of the neck and on the top of the forehead, are artificial. These artificial contour points are removed at this step. The good news is that the artificial contour are present always in the same area of the face. Hence, they are constant. Thus a list of the vertex indexes on the artificial contour can be made. So, finally, all the contour vertex indexes yielded by step 3 that are on the artificial contour list are removed. This step yields the indexes of the contour vertex used for fitting.

The result of this algorithm on an example image is shown in Figure 4.11. A rendering of the images yielded by each step of the algorithm is shown on this Figure.

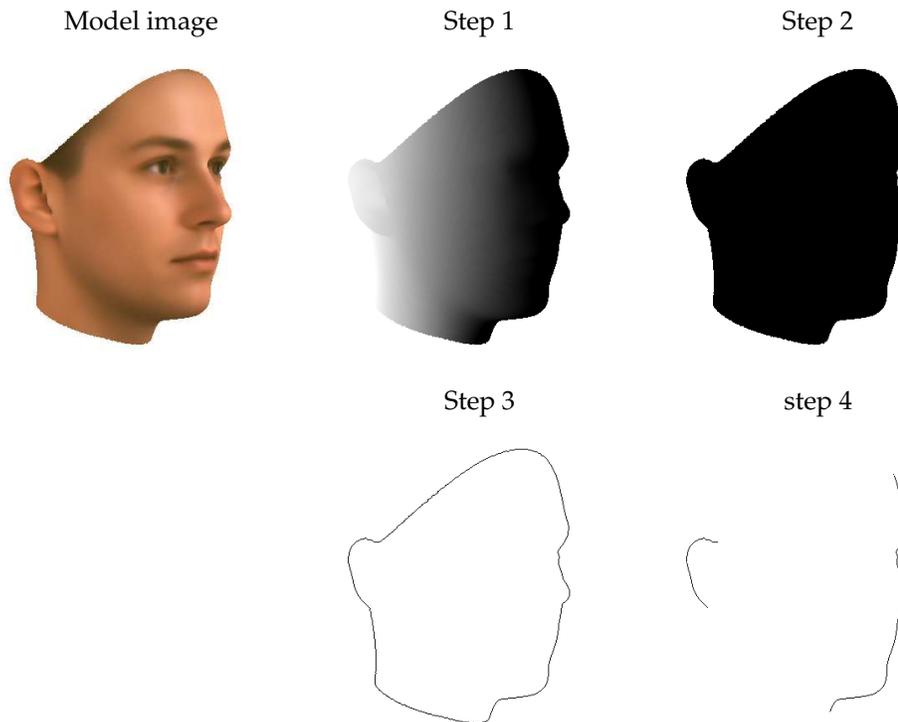


Figure 4.11: Rendering of the images yielded by the four step of the algorithm computing the model contour, when applied to the 'Model image' shown in the upper left part of this figure.

### Robust edge fitting

Two tricks are used to make the edge feature fitting more robust.

**Using edge orientation** The behaviour and the quality of the edge fitting depends mostly on the quality of the estimated correspondence,  $k(i)$ . Hence, it is paramount to increase as much as

possible the quality of these correspondences. Towards this goal, we use an additional clue: the edge orientation. The idea is that if two edge points have different orientation, say one is from an horizontal edge and the other from a vertical edge, it is quite unlikely that they correspond to one another, even if they are close from one another. So, are set in correspondence edge points that have a direction close enough. As directions, the direction of the normal to an edge,  $\varphi$ , is used and is computed using the image gradient:

$$\varphi = \text{atan2} \frac{\partial I / \partial y}{\partial I / \partial x} \quad (4.24)$$

The function  $\text{atan2}$  is the same as  $\tan^{-1}$ , except that, instead of returning a value in  $[-\frac{\pi}{2}, \frac{\pi}{2}]$ , it returns a value that ranges in all four quadrants, using the signs of the numerator and denominator. This formula is used for both the input image edges and the model edges. The orientation of the input image edge pixels is denoted by  $\varphi_j^I$  and the one of the model edge points, by  $\varphi_i^m$ .

The four quadrants in  $[-\pi, \pi]$  are segmented into regions,  $[\varphi_n^l, \varphi_n^u]$ , indexed by  $n$ . Then subsets,  $\mathcal{K}_n$ , of the set of input image edge pixels,  $\{1, \dots, J\}$ , are formed:

$$\mathcal{K}_n = \{j \mid \varphi_n^l \leq \varphi_j^I < \varphi_n^u\} \quad (4.25)$$

If the regions are overlapping, then edge pixels may be included in several regions. One Chamfer Distance Transform is computed for each of these regions:

$$D_n(\mathbf{x}) = \min_{j \in \mathcal{K}_n} \|\mathbf{q}_j^e - \mathbf{x}\| \quad (4.26)$$

Then a mapping between the model edge points  $i$  and the region indexes is computed, by assigning a model edge point to the region whose centre is the closest to its direction:

$$n(i) = \arg \min_n \left\| \varphi_i^m - \frac{\varphi_n^u - \varphi_n^l}{2} \right\| \quad (4.27)$$

As opposed to the input image edge points, a model edge point may be assigned to only one region. Finally, the error vector of the cost function is transformed to:

$$\mathbf{e}_i^e(\boldsymbol{\alpha}, \boldsymbol{\rho}) = D_{n(i)}(\mathbf{p}_i(\boldsymbol{\alpha}, \boldsymbol{\rho})) \quad (4.28)$$

It is important that the edge direction regions overlap. This allows for some discrepancy between the model edge direction and the input image direction. This is most useful at the beginning of the algorithm, when the orientation between edges might be quite different due to initialisation of the rotation angles being far from the global optimum.

Note that there is a major difference in the treatment of the orientation for textured and contour edges. The direction of the image edges can be computed for the textured and contour edges as aforementioned. However, the sign of contour edges is not meaningful as it depends on the colour of the background: If the contour is white, the direction of the image edge has another sign than if the background is black. To compensate for this, the sign of the direction of contour edges is not taken into account. This is achieved by modifying the way the input image edge pixels are separated into subsets. For the contour edges, instead of using the subsets defined by  $\mathcal{K}_n$  on Equation (4.25), I use subsets formed by  $\mathcal{K}_n^c$ , as follows:

$$\mathcal{K}_n^c = \{j \mid \varphi_n^l \leq \varphi_j^I < \varphi_n^u \text{ or } \pi - \varphi_n^u \leq \varphi_j^I < \pi - \varphi_n^l\} \quad (4.29)$$

Naturally, here,  $\varphi_n^l$  and  $\varphi_n^u$  are restrained to  $[0, \pi]$ .

**Missed edge points** If an edge point is not detected in the input image by the edge detector, it is said to be missed. When there are missed edge points, the corresponding model edge points are set in correspondence with wrong image edge points. It could even be that these corresponding

image edge points belong to another object in the image and are located far from the true edge point. As the cost function used in Equation (4.19) is a sum of square cost function, these points have a large influence on the parameter update. To reduce it, the Chamfer Distance Transform is thresholded, such as any distance larger than a given threshold is set to this threshold. Hence the Chamfer Distance Transform of Equation (4.21) is replaced by:

$$D(\mathbf{x}) = \min(\theta_e, \min_{j=1, \dots, J} \|\mathbf{q}_j^e - \mathbf{x}\|) \quad (4.30)$$

Note that it is possible to combine this thresholded cost function with the one that uses edge orientation.

Figure 4.12 shows the same plot as in Figure 4.6, but for the thresholded edge cost function. It can be seen that the robustness towards missed points is obtained at the expense of a reduced domain of convergence: Now, the domain in which the cost function is convex is smaller than the one of Figure 4.6.

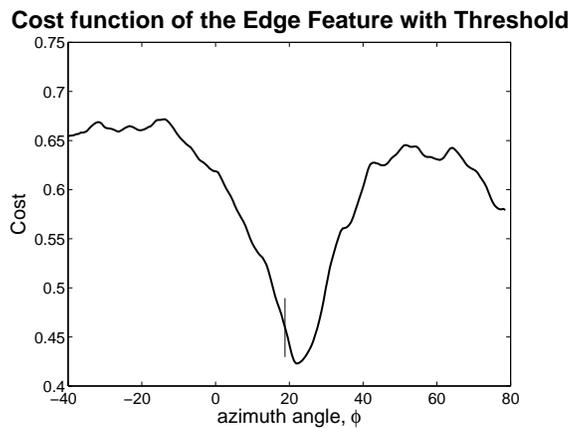


Figure 4.12: Plot of the thresholded edge cost function, with a threshold of 10 pixels, along variations of the azimuth angle of  $\pm 60^\circ$  around the optimum. The cost function is evaluated on the input image shown in Figure 4.4. The vertical line is positioned at the value of the azimuth angle recovered at the end of the fitting.

### Implementation Details

When implementing the edge feature of the fitting algorithm, the following issues are important.

- **Number of edge orientation.** For the textured edges, the angles between  $[-\pi, \pi]$  is divided into 8 bins and for the contour edges in 4 bins.
- **Contour edges computation.** Ideally, the model vertices that form the contour edges should be recomputed at each iteration, as the pose and shape parameters change at each iteration. However, for performance reasons, the contour edges are computed only before a fitting stage (Section 4.6).

### 4.2.3 Anchor points feature

An anchor point is a point of the image plane for which the correspondence with the 3D Morphable Model is known. In this thesis, we assume that a human user set this point by clicking on the image and on the same model point. Alternatively, an anchor point could be detected using an automatic detector. It was shown by Heisele in [41] that a few set of facial points can be automatically detected across various poses: In [41], 14 points are detected.

As the corresponding point of an anchor is known, the anchor point feature is the feature that gives the maximum information, but it is also the sparsest feature, as this information can be obtained only for few points.

As for the edge feature, the anchor cost function involves only shape measurements. There are two difference between the anchor cost function and the edge based one: Firstly, the correspondences do not have to be estimated for the anchor cost function as they are known. Secondly, rather than minimising the euclidean distance, the distances along both X and Y between the model and the image points are minimised:

$$C^a = \mathbf{e}^{aT} \cdot \mathbf{e}^a, \text{ with } \mathbf{e}_i^a(\alpha, \rho) = \begin{pmatrix} \mathbf{q}_i^{a,x} - \mathbf{p}_i^x(\alpha, \rho) \\ \mathbf{q}_i^{a,y} - \mathbf{p}_i^y(\alpha, \rho) \end{pmatrix} \quad (4.31)$$

Here, the index  $i$  runs over the anchor points. As mentioned in the edge feature section, if the correspondences are known, as it is the case for anchor point feature, and if a weak perspective projection is used, then the cost function is bilinear. Hence, few points are needed to find its minimum. This is the reason why a few anchor points may be used to find good initialisation value for the rigid parameters and for a few shape principal components. For instance, 6 anchor points were used to plot the anchor cost function shown on Figure 4.13 along variations of the azimuth angle.

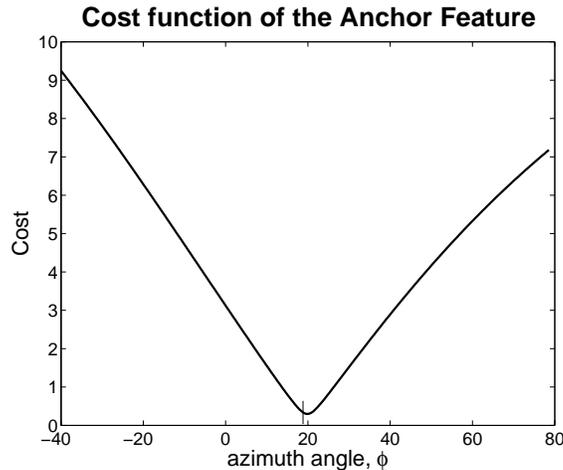


Figure 4.13: Plot of the anchor cost function, with 6 anchor points, along variations of the azimuth angle of  $\pm 60^\circ$  around the optimum. The cost function is evaluated on the input image shown in Figure 4.4. The vertical line is positioned at the value of the azimuth angle recovered at the end of the fitting.

As the correspondences of the anchor points are set manually, they may not be optimal, in the sense that these correspondence may not yield the smallest difference between the synthetic model image and the input image. Therefore, using this feature in the latest stage of the fitting algorithm introduces a bias. Hence, this feature is used only in the initial stage of the fitting algorithm to estimate initial rigid parameters and some shape principal components coefficients.

#### 4.2.4 Specular highlight feature

The specular highlight feature is the last image based feature developed in this thesis. This feature is based on the fact that the peak of the specular lobe of the BRDF (Bidirectional Reflectance Distribution Function) has the direction of the reflection vector (see Figure 4.14): A point of the face surface at the peak of the specular lobe has a normal that has the direction of the bisector of the angle formed by the light source direction and the camera direction. Hence detecting the specular highlight points yields a direct relationship between the geometry of the face surface at

these points and the camera and light directions. This relationship can then be used to recover the surface normal at these points if the camera and light directions are known.

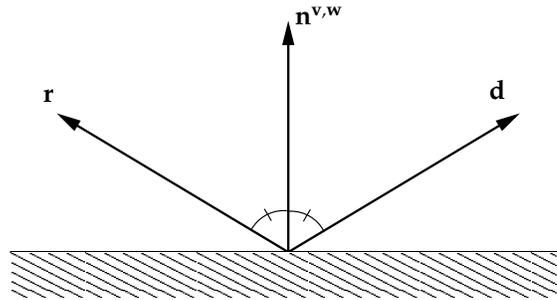


Figure 4.14: The light vector,  $d$ , is reflected along the reflection vector,  $r$ .

The camera direction (relative to the face, given by the rigid parameters) and the light direction influence all the pixels of the face area. Hence, these two directions can be recovered stably and accurately. So, I propose to first estimate these directions using all the pixels, then taking these directions constant, to refine the estimate of the normal of the points that are located on a strong specular highlight.

### Example

Figure 4.15 shows on the left an input image and on the right specular highlight pixels detected with the algorithm of Section 4.2.4 (The interesting points are located on the left of the tip of the nose). A comparison of the fitting results obtained with and without the specular highlight feature is shown on Figure 4.16. The leftmost image on this figure shows an input image, while the rightmost image is a photograph of the same lady as in the first image, at a novel view. The three images on the top row were obtained with a fitting algorithm that *did not use* the specular highlight feature; while the three images on the bottom row reflects results yielded with a fitting algorithm that used the specular highlight feature. The first column shows renderings of the fitting results. Here, both fittings appear to be similar. The second column shows the fitting results at a novel view (the same view as the rightmost image). The tip of the nose is better fitted when the specular highlight feature is used. This is more clear on a rendering of the face contour only, presented on the third column of the figure. The face contour is drawn on the novel view picture (rightmost image). The contour of the tip of the nose fits better the novel view photograph when the fitting result, made on the front view image, that used the specular highlight feature is used.

The conclusion is that the specular highlight is a good clue that gives direct depth information.

### Detecting specular highlights

Points at the peak of the specular lobe appear very bright. If the light source intensity is strong enough, these pixels even saturate. Hence, automatically detecting them can be done with a relatively simple algorithm: If two channels of a pixel are above a threshold, say 250, then this pixel is marked as being at the peak of the specular highlight lobe. An example of the specular highlight detection using this algorithm is shown on Figure ??(b): The detected pixels are marked by a black dot. The points on the left of the tip of the nose are correctly detected. Note that this algorithm would also detect spurious pixels imaged from an object that has a very bright colour, as the wall on the right of the face in the example image. This, however, does not perturb the fitting algorithm, as these points are not in the face area and hence, are not used for fitting.

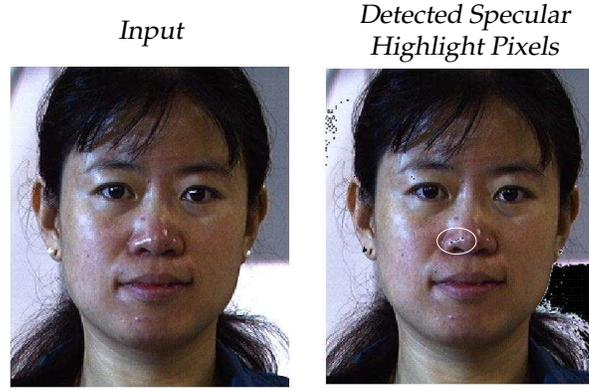


Figure 4.15: Result yielded by the specular highlight pixel detection algorithm. The left image is the input image on which the specular highlight algorithm, detailed in the next section, is applied. The detected points are marked as black pixels on the left image. The interesting pixels are located on the left of the tip of the nose (region surrounded by a white ellipse).

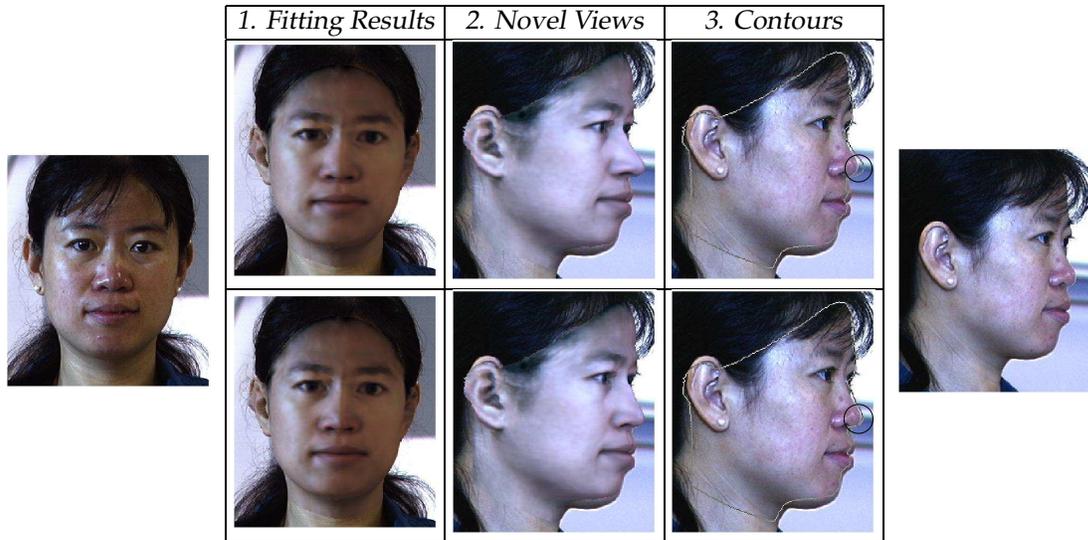


Figure 4.16: Example of the improvement yielded by the specular highlight feature. The leftmost image is a photograph used as input to compare the results of two instances of the fitting algorithm. The rightmost image is a photograph of the same lady at another view. The first instance of the fitting, whose results are shown on the top row, does *not* use the specular highlight feature. The second instance (bottom row) does use the specular highlight feature. The images on the first column show renderings of the fitting results obtained with both algorithms. The second column shows renderings at a novel view (the one of the rightmost image) of the same fitting results. The last column shows the contour of the model images of the second column. It is clear that the tip of the nose matches better the novel view image when the fitting algorithm uses the specular highlight feature (see the region surrounded by a black ellipse on the second and third columns).

### Specular highlight cost function

A point at the peak of the specular highlight lobe has a normal equal to:

$$\mathbf{n}^s(\boldsymbol{\alpha}, \rho) = \frac{\mathbf{r} + \mathbf{d}}{\|\mathbf{r} + \mathbf{d}\|} = \frac{\mathbf{v}(\boldsymbol{\alpha}, \rho) + \mathbf{d}}{\|\mathbf{v}(\boldsymbol{\alpha}, \rho) + \mathbf{d}\|} \quad (4.32)$$

where  $\mathbf{r}$  is the reflection of the light with direction  $\mathbf{d}$  (see graph on Figure 4.14). For a specular point, the reflection vector  $\mathbf{r}$  is equal to the viewing vector,  $\mathbf{v}$ , which is the unit-length vector connecting the point on the face surface and the camera.

So, the specular highlight cost function which pushes the normal of the corresponding model points to  $\mathbf{n}^s$  follows:

$$\mathbf{C}^s = \mathbf{e}^{sT} \cdot \mathbf{e}^s, \text{ with } \mathbf{e}_i^s = \mathbf{n}_{k(i)}^{t,w}(\boldsymbol{\alpha}, \boldsymbol{\rho}) - \mathbf{n}_i^s(\boldsymbol{\alpha}, \boldsymbol{\rho}) \quad (4.33)$$

Note that, here,  $\mathbf{e}_i^s$  is a sub-vector of  $\mathbf{e}^s$  with three components and that  $\mathbf{n}_i^s$  is the normal of the specular highlight point with index  $i$ , computed with Equation (4.32), and  $\mathbf{n}_{k(i)}^{t,w}$  is the normal in world coordinates of the corresponding model triangle. The aim of the specular highlight fitting is to modify the shape parameters  $\boldsymbol{\alpha}$  such as the model normal  $\mathbf{n}_{k(i)}^{t,w}(\boldsymbol{\alpha}, \boldsymbol{\rho})$  is close to the normal obtained by the specular feature,  $\mathbf{n}_i^s(\boldsymbol{\alpha}, \boldsymbol{\rho})$ . This cost function could be applied to change the vertex normal or the triangle normal. As for the pixel colour feature, the triangle normal is chosen owing to its simpler derivative.

Recall that the specular highlight feature is used at the final stage of the fitting algorithm, when the face rigid parameters and the light direction have a stable estimate. Therefore, this feature is used to update only the shape parameters  $\boldsymbol{\alpha}$ , not the rigid parameters neither the light direction. At this stage, the correspondence between the model and the image is also quite accurate. Hence  $k(i)$ , the mapping between the specular pixel  $i$  and the corresponding triangle, is set as follows:  $k(i)$  is the index of the triangle whose centre projects the nearest to the specular pixel  $i$ .

The derivatives of this cost function are calculated in Appendix A.1.

### 4.3 Model based features

The face analysis method that we use is model based. This means that we have constructed a model capable of synthesising human face images. The analysis task is then to find the model of the parameters explaining a given input face image, such as the synthetic image is as close as possible to the input image. Hence it is important that the model is able to render any input face image, such as the discrepancy between the model image and the input image can be small, and optimally null. We say then that the model is *generic*: it is able to reproduce any input face image. Another important characteristic of a model is its ability to generate only valid or realistic human faces. Then the model is said to be *restrictive*.

The image based features detailed in the previous section aimed at making the model image as close as possible to the input image by relying on the generative properties of the 3D Morphable Model. The purpose of the model based features addressed in this section is to restrict the model parameters such as the output of the fitting is a plausible face.

The importance of model based features is highlighted by the problem of over-fitting. Over-fitting happens when the fitted faces matches closely the input face *at the sampling pixel used in the fitting* but the discrepancy between the two face images is large elsewhere. It is the image based features that make the fitting result close to the input image at the sampling points, and the model based features that make it close for all other points.

#### 4.3.1 Gaussian Prior probability features

As seen in Equation 4.6 on page 37, the maximum a posterior estimate is obtained by maximising the likelihood multiplied by the prior. In the development of the 3D Morphable Model, it is assumed that the 3D shape and the texture in correspondence have a normal distribution. The diagonal covariance matrix of these two distributions was estimated by applying a Principal Component Analysis to a set of exemplar shape and textures. This analysis yields the following prior for the shape and texture:

$$p(\boldsymbol{\alpha}) \sim e^{-\frac{1}{2} \sum_i \frac{\alpha_i^2}{\sigma_{S,i}^2}}, \quad p(\boldsymbol{\beta}) \sim e^{-\frac{1}{2} \sum_i \frac{\beta_i^2}{\sigma_{T,i}^2}}. \quad (4.34)$$

Hence the cost function of the prior is:

$$C^p = \sum_i \frac{\alpha_i^2}{\sigma_{S,i}^2} + \sum_i \frac{\beta_i^2}{\sigma_{T,i}^2} \quad (4.35)$$

### 4.3.2 Texture Constraints Feature

What is the proportion of the albedo and the shading intensity in the intensity of a pixel? Resolving this ambiguity is one of the centre piece of the fitting algorithm. It is addressed using an albedo prior model that yields the likelihood of a face texture. In order to clearly separate the light and texture contribution in a face image, the texture model must be restrictive enough such as to allow only valid textures.

Figure 4.3 on page 38 shows the texture obtained by the PCA model with parameters within 2 standard deviations. This texture is quite unlikely but still its parameters are all within their valid range according to the PCA model. This means that the Gaussian distribution is not restrictive enough and improved fitting results could be obtained if it was better modelled.

The second column of Figure 4.17 shows an example of a poor fitting where the light intensity is under-estimated and the texture over-estimated. As these effects compensate each other, the fitting result rendered in image (b) look plausible. However when the texture from the image is extracted and the illumination inverted, the result is affected by some blueish regions that look unnatural. (The process of texture extraction and inverse illumination is detailed in Section 4.8 on page 63.)

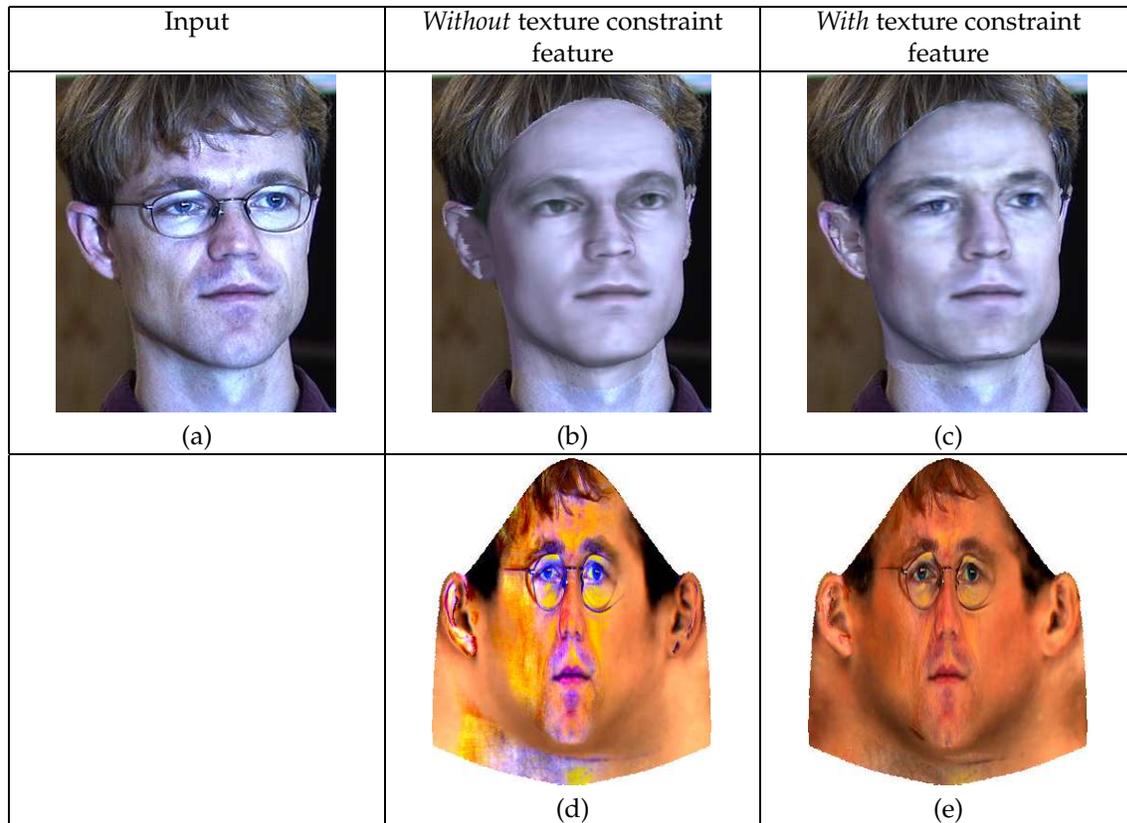


Figure 4.17: Comparison of fitting result with and without the texture constraint feature. (a): Input photograph. (b) and (c): Rendering of the fitting result obtained without and with, respectively, the texture constraint feature. (d) and (e): Inverse illuminated extracted texture obtained from the fitting results shown in images (b) and (c), respectively.

To address this problem, another model based feature is used that constrains the resulting texture. The valid intensity range of a pixel is  $[0, 255]$ . This constraint is not enforced by the Gaussian prior texture model. Imposing a valid range over all the texture points makes the texture less likely to be over-estimated (as in the second column of Figure 4.17) and, hence, makes the light intensity less likely to be under-estimated. The results shown in the third column of the same figure are obtained by imposing, during fitting, the model colour intensity to be in the range  $[5, 250]$ . This is achieved by using the following cost function:

$$C^t = \frac{1}{2} \mathbf{e}^{t^T} \cdot \mathbf{e}^t, \text{ with } \mathbf{e}_i^t = \begin{cases} t_i - l & \text{if } t_i < l \\ 0 & \text{if } l \leq t_i \leq u \\ t_i - u & \text{if } t_i > u \end{cases} \quad (4.36)$$

where  $t_i$  is the colour intensity of a channel of a model vertex used for fitting, and  $l$  and  $u$  are the lower and upper bounds of the valid intensity range. Note that this feature is not a hard constraint, but is rather a soft constraint, whose influence on the resulting estimate is relative to the other features.

## 4.4 Features Summary

A list of the features and their properties is presented on Table 4.1. In this table the last feature, the texture smoothness feature, is not addressed in this thesis because it was not evaluated. However, I think that the results could be improved by constraining the gradient of the texture map. A model of the gradient of the texture map can be made using the exemplar heads. Then an additional cost function could be used, in a similar fashion as the cost function of the texture constraint, that would constraint the estimated texture to have a valid gradient. This would, for instance, reduce the risk of obtaining a texture map similar to the one shown on Figure 4.3 on page 38, whose gradient is clearly invalid.

	<i>Image vs Model based</i>	<i>Convexity</i>	<i>Holistic vs. Sparse</i>	<i>Biased vs. Non-biased</i>
Pixel colour	Image based	Highly non-convex	Holistic	Non-biased
Textured Edge	Image based	Non-convex	Sparse, scattered	Biased
Contour	Image based	Non-convex	Sparse, scattered	Non-Biased
Anchor points	Image based	Non-Convex	Very sparse, scattered	Biased
Specular Highl.	Image based	Non-convex	Very sparse, localised	Non-biased
Prior Prob.	Model based	Convex	Holistic	Non-biased
Texture Constr.	Model based	Convex	Sparse, scattered	Non-biased
Texture Smoothness	Model based	Convex	Holistic	Non-biased

Table 4.1: Summary of features used for fitting and their main characteristics.

## 4.5 Robust Fitting

The cost functions of the image based features are derived assuming that the feature elements (pixels in the case of the pixel colour feature and edge points in the case of the edge feature) are independently and identically distributed (see, for instance, Equations (4.10) and (4.18 on page 44)). In the case of a normal distribution, this led to a quadratic cost function (see, for instance, Equations (4.12) and (4.19 on page 44)).

The problem is that a quadratic cost function, plotted in Figure 4.18, is not robust to outliers. In the context of this work, an outlier is defined as a feature element that does not adhere to the generating equation assumed by the model. For instance, as glasses or facial hair are not modelled by the 3DMM, these pixels and the feature elements derived from them can not be predicted by the model and hence are outliers. If the difference between the pixel values of the model estimate and the outlier pixels induce random and independent fluctuations normally distributed with zero mean, then the sum of squares cost function would be legitimate. However, this is rarely the case. Usually, outliers are not normally distributed and differ by large from the value predicted by the model: they have a large residual. The problem with a sum of square cost function is that the weight in the estimate yielded, of a feature element is proportional to its residual: the larger the residual of a feature element, the more it influences the solution. Hence, in presence of outliers, the sum of square solution might differ drastically from the outlier-free solution.

To increase robustness, a cost function must be more forgiving about outlying measurement; that is, it should increase less rapidly than  $e_i^2$ . Towards this goal, Huber [44], introduced the error norm  $\rho(x)$  and modified the expression of a cost function as follows:

$$C = \sum_i \rho(e_i) \quad (4.37)$$

When the error norm is chosen to be  $\rho(x) = x^2$ , the resulting estimate is the sum of square estimate. Then, Hampel *et al.* [38] introduced the *influence function*,  $\psi(x) = \frac{d\rho}{dx}$ . It characterises the bias that a particular measurement has on the estimate. For a least squares estimate, the influence of a measurement increases linearly and without bound with its residual (see Figure 4.18).

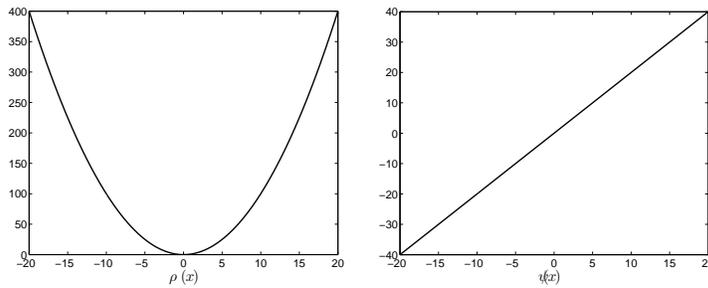


Figure 4.18: Quadratic error norm (left) and its influence function (right).

One example of robust cost function is the Huber's minimax estimator [44] for which the influence function is bounded:

$$\rho_\sigma(x) = \begin{cases} x^2 & |x| \leq \sigma \\ 2\sigma|x| - \sigma^2 & |x| > \sigma \end{cases} \quad \psi_\sigma(x) = \begin{cases} 2x & |x| \leq \sigma \\ 2\sigma \operatorname{sign}(x) & |x| > \sigma \end{cases} \quad (4.38)$$

A plot of the Huber function and its influence function is shown in Figure 4.19.

A function even more robust to large residuals is the *Lorentzian* estimator that assumes that the outliers have a Cauchy distribution [39, 67]. A Cauchy distribution is characterised by its heavy tails. It makes the influence of outliers fall off to zero:

$$\rho_\sigma(x) = \log \left( 1 + \frac{1}{2} \left( \frac{x}{\sigma} \right)^2 \right) \quad \psi_\sigma(x) = \frac{2x}{2\sigma^2 + x^2} \quad (4.39)$$

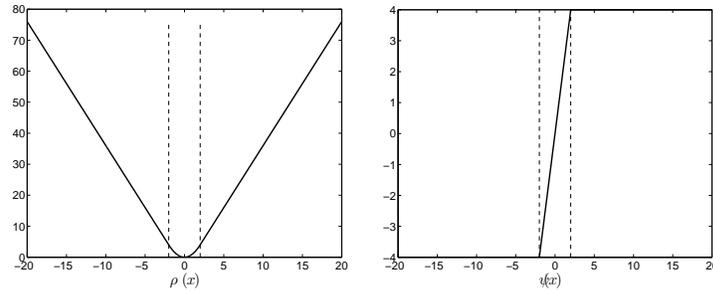


Figure 4.19: Huber error norm (left) and its influence function (right). The vertical lines indicate the  $\pm\sigma$  values.

A plot of the Lorentzian function and its influence function is shown in Figure 4.20.

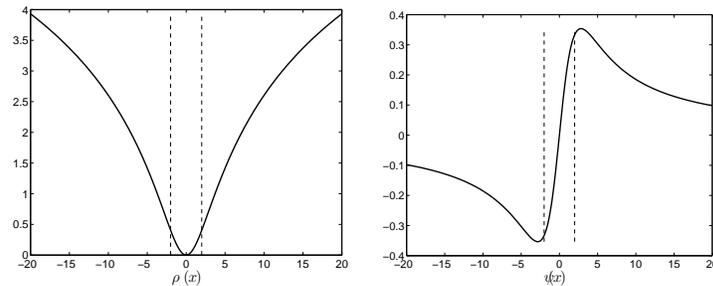


Figure 4.20: Lorentzian error norm (left) and its influence function (right). The vertical lines indicate the  $\pm\sigma$  values.

Note that all feature elements with large residual are not necessarily outliers. Especially at the beginning of the fitting process, due to the lack of correspondence, pixels with large residual could be inliers. This is the reason why I favour using the Huber function rather than the Lorentzian function.

**Example on synthetic images and synthetic noise** Optimisation theory says that if the noise distribution<sup>1</sup> agreed to the assumption, then the estimate would be unbiased and would have a minimum variance equal to the Cramer-Rao lower bound. An unbiased estimate approach the true value as the number of sampled points constituting the cost function tends toward infinity. The assumption of the noise distribution is then of critical importance.

This fact is shown by the following experiment. A facial image was generated with known parameters. Only the texture parameters were fitted, the shape, rigid and illumination parameters remained constant at their optimum values. The noiseless optimisation problem is then linear. As 99 texture parameters were used, in absence of noise, a cost function including  $99/3$  points would be enough to recover a perfect estimate in a single iteration of the fitting algorithm. (The division by 3 is because of the RGB channels available for each point.) The optimum noiseless image is shown on Figure 4.21.

Then, a significant amount of noise was added to the optimum image. A first input image was made with a Gaussian noise with zero mean and a standard deviation of 60. (The standard deviation of the pixels in the face area of the noiseless image is 64). A second input image was constructed with a Cauchy noise with zero as localisation parameter and 15 as scale parameter. (Note that this Cauchy noise is considerably larger than the noise of the first input image, as its standard deviation is more than 5,000). These two input images are shown on the top row of Figure 4.22. They were clamped to the range  $[0, 255]$  for visualisation purposes.

<sup>1</sup>The noise, here, is what the model cannot predict, i.e. the difference between the input image and its optimal model approximation.



Figure 4.21: Ground truth synthetic image used for the cost function comparison.

The two input images were fitted using a least squares cost function and a Lorentzian cost function. A rendering of the fitting result is shown on the middle and bottom row of Figure 4.22. Recall that a least square cost function is based on a Gaussian noise and a Lorentzian cost function, on a Cauchy noise. As seen on the figure, when the noise assumption is correct, the estimate is accurate and the cosine of the angle between the ground truth and estimated texture parameter vectors (used as identification measure in the next chapter) close to one. In the case of a Gaussian noise and Cauchy cost function, the estimate is also accurate owing to the robustness of this cost function. However, in the case of a Cauchy noise and a least squares estimate, the estimate is poor. This experiment leads to two conclusions. First, even in presence of significant noise, if the noise assumption is valid and if the number of sampled point is high (here, 20,000 points were used), the estimation is accurate. Second, if the difference between the input image and the model estimate is not Gaussian, a least square cost function should not be used, but rather a Lorentzian or Huber cost function.

Ideally, after the optimisation, the residual should be checked to validate the choice of the cost function. If the distribution of the residual does not agree to the cost function chosen, another run of the optimisation algorithm should be carried out with a new cost function. As this would significantly increase the fitting time, it is not performed in the multi-features fitting algorithm. However, here, a robust cost function is used.

## 4.6 Multi-Features Fitting Algorithm

In the previous sections of this chapter, I motivated the use of multiple features and detailed the features used in the *Multi-Feature Fitting* algorithm presented in this section.

Different features have different purposes: For example, edges are used to put the predominant facial features in correspondences, while pixel colour is used to infer the 3D face shape using shape from shading. Edges constitute a sparse feature: they contain information about a few facial points. Hence, only a few shape principal components can be stably fitted to them. On the other hand, as the image includes several thousands pixels, the pixel colour feature may be fitted stably to many more shape components. However the edge feature has a wider domain of convergence than the pixel colour feature. Hence, it should be used at an initial stage of the fitting. It should not be used at a final stage of the fitting, though, because of its bias (see Section 4.2.2).

These facts argue in favour of a multi-stage fitting algorithm. The Multi-Feature Fitting algorithm is sectioned in different stages: each stage fits a set of features to a set of parameters using a Levenberg-Marquardt optimisation algorithm[33]. The estimate yielded by one stage constitute the starting value of the next stage. Table 4.2 lists the five stages of the algorithm and details the feature and the parameters fitted at each stage.

At a given stage, a cost function is formed by the weighted addition of the cost functions of each feature fitted (See Section 4.1.2 on page 37). As explained at the beginning of this chapter, the feature cost functions are added because the different features are assumed to be independent.

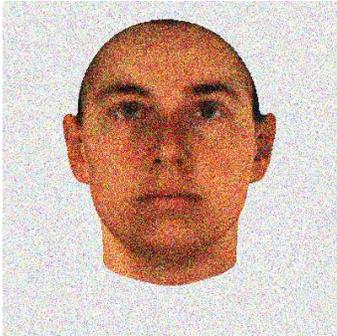
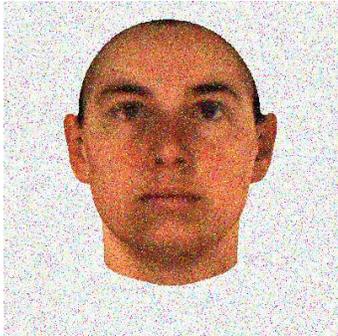
	<i>Gaussian noise</i>	<i>Cauchy noise</i>
<i>Input</i>	 $\sigma_N = 60$	 $\sigma_N = 15$
<i>LSQ fitting</i>	 $\cos(\beta, \hat{\beta}) = 0.957$	 $\cos(\beta, \hat{\beta}) = -0.1334$
<i>Lorentzian fitting</i>	 $\cos(\beta, \hat{\beta}) = 0.939$	 $\cos(\beta, \hat{\beta}) = 0.982$

Figure 4.22: Fitting results obtained on input images affected by Gaussian and Cauchy noise using least square and Lorentzian cost functions. *Top row*: Input images. *Middle row*: Renderings of the fitting results obtained with a least squares cost function. *Bottom row*: Renderings of the fitting results obtained with a Lorentzian cost function. The cosine of the angle between the ground truth and the estimated parameter vectors is printed underneath the each rendering.

For instance, the estimate yielded by step 4 is obtained by:

$$\min_{\alpha, \beta, \gamma} \tau^e C^e + \tau^c C^c + \tau^p C^p + \tau^t C^t \quad (4.40)$$

In this equation the different  $\tau$ 's are the weighting constants.

Figure 4.23 shows the result of the fitting after each stage of the algorithm on an example input photograph. Fitting Figure 4.23 on a 3.0 GHz Intel Pentium IV computer requires 70 seconds. The timing spent for the reparation and optimisation of each stage is listed in Table 4.3. The fitting of each step requires a preparation. For the edge feature, the Chamfer distance transform must be computed for each orientation. For the pixel colour feature, the triangles, on which

Stage Nb.	Features							Parameters				Nb. of Par.
	Anchor	textured edges	contour edges	pixel colour	texture constr.	specular highl.	prior	rigid	$\alpha$	illum.	$\beta$	
1	X	X						X				7
2		X	X				X	X	few			27
3				X			X			X	few	32
4			X	X	X		X	X	X	X	X	217
5			X	X	X	X	X		segm.		segm.	792

Table 4.2: List of features and parameters fitted at each stage of the Multi-Feature Fitting algorithm. On the last line 'segm.' stands for segmented (i.e. a segmented fitting is performed at this stage).

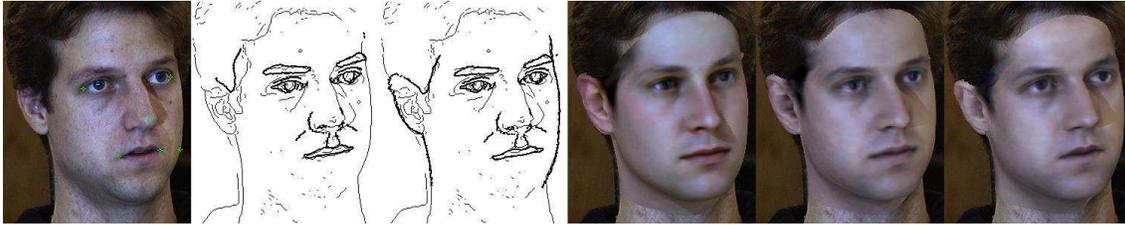


Figure 4.23: Fitting results obtained after each step of the Multi-Features Fitting algorithm. The first image shows the input photograph. The next two images show the fitted model edges in bold and the input image edges in plain. These are the results of the first two steps. Then, renderings of the results of the next three steps are shown overlaying the input image.

the cost function is to be computed, must be sampled; and a morphable model including only these triangles is constructed, which increases the computational efficiency of each iteration of the optimisation. The decrease of the pixel colour error for the stage 3, 4, and 5 is shown on

Stage Nb.	Preparation	Optimisation
1	3.82 s	0.08 s
2	5.36 s	0.36 s
3	1.75 s	0.37 s
4	4.58 s	9.11 s
5	14.74 s	29.41 s
Total	69.58 s	

Table 4.3: This table list the time spent by each stage of the Multi-Feature Fitting algorithm to fit the image shown in Figure 4.23. These timings were measured on a 3.0 GHz Intel Pentium IV computer. The preparation time is the time required to prepare the data before the optimisation.

Figure 4.24. There is one line per segment on the fifth stage because there is one optimisation per segment. The pixel error is computed on the sampling points. As the sampling points change at each stage, there is a small increase of the error at the beginning of one stage.

A Matlab implementation of this algorithm is provided in the Morphable Model Toolbox. This toolbox is detailed in Appendix B.

**Implementation Details** To implement the overall fitting algorithm, the following decisions must be taken.

- **Stages selection.** The number of stages included in the MFF algorithm, the features and the parameters fitted at each stages were chosen empirically such as to maximise the performances of the fitting algorithm. For instance, I tried to combine stages 1 and 2. At the

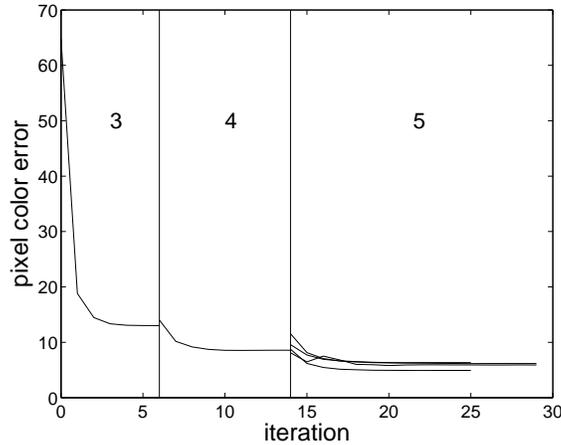


Figure 4.24: Plot of the pixel colour error at each iteration of stages 3, 4, and 5. For the stage 5, as there is one optimisation for each segment, there is one line per segment.

beginning of stage 1, nothing is known about the face. Hence, the contour at a frontal pose (from which the fitting is initiated) might be very different from the one of the input image. As the model contour is computed at the beginning of a stage, it is better to start its fitting when a good estimate of the pose is available. As this estimate can be obtained with inner edges and anchor points only, the first stage is constituted from these two features.

A second trial was made by combining stages 3 and 4. However, through experimentation, I observed, that the best convergence properties of the shape-from-shading term (fitted at stage 4), were obtained when a good initial estimate of the light direction was available. Therefore, I included the step 3, at which the pixel colour feature is used to fit only the texture and light parameters. The aim of this step, is to use the shape recovered by the edges, to get a good initial estimate of the texture and the light direction.

- **Weighting parameters.** The weight of each feature cost function, denoted by  $\tau$ , is the most important parameter of the fitting algorithm. An appropriate value was found by trial and error on a few example input images.
- **Robust cost function.** A robust cost function (Section 4.5) may be applied on each feature cost function. It is not necessary to apply on the edge feature as the edge feature is already robust (Section 4.2.2). Only the pixel intensity cost function is made robust. A Huber cost function is used for it. The sigma of the Huber function is constant for each stage (stage 3 and 4: 20, stage 5: 10).

## 4.7 Fitting Results

Some fitting results and reconstructions are presented on Figure 4.25. The input photographs are part of the PIE and FERET face image databases. The first column show photographs used as input to the fitting algorithm whose estimate is rendered overlaying the input and displayed on the second column. On the third column, the fitting results are rendered with the rigid and illumination parameters estimated by fitting the images shown on the fourth column. This is an illustration of the capacity of the model and the fitting algorithm to compensate for the imaging parameters while keeping the identity of the individual constant.

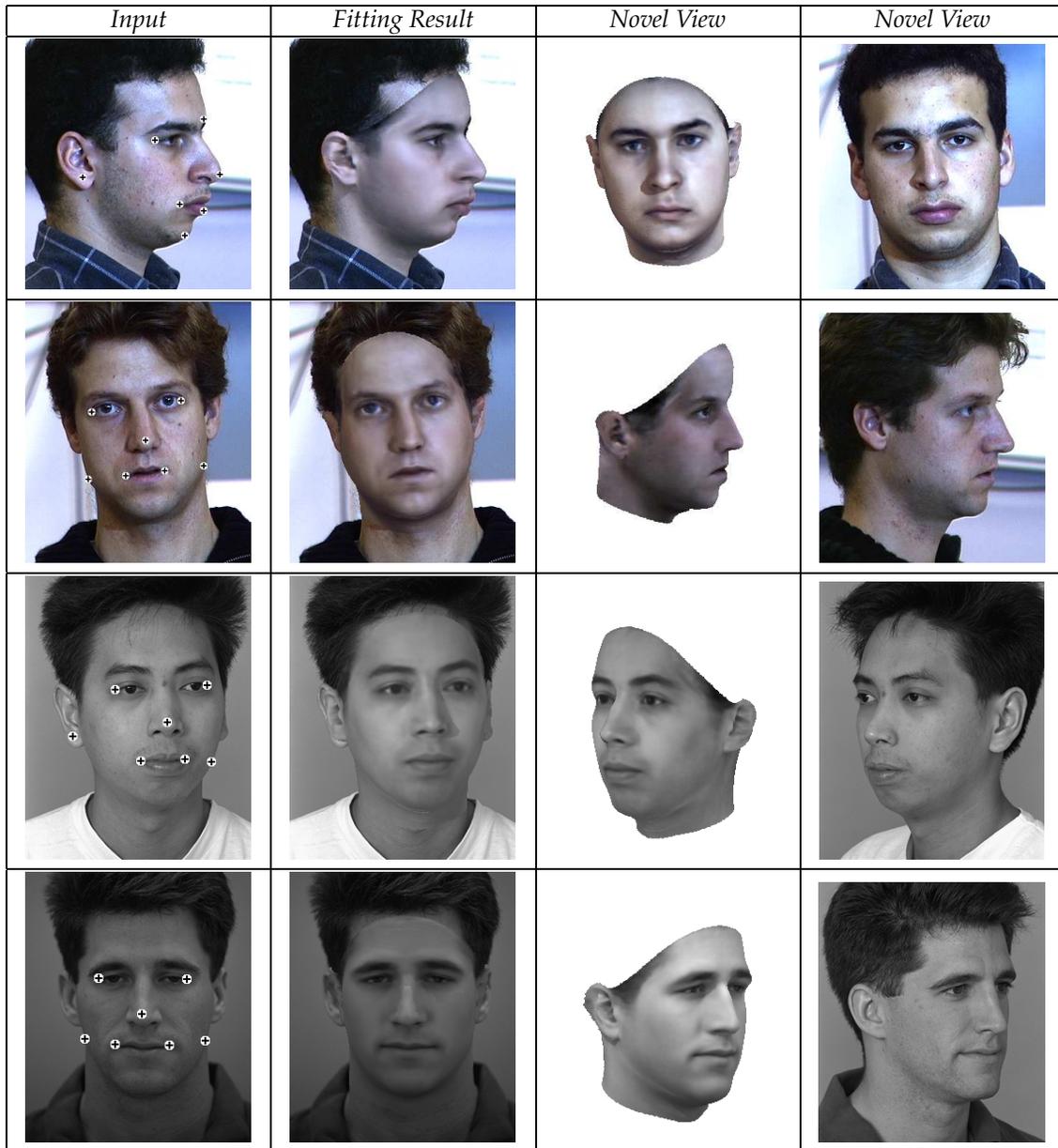


Figure 4.25: Example of fitting results. The second column shows renderings obtained from the input images shown on the first column and rendered overlaying them. On the third column are displayed renderings using the same shape and texture parameters as on the second column but with the rigid and illumination parameters of the fourth column, which shows photographs of the same individuals at a novel view.

## 4.8 Texture extraction and illumination correction

The purpose of the 3D Morphable Model and its fitting algorithm is to recover the 3D shape and the albedo of a face given a photograph. The shape and texture recovered are view and illumination invariant such as a face image, with the same identity, can be synthesised at any pose and under any illumination. Generally, the texture produced by the texture model is rather smooth and lacks the fine details. The photorealism of the model image is improved if the texture extracted from the input image is used instead of the model texture.

The notations used here are introduced in Sections 2.2.1 and 2.2.3.

A texture,  $\mathbf{t}^{e,C}$ , is extracted from the input image, for each visible vertices, by sampling the input image at the pixels where the vertices are projected.

$$\mathbf{t}^{e,C}(u_i, v_i) = I(x, y) \circ \mathbf{p}(u_i, v_i; \boldsymbol{\alpha}, \rho) \quad \text{for } (u_i, v_i) \in \Omega(\boldsymbol{\alpha}, \rho) \quad (4.41)$$

The next step is to inverse the illumination from  $\mathbf{t}^{e,C}$ , such as it can be re-illuminated in another environment. This is performed by inverting the light equations of Section 2.2.3 on page 23 and replacing  $\mathbf{t}_i^C$  by  $\mathbf{t}_i^e$ .

$$\mathbf{t}_i^{e,I} = \mathbf{M}^{-1} \cdot (\mathbf{t}_i^{e,C} - \mathbf{o}) \quad (4.42)$$

Then, inverting Equation 2.15 on page 23 yields the inverse illuminated extracted texture,  $\mathbf{t}_i^e$ .

$$\mathbf{t}_i^e = \left( \begin{pmatrix} L_r^a & 0 & 0 \\ 0 & L_g^a & 0 \\ 0 & 0 & L_b^a \end{pmatrix} + \langle \mathbf{n}_i^{v,w}, \mathbf{d} \rangle \begin{pmatrix} L_r^d & 0 & 0 \\ 0 & L_g^d & 0 \\ 0 & 0 & L_b^d \end{pmatrix} \right)^{-1} \left( \mathbf{t}_i^{e,I} - k_s \cdot \langle \mathbf{r}_i, \mathbf{v}_i \rangle^v \cdot \mathbf{1}_{3 \times 1} \right) \quad (4.43)$$

**Improving the illumination inversion on the cast shadows area** The third images on Figure 4.26 shows the texture extracted from the input image (shown on the first image) and using the fitting result (shown on the second image). The fitting result presents a cast shadow on the left of the nose that is not present in the input image. Several reasons can explain this fact. Firstly, the estimation of the light direction, or of the 3D shape could be poor resulting in an inaccurate cast shadow produced by the rendering. Secondly, the light source illuminating the face could be soft resulting in soft shadows that are not appropriately rendered with our model. A third reason is that the face image could have been manipulated and the cast shadow attenuated. When a cast shadow is present in the fitting result and not in the input image, the fitting result is darker, resulting in a bright spot in the illumination inverted extracted texture at the location of the cast shadow (as seen on the left of the nose on the third image of Figure 4.26). Therefore, an improved illumination inversion algorithm, which handles this case, is presented here.

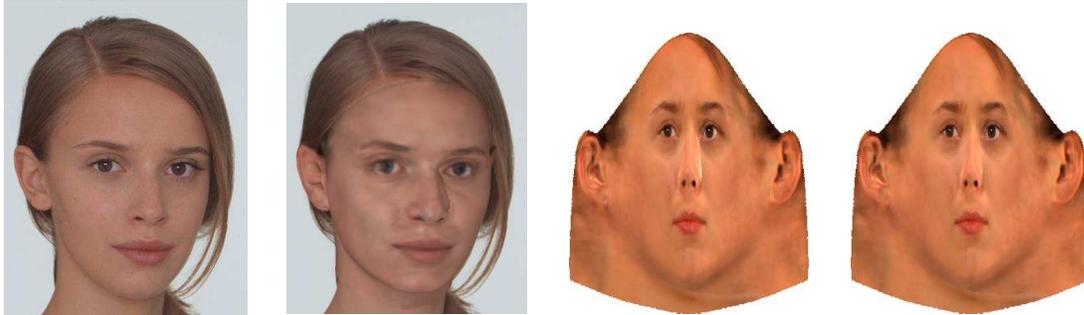


Figure 4.26: Texture extraction and illumination inversion. The second image shows a rendering of the fitting result obtained using as input the first image. The third image is the extracted texture without special cast shadow handling. The last image presents the extracted texture with the special cast shadow handling detailed in this paragraph. The illumination parameters, estimated by the fitting algorithm, are not optimal, as the reconstructed image presents a cast shadow on the left of the nose, not present in the input image. Hence, the reconstructed image is darker in this region due to the lighting. As a result, the illumination inverted texture is too bright in this area. This artifact has disappeared on the last image.

For one vertex in the cast shadow area, there are two ways to invert the illumination: (i) Using the algorithm detailed in previous paragraph and discarding the contribution of the directed light. This would assume that this vertex is indeed projected in the cast shadow area of the input image. (ii) Using the algorithm detailed in the previous paragraph and retaining the directed light, assuming that this vertex is projected to a point in the input image not in the cast shadow area. Then, the question is, which colour to use for this pixel, the one obtained assuming a cast shadow

(scheme (i)), or the one assuming no cast shadow (scheme (ii)). Even though the cast shadow is not properly estimated, the model texture could still be a good estimate of the real texture of the facial image. Therefore, the extracted colour chosen is the one closest to the model colour for the vertex.

This algorithm was applied to extract the texture of the input image of Figure 4.26 and its result is shown on the last image of this figure. The bright spot visible on the extracted texture using the original algorithm (third image) is not present anymore on the one using this novel algorithm.



# Chapter 5

## Applications

In this chapter, the Multi-Features Fitting algorithm is demonstrated on the following three applications: face recognition from a single photograph, expression transfer from one photograph to another and face tracking on a sequence of face images.

### 5.1 Identification and verification

In this section, the 3D Morphable Model and the Multi-Features Fitting algorithm are evaluated on two applications: identification and verification. In the identification task, an image of an unknown person is provided to the system. The unknown face image is then compared to a database of known people, called the gallery set. The ensemble of unknown images is called the probe set. In the identification task, it is assumed that the individual in the unknown image is in the gallery. In a verification task, the individual in the unknown image claims an identity. The system must then accept or reject the claimed identity. Hence, the person is not assumed to be in the gallery set. Verification performance is characterised by two statistics: The verification rate is the rate at which legitimate users are granted access. The false alarm rate is the rate at which impostors are granted access. The performance of a verification task does not depend on the number of individual in the gallery set, as opposed to the identification task. See, for instance, the Face Recognition Vendor Test 2002 report [56] for a more detailed explanation of these recognition tasks.

**Dataset.** We evaluate our approach on two datasets. *Set 1*: a portion of the FERET dataset [57] containing images with different poses. In the FERET nomenclature these images correspond to the series *ba* through *bk*. We omitted the images *bj* as the subjects present a smiling expression. A few examples of the FERET smiling images are shown on Figure. The purpose of this section is pose and illumination invariant face recognition. As expression invariant face recognition is not the aim of this work, the series of images *bj* is not part of the test set.

The portion of the FERET dataset, used in this series of experiments, includes 194 individual across 9 poses at constant lighting condition except for the series *bk*: frontal view at another illumination condition than the rest of the images.

*Set 2*: A portion of the CMU-PIE dataset [66] containing images of 68 individuals at 3 poses (frontal, side and profile) and illuminated by 21 different directions and by ambient light only. Figure 5.2 shows images illuminated by the 21 light directions. Among the 68 individuals, 28 wear glasses, which are not modelled and could decrease the accuracy of the fitting. None of the individuals present in these sets were used to construct the 3D Morphable Model. These sets cover a large ethnic variety, not present in the set of 3D scans used to build the model.



Figure 5.1: FERET images included in the series  $b_j$ , showing a smiling expression, omitted in the recognition experiments

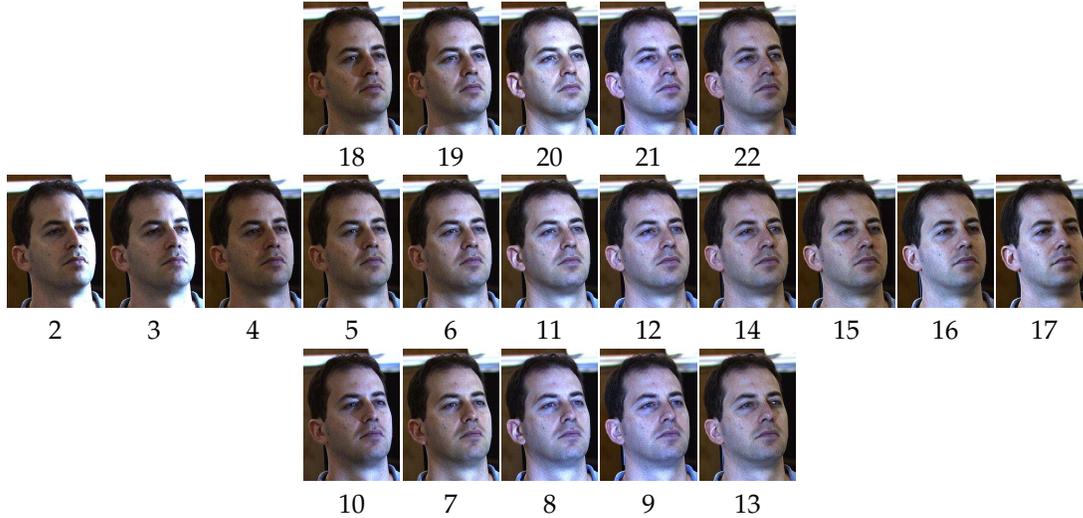


Figure 5.2: Images of illuminated by the 21 light directions of the CMU-PIE dataset. The number of each flash light appears the images.

**Distance Measure.** Identification and verification are performed by fitting an input face image to the 3D Morphable Model, thereby extracting its identity parameters,  $\alpha$  and  $\beta$ . Then, recognition tasks are achieved by comparing the identity parameters of the input image with those of the gallery images. We define the identity parameters of a face image, denoted by the vector  $\mathbf{c}$ , by stacking the shape and texture parameters of the global and segmented models (see Section 2.1.5 on page 20) and rescaling them by their standard deviations:

$$\mathbf{c} = \left[ \frac{\alpha_1^g}{\sigma_{S,1}}, \dots, \frac{\alpha_{99}^g}{\sigma_{S,99}}, \frac{\beta_1^g}{\sigma_{T,1}}, \dots, \frac{\beta_{99}^g}{\sigma_{T,99}}, \frac{\alpha_1^{s_1}}{\sigma_{S,1}}, \dots, \frac{\alpha_{99}^{s_1}}{\sigma_{S,99}}, \dots, \frac{\beta_{99}^{s_4}}{\sigma_{T,99}} \right]^T \quad (5.1)$$

We define a distance measure to compare two identity parameters  $\mathbf{c}_1$  and  $\mathbf{c}_2$ . The measure,  $d_A$ , is based on the angle between the two vectors (it can also be seen as a normalised correlation). This measure is insensitive to the norm of both vectors. This is favourable for recognition tasks as increasing the norm of  $\mathbf{c}$  produces a caricature which does not modify the perceived identity:

$$d_A = \frac{\mathbf{c}_1^T \cdot \mathbf{c}_2}{\sqrt{(\mathbf{c}_1^T \cdot \mathbf{c}_1)(\mathbf{c}_2^T \cdot \mathbf{c}_2)}} \quad (5.2)$$

**FERET Results.** Table 5.1 lists percentages of correct rank 1 identification obtained on the FERET dataset. The pose chosen as gallery is the one with an average azimuth angle of  $11.2^\circ$ , i.e. the

condition *be*. The first plot of Figure 5.3 shows the ROC for a verification task for the same gallery and the nine other poses in the probe set.

Probe View	Pose $\phi$	Correct Identification
<i>bb</i>	38.9°	92.7%
<i>bc</i>	27.4°	99.5%
<i>bd</i>	18.9°	99.5%
<i>be</i>	11.2°	<i>gallery</i>
<i>ba</i>	1.1°	96.9%
<i>bf</i>	-7.1°	99.5%
<i>bg</i>	-16.3°	95.8%
<i>bh</i>	-26.5°	89.6%
<i>bi</i>	-37.9°	77.1%
<i>bk</i>	0.1°	80.7%
<i>mean</i>		92.4%

Table 5.1: Percentage of correct identification on the FERET dataset. The gallery images are the view *be*.  $\phi$  denotes the average estimated azimuth pose angle of the face. Ground truth for  $\phi$  is not available. Condition *bk* has different illumination than the others.

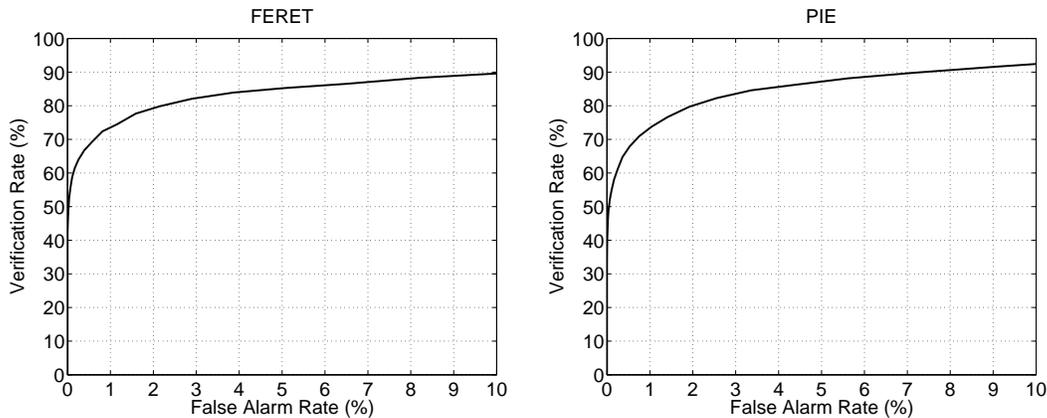


Figure 5.3: ROC for a verification task on the FERET and PIE dataset. The gallery images of the PIE dataset are at the side view and for the illumination 13. The gallery images of the FERET dataset are at the view *be* (Same as for Table 5.1).

**CMU-PIE Results.** The CMU-PIE dataset is used to test the performance of this method in presence of combined pose and illumination variations. Table 5.2 summarises the rank 1 identification performance averaged over all lighting conditions for front, side and profile view galleries. Table 5.3 details the identification result for each light direction. Illumination 13 was selected for the galleries. The second plot of Figure 5.3 shows the ROC for a verification using as gallery a side view illuminated by light 13 and using all other images of the set as probes.

**Discussions** A few remarks and conclusions may be drawn from the identification and verification experiments. Recognition of a profile view is consistently and substantially less accurate than for front or side views. This may be due to the fact that many important facial features such as the eyes, nose and mouth, are less visible on a profile view. Hence, these features provide fewer edges and are then more difficult to fit reliably. The best generalization performances are obtained for a side view gallery.

Gallery View	Probe View			mean
	front	side	profile	
front	99.9% (98.5–100.0)	98.4% (91.0–100.0)	75.6% (38.8–94.0)	91.3%
side	96.4% (89.6–100.0)	99.3% (97.0–100.0)	83.7% (52.2–100.0)	93.1%
profile	76.3% (64.2–91.0)	86.0% (67.2–98.5)	89.4% (64.2–98.5)	83.9%

Table 5.2: Mean percentage of correct identification obtained on the PIE data set, averaged over all lighting conditions for front, side and profile view galleries. In brackets are percentages for the worst and best illumination within each probe set. The overall mean of the table is 89.4%.

light	Front Gallery			Side Gallery			Profile Gallery		
	Front	Side	Profile	Front	Side	Profile	Front	Side	Profile
ambient	100	96	79	97	100	81	76	82	91
2	100	91	58	90	97	61	75	67	76
3	100	97	39	93	100	52	73	78	64
4	99	99	55	94	97	54	72	78	75
5	100	100	70	94	97	76	64	75	79
6	100	97	78	91	99	85	69	81	93
7	100	99	76	94	100	87	73	78	93
8	100	96	75	99	100	91	66	85	97
9	100	100	82	99	100	94	78	97	99
10	100	99	75	94	99	78	69	72	78
11	100	97	78	97	100	90	70	90	96
12	100	100	84	99	100	100	79	93	99
13	-1	100	87	100	-1	96	85	96	-1
14	100	100	91	100	100	97	91	99	97
15	100	100	88	100	100	94	88	99	97
16	100	100	88	100	100	96	91	99	99
17	100	99	79	100	100	93	85	97	99
18	100	100	61	94	97	67	69	79	72
19	100	97	67	91	100	73	64	75	82
20	100	100	79	97	100	91	75	85	99
21	100	100	81	100	100	91	82	94	99
22	100	100	94	100	100	97	87	97	99
mean	100	98	76	96	99	84	76	86	89

Table 5.3: Details of the identification results in presence of combined pose and illumination variation for the CMU-PIE dataset. The first column lists the directed light number. The three groups of triple columns show the results obtained for three different galleries: Front, side and profile. For each gallery, the percentage of correct identification for a pose-illumination condition is displayed.

Good results are obtained for identification across light variation. For front and side views, the lowest identifications percentages are obtained on the flash light number 2. On this illumination condition, a large part of the face area is either in the shadow or on a specular highlight (see Figure 5.4). In these regions, shape-from-shading is not applicable and the only depth clue used by the fitting is the specular highlight feature. Therefore, the 3D shape estimation is not optimal, explaining the poor pose generalisation performance for this illumination condition.

Additionally it was shown in [11] that the identification performance depends on the fitting accuracy. The more precise the fitting, the better is the identification across pose and illumination



Figure 5.4: Example of imaging condition that yields lower identification performance. These photographs are part of the CMU-PIE dataset and they were taken with the flash light number 2.

variation.

## 5.2 Automatic facial expression transfer

In this thesis, I focused mostly on three modes of variations of human face photographs: identity, pose and illumination. In this section and the next, I address another major mode of variation: facial expression. The conclusion that will be drawn is that the change of expression can also be handled by the 3D Morphable Model paradigm. One of the major achievement of the 3DMM, is its ability to model the identity, the pose and the illumination independently from each other, using independent parameters. This enables the facial analysis to be invariant to the imaging conditions but also to estimate these parameters separately and to compensate for them. The goal of the expression modelling is similar: the expression variations should be modelled independently from the identity. This would allow the separation of these two sources of variations in an input image. From there, any image with non-neutral expression could be fitted and, then, its expression modified. For instance, the expression of two face images could be exchanged while preserving their identity. An example of this application is shown on Figure 5.5, where the expression of Mr Bean is transferred to Tony Blair and vice-versa.

**Expression model.** When the expression of a given face vary, its shape is modified but not the texture: the albedo of a point on the face remains constant. Hence, the expression part of the 3d morphable model only affects the shape part of the model, not the texture.

In the 3DMM, the identity subspace is modelled by deviations from the mean identity. Similarly, the expressions are modelled by deviations from the identity subspace. This is based on the assumption that the expression subspace is orthogonal to the identity subspace and that the expressions of each individual have a general likeness. While it is true that the precise motion of, say, the lips between the neutral expression and, say, a smile is different for two individuals, we can nonetheless say that these motions are similar enough, such as an accurate model can be constructed from them. Then, an expression can be simply added to a neutral face to create a face with an expression and with the same identity as the neutral face.

The dataset used to construct the expression morphable model is composed of 3D scans of individuals with a neutral expression and with various expressions such as smiling, frowning and mouth opened. They are shown on Figure 5.6. The scan of the first individual was acquired with a different scanner and under different illumination condition than the other scans, which explains its different overall colour. Some people were marked with black and white dots to ease the correspondence computation of the 3D scans. Note that, to avoid a topology conflict with the

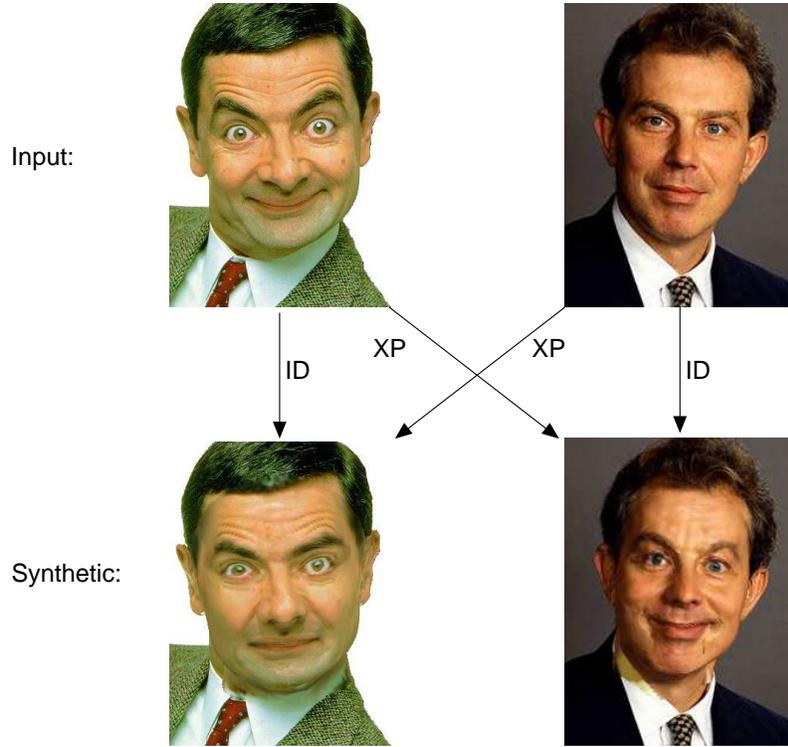


Figure 5.5: Example of facial expression transfer. The expression of Mr Bean is transferred to Tony Blair and vice versa.

identity morphable model, it was decided not to model the inside part of the mouth.

First the neutral scans are put in correspondence with the identity morphable model. The scans with expressions are then put in correspondence with the neutral scan of the same individuals. Hence these expressions scans are also in correspondence with the identity morphable model. As these scans are in correspondence, they are then in a vector space and can be added and subtracted to/from one another. Hence, to obtain the motion of the face points between the neutral and another expression, the neutral scan of one individual is subtracted from an expression scan of the same individual:

$$\mathbf{S}_i^{\Delta,j} = \mathbf{S}_i^j - \mathbf{S}_i^0 \quad (5.3)$$

In this equation, the expression scan  $j$  from the individual  $i$  is subtracted from its neutral expression (indexed by 0) to form the expression deviation, free of its original identity. A PCA model is then constructed from the expression vectors,  $\mathbf{S}_i^{\Delta,j}$ , in a similar way as the identity PCA. The first twelve principal components are shown on Figure 5.7. In fact, this figure displays a set of renderings of a face constructed from the mean texture and the expression shape deviations added to the mean shape.

The shape model is then composed of an identity part and an expression part. A linear combination of the principal components of these models can be made to obtain the shape of the face of any individual with any expression (within the linear span of the model, naturally):

$$\mathbf{S} = \bar{\mathbf{S}} + \sum_{i=1}^{N_S} \alpha_i \cdot \mathbf{S}^i + \sum_{i=1}^{N_E} \epsilon_i \cdot \mathbf{S}^{E,i} \quad (5.4)$$

In this equation a linear combination of the  $N_E$  shape expression principal components,  $\mathbf{S}^{E,i}$ , with the coefficients  $\epsilon_i$  yields the expression deviation. The expression components are not orthogonal to the identity components. Hence, the Equation (5.4) cannot be inverted easily, but this does not have any influence on the fitting algorithm.

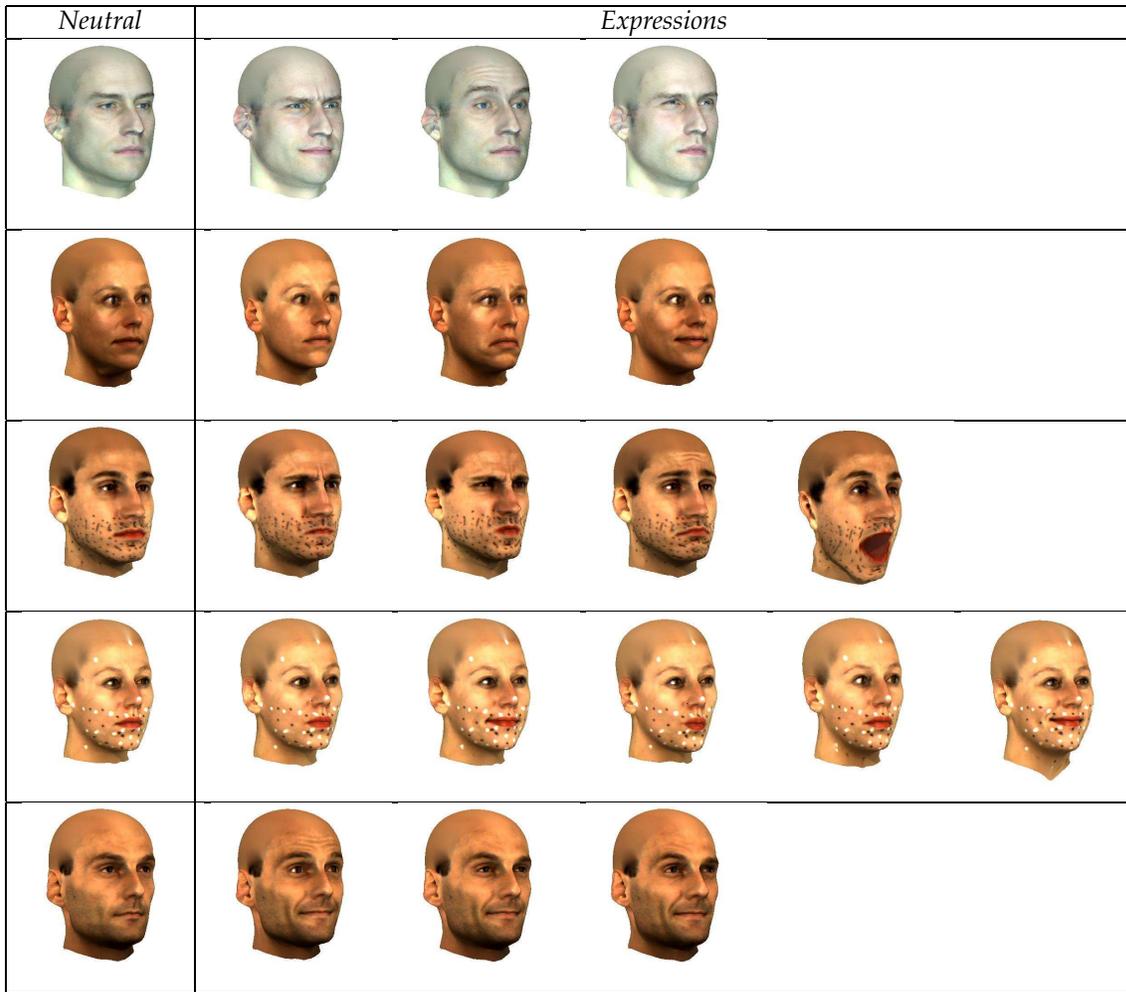


Figure 5.6: Renderings of the exemplars 3D scans used to build the shape expression model. The first column shows the neutral expression scans and the other ones, scans with non-neutral expression. The expression space spans this set of expressions.

The correspondence computation of the expression 3D scans and the construction of the expression morphable model is not part of the material of this thesis. This was achieved by Curzio Basso in our research group. However, what is part of this thesis, is the usage of this model to fit facial photographs with non-neutral expression.

**Fitting of a non-neutral expression face photograph.** The fitting algorithm used to fit faces with expressions is exactly the same as for faces with neutral expression. The only difference is that the shape model is augmented with the expression principal components. In fact, even the fitting algorithms parameters are the same for the identity only fitting and for the identity and expression fitting. This shows the robustness of the fitting algorithm.

The fitting results on the input images of Mr Bean and Tony Blair (see Figure 5.5) are shown on the second column of Figure 5.8. The first column shows the anchor points used for the fitting, and the last column shows a rendering of the fitting results with the expression parameters set to zero:  $\epsilon_i = 0$ . The expression of these images seems rather neutral while the identity seem to be preserved. This shows that the expression variation is only explained by the expression coefficients and the identity by the identity coefficients.

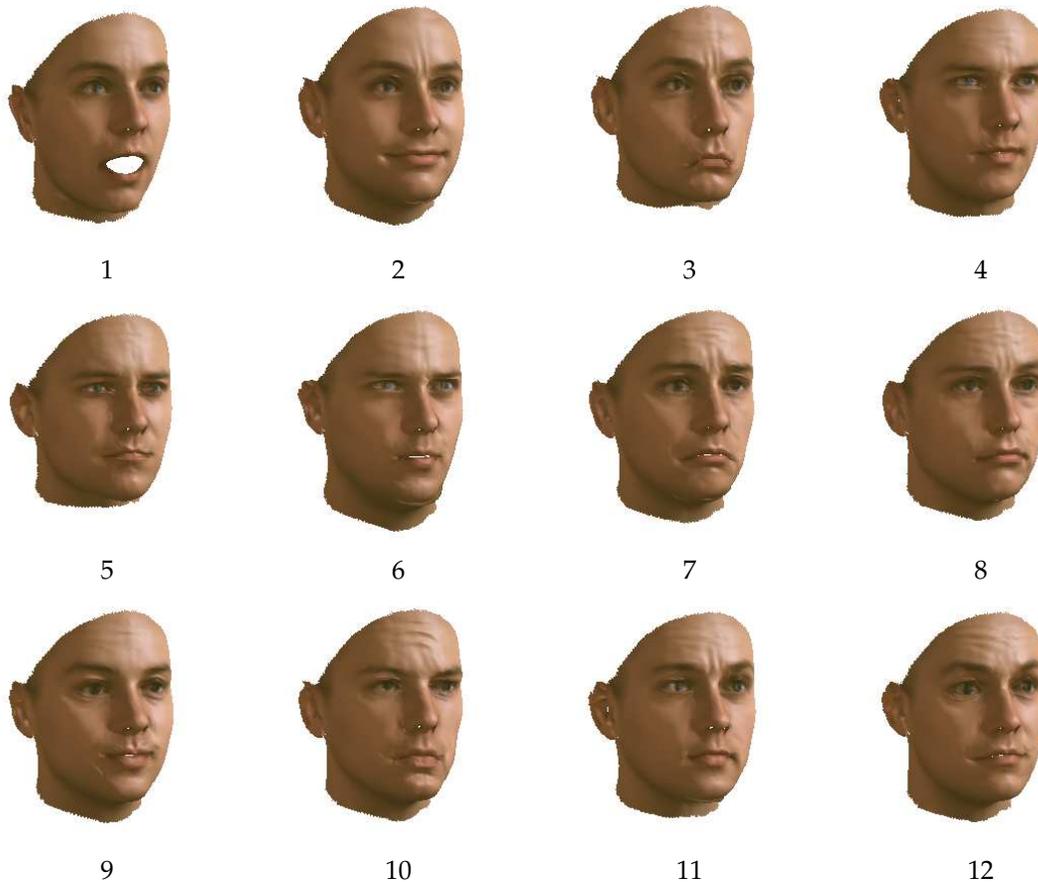


Figure 5.7: Renderings of the first 12 of the 18 shape expression principal components. The expression principal components are added to the mean shape of the identity model. A face is formed from this shape and from the mean texture of the identity model. Renderings of the side view of these faces are displayed.

**Expression transfer.** To transfer the expression from one photograph to the other, the expression parameters are simply exchanged between the fitting results of the two photographs. The result is shown on Figure 5.5.

**Future work.** One problem of the expression model is its inability to model appropriately the forehead wrinkles. Hence, these wrinkles are not recovered by the fitting result of Mr Bean (see Figure 5.8), they are still present on the synthetic image of Mr Bean with the expression of Tony Blair and they are not transferred to Tony Blair (see Figure 5.5). The expression model is, currently, rather sparse, and does not account for the forehead wrinkle. Hence, a better compensation of these wrinkles could be obtained by a more complete expression model.

### 5.3 Face tracking without anchor points

The last application addressed in this thesis is face tracking. Face tracking is about following the motion of the face of an individual through a sequence of images.

In a tracking application, the identity is known to remain constant throughout the sequence. Hence, once the identity is fitted to the first frame, only the pose and the expression parameters need to be subsequently updated (assuming that the illumination within the scene is constant). The idea is, then, that the edges, alone, encode enough information to recover the pose and the

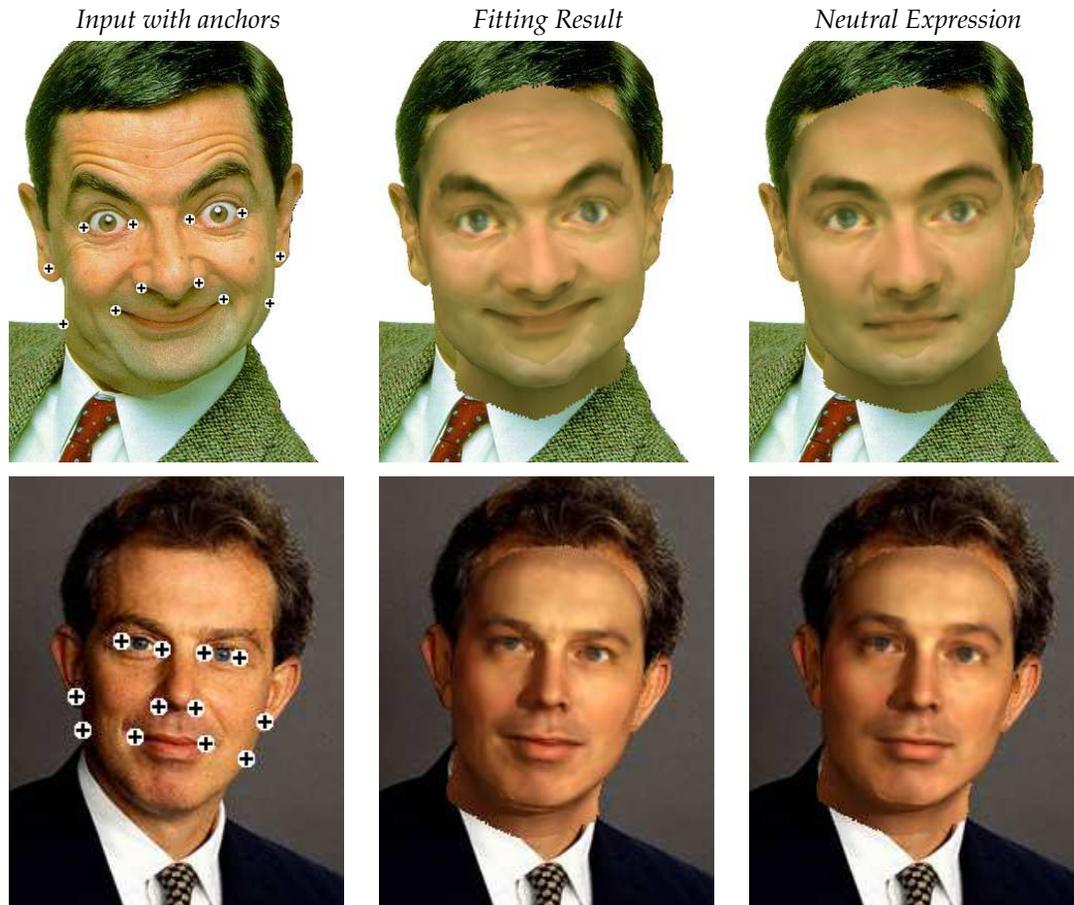


Figure 5.8: Fitting of images with the identity and expression 3DMM. The second column shows renderings of the fitting results obtained on the input images of the first column, which also displays the anchor points used. The third column shows renderings of the fitting results with expression parameters set to zero.

expression of an individual. From a perception point of view, this argument seems legitimate, as a human can easily recognise the pose and the expression from a rough hand-drawn sketch of a face.

The first frame is then fitted as usual, using landmarks and using all the stages of the multi-features fitting algorithm, thereby recovering the shape, the texture, the rigid and the illumination parameters. Then, only the second stage of the fitting algorithm is used to fit the next frames of the sequence. This stage relies only on the edges to fit the textured edges and the contour edges of the model. The only difference with the original second step of the algorithm, is that here, the rigid and the expression parameters, only, are fitted, not the identity parameters.

This results in a relatively fast tracking algorithm. The Matlab implementation of the tracker, on a Intel 3.0GHz Pentium IV computer takes 4.4s to prepare each image (compute the edges and the chamfer distance transforms) and 0.15s for the optimisation.

The tracking is experimented on a sequence of 450 frames acquired at about 30 fps. This is a sequence of a person turning his head and at the same time changing his expression: from neutral to smiling, then he opens the mouth and afterwards starts speaking. The avi file is located in the file named `matlab/tracking/tracking2.avi`. Each fiftieth frame is shown on Figure 5.9.

**Conclusion.** The rigid parameters are always accurately recovered. the expression is also most of the time nicely fitted: the model follows the smile smoothly and open the mouth with the input.

A problem occurs though, when the subject starts speaking after having widely open his mouth. At that point there is a fitting error: the upper edge of the lower lips is fitted to the lower edge of the lower lips (see frame 401). The interior of the mouth is not modelled, hence the upper edge of the lower lips is considered as a contour edge. As we have seen on Section 4.2.2, although an orientation can be computed for contour edges, their sign is arbitrary: depending on the colour of the interior of the mouth (white if the teeth are visible, pink or black if not), the normal is upward or downward. As the direction of the edge cannot be computed, the corresponding edge in the image, could be sometime mistaken with another edge which has the same orientation but which could have another direction. This is what happens between the upper edge of the lips and its lower edge. To improve the result, the interior part of the mouth should be modelled additionally.

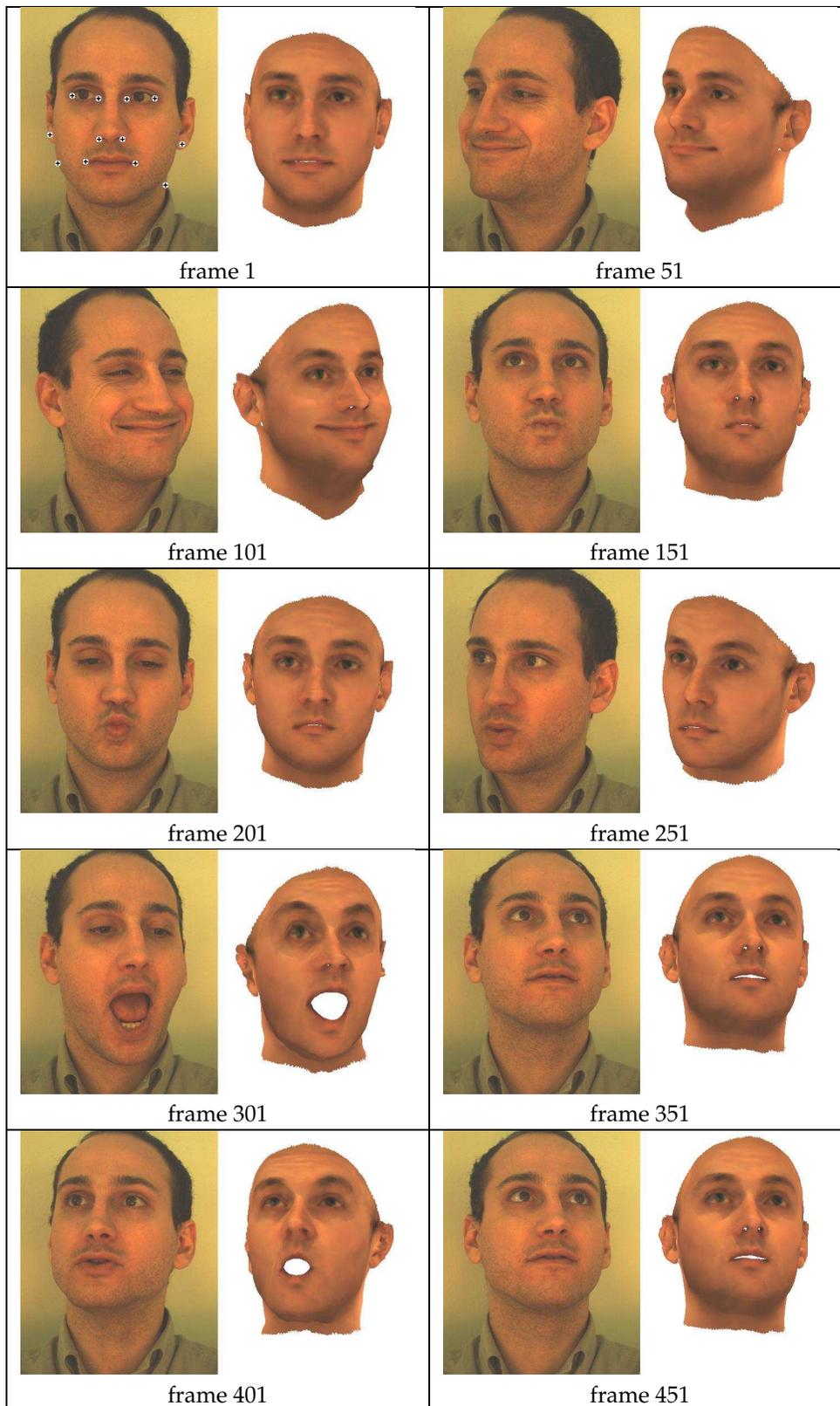


Figure 5.9: Input and fitting result of each fiftieth frame of the tracked sequence.



## Chapter 6

# Conclusions

In this thesis, I presented a novel method able to address several facial image analysis problems. The first application demonstrated was the facial image identification and verification. In an identification experiment, the objective is to find, in a set of images, called the gallery set, a face image of the same person as the test image. In a verification problem, the aim is to determine whether two input facial photographs show the same person. These two applications were set up in their most general form, in which a single image per individual was available in the gallery set and a single test image was used; additionally, no constraint were put on the facial photographs, allowing the pose and the illumination to vary (the only requirement was that the facial expression is neutral). To tackle these problems, one must derive a face signature invariant to illumination variation and rigid transformations. A natural way to reach this goal is to use a 3D representation.

Another application presented in this thesis concerned facial expressions. A facial image analysis system addressing expression variations could have many objectives: A human-computer interaction device would attempt to recognise the expression of a face, regardless of its identity, in order to decipher the emotional state of a user. An identification algorithm would endeavour to recognise a person irrespective of his/her expression. In computer graphics, synthesising realistic facial expressions is a difficult problem. One way of tackling this problem is by mapping the expression of a performer to the face of someone else. This is the second application presented in this work. One obvious requirement for an expression transfer (or mapping) system is to be capable of separating the identity from the expression of a facial image. Then, this system has to apply this expression on another individual's photograph.

The third application developed here is face tracking in a video sequence. The aim is to follow the facial movement induced by rigid transformations and expression variation of someone's face in a series of images acquired at very small intervals in time (such as 25 frames per second). It was demonstrated that edges, alone, formed a good feature to track these facial motions.

All the applications aforementioned necessitate the separation of the different sources of variation of facial images. In an analysis-by-synthesis approach, the first step is to construct a model able to generate facial images in all conditions and in which the sources of variation are factored. In the first two chapters of this thesis, a face model was presented that separates rigid transformation, illumination and identity variations. Rigid transformation and illumination are modelled using physically-based techniques, while identity is modelled by learning the differences between individuals over a training set.

In a 3D framework, the image analysis problem is to estimate the 3D shape of a face from a single input facial image. The only clue available in a single image about depth is contained in the shadings and the shadows. However using this information requires the illumination environment and the texture of the face to be known. As these are not known, the problem is ill-posed. One way out of this dilemma is by using prior knowledge. Therefore, texture and shape PCA models are employed to constraint the set of possible solution. A lighting model is also used, the Phong reflectance model, which has a few parameters when only one light source is handled. However, even applying this prior models is not enough to obtain an accurate estimate of the

3D shape when just a few manually set anchor points are used as input. This is because the cost function to be minimised is highly non-convex and exhibits many local minima. In fact, the shape model requires the correspondences between the input image and the reference frame to be found. Using only facial colour information to recover these correspondences is not optimal and may be trapped in regions that present similar pixel intensity variations (eyes/eyebrows, for instance). Other features could be used to obtain a more accurate estimate of the correspondences and, as a result, of the 3D shape. One example of such feature is formed by the edges. Other features that improve the shape and texture estimate are the specular highlight and the texture constraints. The specular highlight feature uses the specular highlight location, detected on the input image, to refine the normals, and, thereby, the 3D shape of the vertices affected. The texture constraint enforces the estimated texture to lie within a specific range, which improves the illumination estimate.

To avoid getting trapped in local minima, the SNO algorithm used a stochastic optimization that introduced a random error on the derivative of the cost function. This is performed by sampling the cost function at a very small number of points (40 pixels). This random error enables the algorithm to escape from local minima. On the other hand, the MFF algorithm, presented in this paper, has a smoother cost function owing to the various sources of information used. Hence, it is not required to use a stochastic optimization algorithm and the pixel intensity feature may be safely sampled at many more points providing a more stable parameter update. Hence, much fewer iterations are needed to reach convergence (a few tens instead of several thousands in the case of SNO). As a result, as well as being accurate, MFF is also more efficient than SNO.

Because the 3DMM tries to account for all sources of variation and to separate them, we think that it sets the path to a unified approach addressing the full problem of facial image analysis. However, there are still a number of sources of variations that it does not model. Addressing these will not only expand the type of images the model could render and fit, but also, improve the accuracy of the estimation of the 3D shape by the fitting algorithm. This is why, handling the following issues would improve the overall performances of the system:

- *Reflectance and illumination:* Currently, the fitting can recover only a single directed light source using the Phong reflectance model. Several limitations follow, which have an impact on the accuracy of the 3D shape recovery. The Phong reflectance model represents the way a surface point reflects light. Additionally to the diffuse component, it can generate specular highlight and, using the 3D shape, cast shadows. However, it is only an approximation of how the human skin reflects light. As aforementioned, the depth information lies in the shading. As a result, using a reflectance closer to the one of the human skin and an illumination environment closer to the one in which a photographed face is surrounded, will improve the 3D shape recovery. The reflectance of a surface point is given by its Bidirectional Reflectance Distribution Function (BRDF). Marschner *et al.* [51] and Debevec *et al.* [27] show a method to measure and to use the BRDF of human skin. From these measurement a model of the BRDF could be constructed and used in our 3DMM and its fitting algorithm. Using a proper BRDF is only one part of the way. Another part is to use not only one light sources but several of them, such as an environment map, which is used in computer graphics since years. This last modification is, however, not trivial to put in practice as it would increase dramatically the number of parameters to fit.

The cast shadows also reveal some depth information. The manner by which the cast shadow information is taken into account by the pixel colour feature of the fitting algorithm is not optimal. A binary attribute is given to the fitted vertices reflecting their belonging to a cast shadow. If a vertex is in the cast shadow, then there is no contribution of the directed light in the cost function and in its derivatives. This binary attribute is set at the beginning of a fitting stage and kept constant for the entire stage. However, to improve the light direction estimate and the 3D shape recovery, the attribute should be (i) differentiated with respect to the light direction and to the shape parameters, and (ii) updated after each iteration. The problem is that doing so would considerably increase the computational load of the algorithm. Hence, a technique should be introduced to use efficiently the cast shadows.

- *Expression*: The expression model used in the expression transfer application is not representative of all the variations of human faces. There is only one example face with an open mouth. The 'smiling' expression does not cover the whole range of smiles possible. The wrinkles on the forehead are not modeled at all. This is why the wrinkles of Mr. Bean were not fitted and hence could not be transferred to Tony Blair. When the expression model will span more expression variations, then the expression fitting will improve. The interior of the mouth and the teeth are not modeled, either. Hence, it is difficult to fit an input facial photograph with an open mouth. Curzio Basso, a doctorate student in our group, is working on extending the expression model toward these goals.
- *Localised model*: The 3D Morphable Model is a holistic model. The shape and texture parameters influence the whole shape and texture. This is based on the assumption that any vertex of the face is correlated to all the other ones. While there may be some correlation between all the vertices at low spatial frequency, it is quite unlikely to find this type of correlation at high frequency. There is some evidence supporting this assertion. Figure 6.1 shows the decrease of the reconstruction error of the shape and texture as the number of holistic principal components increase. These plots were performed using PCA models trained on the 200 heads scanned at the Max Planck Institute of Tübingen. For testing, the scans provided by the University of South Florida were used. The shape reconstruction error is measured by the Euclidean distance between corresponding vertices of a test scan and its model reconstruction. This error is then averaged over all vertices of one scan and over all testing scans. The shape reconstruction error is expressed in millimetre. For the texture the Euclidean distance between the three colour channels is used as a measure.

Initially, the decrease is rapid. However, as the number of holistic PCA dimensions increases, the influence on the reconstruction error is attenuated. A regime is reached about the hundredth dimension. An exponential line is then fitted. Using this exponential line, we may extrapolate the number of dimensions that would be required to reach a certain reconstruction accuracy. It turns out that 458 shape dimension would be required to obtain 1 mm accuracy; 927 dimensions, for 0.5 mm; and 2017 dimensions, for 0.1 mm accuracy. Concerning the texture, 617 dimensions are necessary to attain 10 grey-levels; 1689 dimensions, for 5 grey-levels; and 4177, to reach 1 grey-level. Observe that the exponential lines, at the 200th dimensions, are below the reconstruction error line. Hence these extrapolated values are rather conservatives.

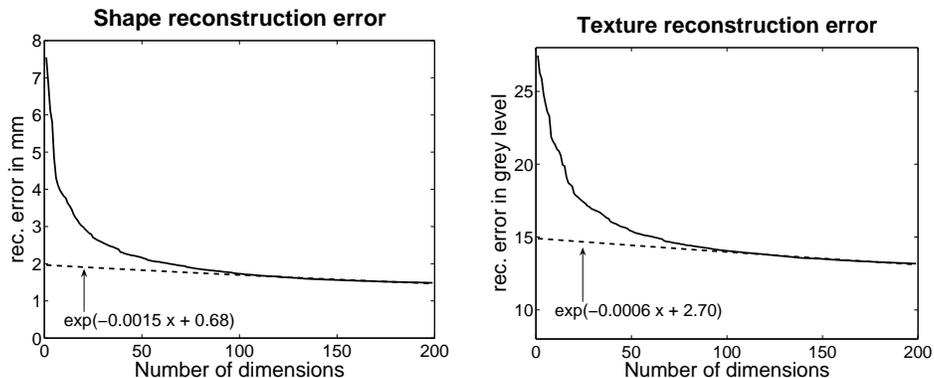


Figure 6.1: Plots of the shape and texture reconstruction error computed on a test set of 3D scans. For each plot an exponential line (dashed) is fitted to the part of the curve that has reached its regime (after the 100th dimension).

The conclusion that can be drawn is that a holistic model may be appropriate to account for the low frequencies, but not to code the higher frequencies. This is why a model localised in the spatial domain and in the frequency domain could yield a more efficient coding of the

facial shape and texture. Reinhard Knothe, a doctorate student in our group is working on the construction of such a localised morphable model.

Addressing these modelling issues will not only refine the generative expressiveness of the model but will also ameliorate the fitting accuracy. There are also other points that would augment the fitting performances:

- *Outlier detection*: The major source of fitting inaccuracies is the presence of outlying pixels in a facial image. An outlying pixel, or outlier, is a pixel inside the face area of an image whose value cannot be predicted by the model nor by the noise distribution assumed by the fitting cost function. Typical examples of outliers are glasses, specular highlight due to the presence of glasses, and occluding objects such as facial hair. Naturally, a black pixel in the eye region due to sun glasses may be predicted by the model, but doing so, would substantially modify the model parameters and deteriorate the fitting of the rest of the face. Hence fitting an outlier is prejudicial not only because the fitted model parameter would account for an artifact such as glasses but also because it would substantially decrease the overall quality of the fitting in the outlier-free region of the face. This problem was raised in Section 4.5 and addressed by the use of a robust cost function such as the Huber function or the Lorentzian function. These estimators are part of the class of estimators called M-estimator [38, 44] and are based on the fact that outliers have a substantial discrepancy and hence pixel with large error should be down-weighted. These function depend on a value,  $\sigma$ , that sets the limit between the inliers and the outliers. This value depends on the image and is difficult to adjust: For instance, the value of  $\sigma$  would be different for a bright image than for a darker image. Another drawback of the M-estimator is that some inliers may have a large fitting error and discarding them would jeopardise the quality of the fitting. This may happen at the beginning of the fitting algorithm when there is an important lack of correspondence. For instance, the model pupil may overlap the skinny area between the eye and the eyebrow inducing a large residual. It is this residual which is to drive the model parameters to improve the correspondence and, hence, it should not be down-weighted. Therefore, a better scheme should be used to detect outliers, based not only on the residual but also on the Jacobi matrix.

One strategy towards this goal might be the following: In a local optimisation technique, each pixel used in the fitting error gives an update direction of the model parameter [33]. The update direction obtained at one iteration is the sum of the update directions of all pixels used. If all these directions are parallel or nearly parallel, then all pixels are in agreement about the direction to take to reduce the cost function. In this case, all pixels are inliers. However, if there are pixels giving rise to directions deviating from the direction of the majority of pixels, then these are outliers. an algorithm detecting these deviations is not trivial to set up. Jean-Sebastian Pierrard, a doctorate student of our group, is working on these issues.

- *Automatic anchor points localisation*: A drawback of the fitting algorithm presented here is that it is not automatic. A user must click on a few anchor points to initialise the rigid parameters of the model. These points could be automatically detected using detection algorithm such as the one of Heisele *et al.* [42] or Burl *et al.* [16]. Then, potentially, more points could be used to constraint the algorithm. For instance, there could be a detector of an open/closed mouth that would initialise the expression model to an open or closed mouth. Additionally, these algorithms would not only yield a localisation for anchor points but also a likelihood that would be used in a probabilistic fitting algorithm to increase its robustness.

# Appendix A

## Derivative

### A.1 Derivative of the Specular highlight cost function

In this appendix, the calculus of the derivative of the specular highlight cost function is detailed. Recall from Equation 4.33 on page 54, that the signed cost of fitting one specular point is:

$$\mathbf{e}^s = \mathbf{n}^{t,w}(\boldsymbol{\alpha}) - \mathbf{n}^s(\boldsymbol{\alpha}) \quad (\text{A.1})$$

As explained in the Section 4.2.4 on page 51, only the shape parameters are updated as this feature is used at the end of the fitting to refine the shape. Therefore, in the cost function, only the dependence on the shape parameter is mentioned. The normal of a triangle in world coordinate,  $\mathbf{n}^{t,w} = \mathbf{R}_\gamma \mathbf{R}_\theta \mathbf{R}_\phi \mathbf{n}^t$ , where  $\mathbf{n}^t$  is the normal in object coordinate expressed in Equation 2.19 on page 23, is rewritten as follows:

$$\mathbf{n}^{t,w} = \sqrt{L_n} \begin{pmatrix} n_x \\ n_y \\ n_z \end{pmatrix} \quad (\text{A.2})$$

with

$$L_n = (n_x^2 + n_y^2 + n_z^2)^{-1} \quad \text{and} \quad \begin{pmatrix} n_x & n_y & n_z \end{pmatrix}^T = (\mathbf{s}_1 - \mathbf{s}_2) \times (\mathbf{s}_1 - \mathbf{s}_3) \quad (\text{A.3})$$

where  $\mathbf{s}_1$ ,  $\mathbf{s}_2$ , and  $\mathbf{s}_3$  are the 3D coordinates of the three corners of the triangle, in world coordinates, that depend linearly on the shape parameters  $\boldsymbol{\alpha}$ . The derivative of the normal is then:

$$\frac{\partial \mathbf{n}^{t,w}}{\partial \boldsymbol{\alpha}} = \sqrt{L_n} \begin{pmatrix} \partial n_x / \partial \boldsymbol{\alpha} \\ \partial n_y / \partial \boldsymbol{\alpha} \\ \partial n_z / \partial \boldsymbol{\alpha} \end{pmatrix} - L_n \cdot \left( n_x \frac{\partial n_x}{\partial \boldsymbol{\alpha}} + n_y \frac{\partial n_y}{\partial \boldsymbol{\alpha}} + n_z \frac{\partial n_z}{\partial \boldsymbol{\alpha}} \right) \cdot \mathbf{n}^{t,w} \quad (\text{A.4})$$

To differentiate the cost function, the derivative of the normal of the points at which the specular highlight occur is also required. This normal is expressed in Equation 4.32 on page 53.

$$\frac{\partial \mathbf{n}^s}{\partial \boldsymbol{\alpha}} = \frac{\partial \mathbf{v}}{\partial \boldsymbol{\alpha}} \cdot L_{n^s} - (\mathbf{v} + \mathbf{d}) \cdot L_{n^s}^3 \cdot \left( (v_x + d_x) \frac{\partial v_x}{\partial \boldsymbol{\alpha}} + (v_y + d_y) \frac{\partial v_y}{\partial \boldsymbol{\alpha}} + (v_z + d_z) \frac{\partial v_z}{\partial \boldsymbol{\alpha}} \right) \quad (\text{A.5})$$

with

$$L_{n^s} = \left( (v_x + d_x)^2 + (v_y + d_y)^2 + (v_z + d_z)^2 \right)^{-1/2} \quad (\text{A.6})$$

The viewing direction  $\mathbf{v}$  depends on the world coordinates:

$$\mathbf{v} = \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix} = -\sqrt{L_v} \begin{pmatrix} w_x \\ w_y \\ w_z \end{pmatrix}, \quad \text{with} \quad L_v = (w_x^2 + w_y^2 + w_z^2)^{-1} \quad (\text{A.7})$$

its derivative is then:

$$\frac{\partial \mathbf{v}}{\partial \boldsymbol{\alpha}} = -\sqrt{L_v} \begin{pmatrix} \partial w_x / \partial \boldsymbol{\alpha} \\ \partial w_y / \partial \boldsymbol{\alpha} \\ \partial w_z / \partial \boldsymbol{\alpha} \end{pmatrix} + L_v \cdot \left( w_x \frac{\partial w_x}{\partial \boldsymbol{\alpha}} + w_y \frac{\partial w_y}{\partial \boldsymbol{\alpha}} + w_z \frac{\partial w_z}{\partial \boldsymbol{\alpha}} \right) \cdot \mathbf{v} \quad (\text{A.8})$$

Now the derivative of the specular highlight cost function for one vertex is function of the derivative of the unnormalized normal elements  $n_x$ ,  $n_y$ , and  $n_z$ , and of the derivatives of the world coordinates, which can easily be computed.

# Appendix B

## Matlab Morphable Model Toolbox

As part of this doctoral work, a Matlab toolbox was implemented, called the Morphable Model Toolbox (MMT). Its main features are programs able to generate 3D faces from model parameters, render 3D faces, fit 3D face from an input image using the Multi-Features Fitting algorithm and perform identification. This appendix is a user manual of the MMT.

All the figures and the tables of this thesis are entirely reproducible using the MMT. The next appendix details the procedure required to reproduce them.

At the time of writing this thesis, I am not allowed to make the Morphable Model Toolbox public for copyright reasons. I hope that this will change in the future.

**System requirements** The MMT is stand-alone. It requires Matlab version 6.5 or higher, the Matlab Optimization Toolbox and the Matlab Image Processing Toolbox. With Matlab version 6.5, the MMT is required to be run on a computer with 1Gb of RAM. With Matlab 7.0, or higher, 500Mb of RAM is necessary.

### B.1 Installation

**Path installation** It is assumed that the Morphable Model Toolbox is extracted into the directory `MMT_DIR`. Before each use of the MMT, the file `mmt_install_path.m` must be executed. This file, first sets the `MMT_DIR` as a global variable then registers all the subdirectories of the MMT with matlab. As this file is to be called before each use of the MMT, it is recommended to put the following two lines in the `startup.m` file, located in the `$HOME/matlab` directory.

```
addpath ~/matlab/MMT
mmt_install_path
```

In this example, the MMT is in the directory `MMT` located in the Matlab directory of your home directory.

**MEX Files** The MMT includes some functions implemented in C++ for efficiency. These functions use loops intensively, such as the rendering function for instance that loops over the pixels. They are coded in standard C++ and do not use any external library. Hence they are stand alone and should be compilable on any machine running Matlab. `mmt_install_mex.m` is a file that produce all the MEX files of the MMT. It must be executed the first time the MMT is used. If the MEX compiler is properly installed in the Matlab distribution, the MEX files of the MMT should be generated without problem. Refer to the Matlab manual for more information about the MEX Compiler.

## B.2 Morphable Model Toolbox Tour

In this section, some example of usage of the Morphable Model Toolbox are presented.

**Example of generating a 3D head and rendering it.** The `script` directory includes two examples. The first example script, `script_rendering.m` shows how to generate a random 3D head from the model and how to render it. It first load the model, its triangle list, and the data used to blend the segments of a segmented face (Section 2.1.5 on page 20):

```
% load the model
load MPI_99.mat
% load the triangle list and the mask
load topo_5.mat
% load the data needed for the blending of the 4 segments
load segment_5.mat
```

The model loaded includes 99 shape and texture dimensions. Then, random shape and texture coefficients are generated by sampling from a Gaussian distribution with zero mean and identity covariance matrix.

```
% generate random shape and texture parameters
alpha = randn(99, 1);
beta = randn(99, 1);
```

Rendering parameters are loaded from the file `rp.mat` in the directory `data/param`.

```
load rp.mat           % load a few realistic rendering parameters
rp = rp_garden;      % use lighting parameters estimated from a
                    % garden photograph
rp.phi=0.6;         % phi is the azimuth rotation angle
rp.width = 256;     % output image size
rp.height = 256;
```

Then the face is generated, the image rendered and displayed.

```
img = model2img(alpha, beta, rp, model, t1);
figure(1)
imshow(img)
```

A segmented 3D face (Section 2.1.5 on page 20) is generated and rendered with the following code:

```
% generate random shape and texture parameters
% for the four segments
alpha = randn(99, 4);
beta = randn(99, 4);

% render a segmented head
img = model2img(alpha, beta, rp, model, t1, ...
                [], [], [], segment);
figure(2)
imshow(img);
```

The process of 3D face generation and rendering can be decoupled. First the 3D shape and the texture are computed.

```
shape = smahal2shape(alpha, model, segment);
tex = tmahal2tex(beta, model, segment);
```

Then the 3D face can be rendered.

```
img = shatex2img(shape, tex, rp, tl);
figure(3)
imshow(img);
```

**Example of Face Fitting** The code of this example is located in the file `script_fitting.m` in the directory `script`. First some global variables are set: `MMT_DIR` is the root directory of the Morphable Model Toolbox that should be defined in the `startup.m` file. The variable `SHOW_SAMPLING_POINTS` displays, at each iteration of the fitting, the projection of the model points on which the pixel error function is computed. The variable `FEATURE_SIGNIFICANCE` shows at the end of each stage, the importance of each feature in determining a parameter update at each iteration. Setting these variables enable or disable the corresponding visualisation feature.

```
global MMT_DIR
global SHOW_SAMPLING_POINTS
SHOW_SAMPLING_POINTS=0;
global FEATURE_SIGNIFICANCE
FEATURE_SIGNIFICANCE=1;
```

The fitting data, the image and anchor files are loaded.

```
% load the default fitting parameters
fit_data = get_fit_data;

img_fn    = fullfile(MMT_DIR, 'data', 'img', 'Hee.png');
anchor_fn = fullfile(MMT_DIR, 'data', 'img', 'Hee.anchors');
in_img    = double(imread(img_fn));
anchor    = read_anchor(anchor_fn);
```

Some optional parameters are set in the structure `opt`. The field `quality` sets the number of stages to use, and the field `verbose` sets the level of information output during fitting. This field takes a character string. The string `'verbose'` prints the fitting error at each iteration, the string `'brief'`, prints the fitting error after each fitting stage, and the string `'silent'` would not output anything.

```
% fitting will be verbose and run to the highest quality
opt.quality = length(fit_data.fp);
opt.verbose = 'verbose';
```

Then the Multi-Features Fitting algorithm can be launched.

```
fres = fit_image(in_img, anchor, fit_data, opt);
```

At the end of the fitting, a figure, similar to the Figure 4.23 on page 61, showing the fitting result at the end of each stage is displayed. Once the fitting is done, the resulting 3D face is computed.

```
shape = smahal2shape(fres.alpha, fit_data.model, ...
                    fit_data.segment);
tex    = tmahal2tex (fres.beta, fit_data.model, ...
                    fit_data.segment);
```

Then a rendering of the fitted image overlapping the input image is computed and displayed.

```
img = shatex2img(shape, tex, fres.rp, fit_data.tl, in_img);
figure(3)
imshow(img);
```

The texture of the input image may be extracted and the illumination inverted. The extracted texture may then be use for rendering. This should produce an image very close to the input image.

```
% extract the texture from the input image
ext_tex = inv_ext_tex(in_img, shape, tex, fres.rp, ...
                    fit_data.tl);

figure(4)
imshow(tex2img(ext_tex(:)/255, fit_data.mask, 'interleaved'));

% make a rendering using the extracted texture
figure(5)
imshow(shatex2img(shape, ext_tex, fres.rp, ...
                 fit_data.tl, in_img));
```

A Wavefront OBJ 3D file may then be produced to export the 3D head to any Computer Graphics software package.

```
objwrite(fullfile(MMT_DIR, 'script', 'fres_Hee'), shape, ...
         ext_tex(:), fit_data.tl, fit_data.mask);
```

**Directory Structure** The .m files are located in subdirectories of the MMT\_DIR directory. The list of subdirectories follows.

**data** The data directory contains the .mat files that hold the necessary data for the MMT. The files are arranged in subdirectories of the data directory depending on the part of the MMT that the files are used for. The param directory includes the file rp.mat that stores a few rendering parameters with realistic illumination configurations. The directory model includes the files that store the morphable model, i.e., the shape and texture PCA models and topological information such as the triangle list. The directory fit\_data includes fitting parameters and other data used for the fitting. The directory img includes a few images and their anchor files used as example to illustrate the fitting.

**doc** The doc directory includes some files that provides documentation about the MMT.

**fitting** The directory fitting includes all the .m files necessary for the implementation of the MFF algorithm.

**html** This directory provides documentation of the MMT in HTML format. The file index.html is the main documentation file. The whole documentation tree is produced by the function mmt\_html\_help.m located in the MMT\_DIR directory. It is based on the package m2html available at <http://www.artefact.tk/software/matlab/m2html/>.

**id** The Matlab functions located in the directory id are used to do identifications and verifications experiments after fitting a series of images. It contain specific files for making directly statistics on the PIE and FERET face image databases.

**image\_proc** This directory includes a few files performing image processing operations needed by the MFF algorithm.

**interactive** There are two GUI interactive programs implemented in the MMT. view\_shatex.m uses the Matlab patch function that renders a 3D textured mesh using OpenGL and allows its 3D rotation interactively. The second program, place\_anchors.m is a tool to easily place anchor points on an input image and return an anchor point structure that can then be used for the fitting algorithm.

**mex** The mex files are located in the mex directory. The C++ source files are in its subdirectories include and src.

**misc** The directory `misc` includes several miscellaneous functions used in the toolbox.

**rendering** All the function implementing the face generation and the rendering, described in Chapter 2 on page 15, are located in the `rendering` directory.

**script** Several examples showing how to use some functionalities of the toolbox are located in the `script` directory.

## B.3 Documentation

If the Morphable Model Toolbox is installed (Section B.1) in the directory `mmt` on the Matlab path, issuing the following command at the Matlab prompt lists the MMT functions and a one line document describing them.

```
help mmt
```

Typing `help` followed by any function name provides a more detailed description of the function.

## B.4 3D Face Generation and Rendering

This section gives an overview of the functions available in the directory `rendering` for generating 3D heads from model parameters and rendering them.

### High level 3D face generation functions

**smahal2shape** Compute the shape from the morphable model shape parameters. This function implements the Equation 2.10 on page 20. If the input coefficients include four segments, then this equation is computed four times (one for each segment) and the segments are blended using the functions `tex2img`, `blendImage`, and `img2tex`.

**tmahal2tex** Compute the texture from the morphable model texture parameters. Performs intrinsically the same thing as `smahal2shape`, but for the texture.

### High level rendering functions

**model2img** Render an image from its morphable model and rendering parameters. Calls successively the functions `smahal2shape`, `tmahal2tex`, `shape2screen`, `rasterize`, `normal.vertex`, `illumtex` and `render` (or `render_gray`, if the image is a grey-level image).

**shatex2img** Render a shape and a texture to an image. This function performs a subset of the previous function. The only difference is that the functions `smahal2shape` and `tmahal2tex` are not called, as the shape and texture are directly given as input to the function.

**defrp** Return default rendering parameters. The rendering parameters structure is detailed in Section B.6 on page 92.

**Low level rendering functions** Equation 2.12 on page 21 that is performing the rigid transformation, is a simpler version of the one implemented by the MMT. This simple form of this Equation is used in Chapter 2, because it uses only the parameters tuned during fitting. However, to allow richer transformation other parameters are used for the rendering. The projection equation implemented follows:

$$\mathbf{W} = s \cdot \mathbf{R} \cdot \mathbf{S} + (\mathbf{R} \cdot \mathbf{t}_o + \mathbf{t}_w - \mathbf{t}_c) \mathbf{1}_{1 \times N_v} \quad (\text{B.1})$$

Where  $s$  is a 3D scaling factor,  $\mathbf{R}$  is the full 3D rotation matrix equal to  $\mathbf{R}_\gamma \mathbf{R}_\theta \mathbf{R}_\phi \mathbf{R}_\alpha$ ,  $\mathbf{t}_o$  is a 3D object translation, and  $\mathbf{t}_c$  is the 3D camera position. Translating by  $\mathbf{t}_c$  transform the coordinates from an object centred representation to world coordinates. The resulting translation is multiplied by a matrix full of ones such as it is applied to all the vertices of the head. During fitting  $s$ ,  $\mathbf{t}_o$  and  $\mathbf{t}_c$  are kept constant.

**proj\_param** Return projection parameters given rendering parameters. This function calls the function `rotmat` that computed the rotation matrix. Then it scales the matrix by  $s$ . It also compute the translation resulting from the addition of all the translation vectors.

**rotmat** Compute the multiplication of the four rotation matrices,  $\mathbf{R}_\gamma \cdot \mathbf{R}_\theta \cdot \mathbf{R}_\phi \cdot \mathbf{R}_\alpha$ , given the 4 angles.

**smahal2screen** Project to 2D screen coordinates a shape given by its parameters. This function is a shortcut calling `smahal2shape` and `shape2screen`.

**shape2screen** Project a shape from 3D object coordinates to 2D image frame. This function implements the Equation (B.1) and the Equation 2.13 on page 21.

**shape2world** Transform a shape from object coordinates to world coordinates. This function implements a subset of the previous function: It computes the world coordinates given by Equation (B.1).

**world2screen** Compute the 2D screen coordinates projected from 3D world coordinates. This function implements Equation 2.13 on page 21.

**normal\_vertex** Compute the normals of the vertices. Depending on the input being the model vertices in object coordinate,  $\mathbf{S}$ , or in world coordinate,  $\mathbf{W}$ , the normals in object coordinate  $\mathbf{n}^v$ , or in world coordinates  $\mathbf{n}^{v,w}$  are returned. Equation 2.18 on page 23 and Equation 2.19 on page 23 are implemented in this function. The normal computation is performed by a C++ function, `normal_vertex`, implemented in the file `mex/include/normals.h`.

**illumtex** Illuminate a texture vector using the Phong reflectance model. To compute the cast shadows, the functions `comp_shadow` and `isshadow` are used. Equations (2.15, 2.20, 2.21, 2.22, and 2.23 on page 24) are implemented in this function.

**comp\_shadow** Compute the cast shadow buffer for every light sources. This function computes a rotation matrix and a translation that transform the Z-direction of the world coordinate system to the lighting direction and then calls the function `comp_sbuffer` that performs a Z-buffer computation.

**comp\_sbuffer** Compute the cast shadow buffer. The buffer computation is performed by a C++ function, `comp_sbuffer`, implemented in the file `mex/include/isshadow.h`. Actually, this function is very similar to the Z-buffer computation function.

**isshadow** Compute if a vertex is in a cast shadow based on the cast shadow buffer and the vertices in light coordinates. This function is implemented in C++ in the function `isshadow` coded in the file `mex/include/isshadow.h`.

**rasterize** Compute for each pixel of an image which triangle projects to it and the barycentric coordinates of the pixel in the triangle. This is a C++ functions implemented in `mex/src/rasterize_matlab_driver.cpp` and in `mex/include/rasterize.h`.

**render** Draw a pre-rasterized object into an image. This function implements the Equation 2.24 on page 24. Its code is located in `mex/src/normal_render_mex.cpp` and in `mex/include/normal_render.h`.

**Segment blending functions**

**blendimage** Blends several parts of an image in one image. This C++ is implemented in the file `mex/include/blend.h`.

**pyrgauss** Compute a gaussian pyramid of a set of images. This C++ is implemented in the file `mex/include/blend.h`.

**Visualisation functions**

**img2tex** Converts a texture map image into a texture vector. This function convert a texture map represented as an image in the reference frame  $(u, v)$  shown in Figure 2.1 on page 19 to the vectorised representation,  $T$ , of Equation 2.6 on page 19.

**tex2img** Converts a texture vector into an texture map image. This function is the inverse of the transformation carried out by the previous function.

**render\_contour** Overlay a face contour on an image. This function applies the algorithm described in Section 4.2.2 on page 46 to find the contour of a 3D object and then draws them in an image.

**normalize** Rescale a n-dimensional data to ease its display. This function is usually used before displaying an image.

**objwrite** Save a head to a Wavefront OBJ file in order to export it to another computer graphics modelling software package.

**B.5 3D Face Fitting - An Implementation of MFF**

The multi-features fitting algorithm software is an algorithm made for researchers to experiment new features and new cost functions. Therefore, one of its key feature is its maintainability and re-usability. To achieve this goal, the code is separated into five layers, listed on Table B.1. Each function of one layer calls a function of the next layer.

Layer	Functions	
1	<code>fit_image</code> , <code>fit_fres</code> , <code>fit_feret</code> , and <code>fit_pie_lights</code>	User level.
2	<code>mff_fit</code>	Main fitting function.
3	<code>lsqnonlin</code>	Matlab nonlinear least square minimisation.
4	<code>cost_full</code>	Main cost function.
5	<code>cost_anchor</code> , <code>cost_lmcp_norm</code> , <code>cost_tex</code> , <code>cost_spec</code> , <code>cost_map</code> , and <code>cost_tex_constr</code>	Feature cost functions.

Table B.1: The five layers of the implementation of the MFF algorithm in the Morphable Model Toolbox.

**Layer 1 - User level functions** The first layer is the user layer. A fitting is initiated from these first layer functions. They are all prefixed by `fit_`. These function vary with respect to the type of input they accept. They all output a fitting result structure. `fit_image` accepts an image, a list of anchor points, and a fitting parameter structure. `fit_fres` is used to continue a fitting from a fitting result structure. This function is useful to test a specific stage: the tester does not have to restart the whole fitting for each test, but may resume it at a specific stage. `fit_feret` and `fit_pie_light` are used to fit the FERET and PIE face image databases.

**Layer 2, 3 and 4 - Main fitting and cost functions** All the user level functions call the function `mff_fit`, which is the main fitting function. The task of this function is to loop on the fitting stages and on the segments to be fitted at each stage. The number of stages is the number of elements in the fitting parameters structure (Section B.6). If one stage is a segmented fitting, an inner loop on the segments is also performed. For each segment of each stage, three tasks are accomplished: (i) `mff_oi` is called. This function aims at collecting the necessary data for each feature that is fitted at one stage. It takes an element of the fitting parameter structure as input and returns a cell of optimisation parameters (Section B.6). (ii) The Matlab Optimization Toolbox, `lsqnonlin` is called. This function performs a nonlinear least squares fitting using an implementation of the Levenberg-Marquardt algorithm[33]. The cost function minimised is `cost_full`, with, as parameter, a cell of optimisation parameters. In this cell, there is one element per feature to fit at the current stage. The function `cost_full` has two tasks. First, it calls the features cost functions (see next layer) and assemble them into a single residual vector and Jacobi matrix. Second, it transforms each feature cost function and Jacobi matrix such as an M-estimator (Section 4.5 on page 57) instead of a least square estimator is used by the `lsqnonlin` function. (iii) The function `mff_fit` also prints some information about the fitting process and may renders some image to visualise the fitting result.

**Layer 5 - Features cost functions** Each function of this layer implements a feature cost function and computes its Jacobi matrix. These functions are called by `cost_full`. Table B.2 relates the MMT functions to the cost function that it computes.

<code>cost_anchor</code>	Anchor points feature (Section 4.2.3 on page 50)
<code>cost_lmipc_norm</code>	Oriented edge feature (Section 4.2.2 on page 43)
<code>cost_tex</code>	Pixel color feature (Section 4.2.1 on page 40)
<code>cost_spec</code>	Specular highlight feature (Section 4.2.4 on page 51)
<code>cost_map</code>	Gaussian prior probability features (Section 4.3.1 on page 54)
<code>cost_tex_constr</code>	Texture constraints feature (Section 4.3.2 on page 55)

Table B.2: List of MMT cost functions and their related feature.

## B.6 Data Structures

This section provides some information on the data structures used in the MMT.

A 3D shape and an RGB texture including  $N_v$  vertices may be represented by a  $3 \times N_v$  matrix or by a  $3N_v \times 1$  vector. The former representation is called the *channelised* format and the later the *interleaved* format.

### B.6.1 Morphable model

The morphable model data is stored in a structure called `model` that includes the following fields.

**sha\_mean** Mean shape in interleaved format.

**sha\_evect** Shape principal components in the interleaved format. For  $N_S$  shape components, it is stored in a  $3N_v \times N_S$  matrix. The column  $i$  stores  $\text{vec}(\mathbf{S}^i)$  (see Equation 2.10 on page 20).

**sha\_eval**  $N_S \times 1$  vector for which element  $i$  is  $\sigma_{S_i}$  (see Equation 2.11 on page 20).

**tex\_mean, tex\_evect, tex\_eval** Same as the above but for the textures instead of the shapes.

**n\_sha\_dim** Number of shape principal components, i.e.  $N_S$ .

**n\_xp\_sha\_dim** If the shape model contains some expression variation, then `n_xp_sha_dim` is the number of components of `sha_evect` coding the expression variations. Otherwise, `n_xp_sha_dim` must be null. Note that these expression components must be the last columns of `sha_evect`.

## B.6.2 Rendering parameters

The rendering parameters, denoted in this thesis by  $\gamma$ , encode the rigid parameters,  $\rho$ , and the illumination parameters,  $\iota$ . The rendering parameters are used to render an image of a 3D mesh (a 3D mesh is defined by a set of  $N_v$  vertices and a triangle list). Default rendering parameters are returned by the function `defrp`. Rendering parameters are stored in a Matlab structure that is usually denoted by the variable `rp` and that contain the following fields:

<i>name</i>	<i>type</i>	<i>default value</i>	<i>description</i>
<code>width</code>	scalar	640	Width of the rendered image.
<code>height</code>	scalar	486	Its height.
<code>gamma</code>	scalar	0	Rotation angle about the Z-axis.
<code>theta</code>	scalar	0	Rotation angle about the X-axis.
<code>phi</code>	scalar	0	Rotation angle about the Y-axis.
<code>alpha</code>	scalar	0	Rotation angle about the X-axis.
<code>t2d</code>	$2 \times 1$	(0, 0)	2D translation, denoted by $t_x$ and $t_y$ in Equation 2.13 on page 21.
<code>camera_pos</code>	$3 \times 1$	(0, 0, 3400)	3D camera position in world coordinates in millimetres, denoted by $\mathbf{t}_c$ in Equation (B.1).
<code>scale_scene</code>			Not used anymore.
<code>object_size</code>			Not used anymore.
<code>shift_object</code>	$3 \times 1$	(0, 0, -46125)	3D translation performed before the rotation. The value is in micrometres and it is denoted by $\mathbf{t}_o$ in Equation (B.1).
<code>shift_world</code>	$3 \times 1$	(0, 0, 0)	3D Translation of the world coordinates origin in millimetres, denoted by $\mathbf{t}_w$ in Equation (B.1). When fitting using a perspective projection then the Z value of this parameter is fitted. In the case of an orthographic projection, it is not fitted.
<code>scale</code>	scalar	$10^{-3}$	3D Scale denoted by $s$ in Equation (B.1).
<code>ac_g</code>	$3 \times 1$	(1, 1, 1)	Colour gain denoted by $[g_r, g_g, g_b]$ in Equation 2.23 on page 24.
<code>ac_c</code>	scalar	1	Colour contrast denoted by $c$ in Equation 2.23 on page 24.
<code>ac_o</code>	$3 \times 1$	(0, 0, 0)	Colour offset denoted by $\mathbf{o}$ in Equation 2.22 on page 24.
<code>ambient_col</code>	$3 \times 1$	(0.6, 0.6, 0.6)	Ambient light colour denoted by $[L_r^a, L_g^a, L_b^a]$ in Equation 2.15 on page 23.
<code>rotm</code>	$3 \times 3$	<b>I</b>	3D Rotation matrix denoted by $\mathbf{R}$ in Equation (B.1).
<code>use_rotm</code>	scalar	0	If this parameter is zero, the rotation matrix used is the one obtained by the multiplication of the four rotation angles <code>gamma</code> , <code>theta</code> , <code>phi</code> , and <code>alpha</code> . If it is zero, then <code>rotm</code> is used instead.

do_remap	scalar	0	If zero, the colour transformation described by Equations 2.22 and 2.23 on page 24 is not performed. In this case $t_i^c = t_i^l$ .
dir_light	struct.	empty	Structure storing the light direction and its intensity.
do_specular	scalar	1	If zero, does not render the specular highlight.
phong_exp	scalar	8	Phong exponent of the specular highlight denoted by $\nu$ in Equation 2.15 on page 23.
specular	scalar	25.5	Strength of the specular highlight denoted by $k_s$ in Equation 2.15 on page 23.
do_cast_shadows	scalar	1	If zero, no cast shadows are rendered.
sbufsize	scalar	200	Half of the size of the cast shadow buffer.
proj	string	perspective	Select the type of 2D projection. May be 'perspective' or 'orthographic'.
n_chan	scalar	3	Specify the number of colour channel. Must be 3 for colour images and 1 for grey-scale images.
illum_method	string	'phong'	Type of illumination method. Only 'phong' is supported for the moment.

The field `dir_light` is a structure with the following two fields:

<i>name</i>	<i>type</i>	<i>description</i>
dir	$3 \times 1$	Light direction in Euclidean coordinates.
intens	$3 \times 1$	Colour intensity of the directed light.

The following parameters are estimated during fitting: `t2d`, `f`, `gamma`, `theta`, `phi`, `ac_g`, `ac_c`, `ac_o`, `ambient_col`, and both fields of `dir_light`. If the parameter `proj` is set to 'perspective', then the Z component of `shift_world` is also estimated.

### B.6.3 Fitting parameters

**Top-level fitting parameters** The parameters controlling the fitting are stored in a structure referenced by `fp`. Default fitting parameters are returned by the function `get_fit_data`. The fitting parameters structure includes the following fields:

<i>name</i>	<i>type</i>	<i>description</i>
match_shape	scalar	If zero, the shape parameters, $\alpha$ , are not fitted.
match_tex	scalar	If zero, the texture parameters, $\beta$ , are not fitted.
match_pose	scalar	If zero, the rigid and projection parameters, $\rho$ , are not fitted.
match_illum	scalar	If zero, the illumination parameters, $\iota$ , are not fitted.
match_xp	scalar	If zero, the expression shape parameters denoted by $\epsilon_i$ in Equation 5.4 on page 72, are not fitted. The current release of the Morphable Model Toolbox does not include an expression model, hence this parameter may not be set to one.

do_inner	scalar	If one, the textured edge feature is used.
do_ctr	scalar	If one, the contour edge feature is used.
do_tex	scalar	If one, the pixel colour feature is used.
do_map	scalar	If one, the prior probability feature is used.
do_anchor	scalar	If one, the anchor point feature is used.
do_spec	scalar	If one, the specular highlight feature is used.
do_tex_constr	scalar	If one, the texture constraint feature is used.
init_rp	struct.	Initial rendering parameters. Any field missing is initialised to its default value.
sha_dim	$1 \times N_S$	If <code>match_shape</code> is one, this field specify which shape principal component is to be fitted. If <code>sha_dim(i)=0</code> , then the PC $i$ is not fitted.
tex_dim	scalar	Number of texture principal component to fit.
nseg	scalar	Should be 1 if <code>do_segment = 0</code> and 4 if <code>do_segment = 1</code> .
do_segment	scalar	If one, perform a segmented fitting. If 0, perform a global fitting.
seg_name	$4 \times 1$	Cell array with the name of each segment stored as a string.
edge	struct.	Parameters passed to the canny edge algorithm to detect the edges in the input image.
cdt	struct.	Parameters used to computed the Chamfer Distance Transform.
f_inner	struct.	Parameters used to compute the textured edge cost functions.
f_ctr	struct.	Parameters used to compute the contour edge cost functions.
f_tex	struct.	Parameters used to compute the pixel colour cost functions.
f_map	struct.	Parameters used to compute the prior probability cost functions.
f_anchor	struct.	Parameters used to compute the anchor cost functions.
f_spec	struct.	Parameters used to compute the specular highlight cost functions.
f_tex_constr	struct.	Parameters used to compute the texture constraint cost functions.

**Edge detector parameters** The following fields of the structure `edge` are parameters of the canny edge detector:

<i>name</i>	<i>type</i>	<i>description</i>
percent	scalar	This value is used to compute the threshold of the high threshold [17]. The higher is this parameter, the more edges will be returned. Default value: 0.7.
thresh_ratio	scalar	The low threshold is obtained by multiplying this parameter by the high threshold. This parameter must be between 0 and 1. Default value: 1.
sigma	scalar	Standard deviation of the Gaussian used as a low-pass filter. Default value: 1.

**Chamfer Distance Transform parameters** The following fields of the `cdt` structure are used to tune the computation and of the CDT and the transform applied to it:

<i>name</i>	<i>type</i>	<i>description</i>
<code>interpol</code>	string	Specifies the way the CDT is interpolated. As the vertices are projected on non-integer values of the image frame, the CDT must be interpolated between the pixels. Two values are accepted for this parameter: 'bilinear' computes a bilinear combination using the four neighbouring pixels. 'round' uses the value of the nearest pixel. 'bilinear' should always be used. Default value: 'bilinear'.
<code>transform</code>	string	After the CDT is computed, it is transformed by the function <code>cdt_transform</code> . The purpose is to make the edge fitting more robust. Several transformations are allowed, see the documentation of the <code>cdt_transform</code> function. Default value: 'huber_thresh'.
<code>sigma</code>	scalar	CDT transformation parameters, see the documentation of the <code>cdt_transform</code> function. Default value: 4.
<code>thresh</code>	scalar	CDT transformation parameters, see the documentation of the <code>cdt_transform</code> function. Default value: 10.

**Parameters common to each feature** The parameters of each feature are arranged in structures. The name of these structures are prefixed by a `f_`. The following three fields are common to all features.

<i>name</i>	<i>type</i>	<i>description</i>
<code>name</code>	string	Name of the feature.
<code>tau</code>	scalar	Weight of the feature. This parameter correspond to the $\tau$ parameter of Equation 4.40 on page 60.
<code>err_trsfm</code>	struct.	This structure specifies which error transform, $\rho_\sigma(x)$ in Equation (4.37), is used to fit the feature and its parameters.

The followings are fields of the `err_trsfm` structure that define the error norm used to fit a feature.

<i>name</i>	<i>type</i>	<i>description</i>
<code>transform</code>	string	Name of the error norm. One of the following three string must be set: 'nothing' for a least squares fitting, 'huber' for a fitting using the Huber's minimax estimator of Equation 4.38 on page 57, or 'lor' for the Lorentzian error function of Equation (4.39).
<code>sigma</code>	scalar	Value of the parameter $\sigma$ of the Huber and Lorentzian error norms. This parameter is not used for a least squares fitting.

**Edge feature parameters** Additionally to the fields common to all features, the textured edges and contour edges features include the character string parameter `orient`. It specifies the number of bins that are used to quantise the  $[-\pi, \pi]$  angle range between the image and contour edges direction. More information about this parameter is available in the help of function `makeorient`. The structure `f_ctr` includes the parameter `true_ctr`, as well. This is a  $N_v \times 1$  vector for which element  $i$  is one if the vertex  $i$  may be a valid contour vertex. This vector is used at step 4 of the contour computation (page 48).

**Pixel colour feature parameters** The pixel colour feature parameter structure, `f_tex`, includes the following fields.

<i>name</i>	<i>type</i>	<i>description</i>
<code>npoint</code>	scalar	Number of points used to compute the pixel colour cost function. If the input image is a colour image, the error vector $\mathbf{e}^c$ , includes three times as many elements. This parameter is used in the preparation function <code>mff.o.i</code> , in which <code>npoint</code> triangles are selected to be fitted.
<code>blind_tri</code>	$N_t \times 1$	The element $i$ of this vector is 1, if the triangle $i$ may be used to fit the model. The file <code>fp_data.mat</code> holds such a vector for which the lower part of the neck, the ears and the upper forehead are set not to be fitted.
<code>blind_isvis</code>	$N_v \times 1$	This parameter has no influence on the fitting. It is only used to computed the full pixel colour error obtained at the end of the fitting. The vertex $i$ is not used to compute the residual if the element $i$ of this parameter is null.
<code>sel_tri_coef</code>	$N_t \times 1$	This parameter sets the likelihood of selecting a triangle. It is computed by the function <code>make_tri_coef</code> and depends on the location of a triangle on the face (mouth, nose, eyes, and rest).

The probability density function of selecting one of the `npoint` triangle to be fitted among the set of visible model triangles is not uniform. If it was uniform, very few triangles would be selected in the nose, mouth and eyes region, as most of the model vertices are located on the cheek area. Hence, to favour the nose, mouth and eyes area over the cheek region, a non-uniform PDF is used to sample the model triangles that are fitted.

The probability of selecting a triangle is proportional to the multiplication of two factors: The first factor depends on the visibility of the triangle, the most visible a triangle is, the most likely it is to be chosen. The visibility of a triangle is given by the dot product of the triangle's normal,  $\mathbf{n}^{t,w}$ , and its viewing direction,  $\mathbf{v}$ . The second factor depends on location of the triangle on the face. Each region of a face is assigned to a weight that defines the importance of the area in the fitting. The higher the weight, the more likely a triangle is to be chosen. The weight is set in the field `sel_coef` of the structure `f_tex`. The weight of each triangle is stored in `sel_tri_coef` and computed in the function `make_tri_coef` based on `sel_coef`.

**Prior probability parameters** The prior probability feature parameters structure, `f_map`, includes two parameters, `lambda_shape` and `lambda_tex`, that weights the shape and texture part of the Gaussian prior cost function of Equation 4.35 on page 55.

**Anchor point feature parameters** The anchor point feature parameters structure, `f_anchor`, holds one parameter, `tau_ctr`, that weight the residual of the contour anchor points. If this

parameter is null, the contour anchor points are not used for the fitting, if it is infinite, the inner anchor points are not used.

**Specular highlight feature parameters** The specular highlight feature parameters structure, `f_spec`, includes the parameter `npoint` that sets the maximum number of specular points used for fitting.

**Texture constraint parameters** The texture constraint feature, `f_tex_constr`, includes the following three parameters:

<i>name</i>	<i>type</i>	<i>default value</i>	<i>description</i>
<code>npoint</code>	scalar	300	Maximum number of points not in the valid texture range, that can be included in the texture constraint cost function.
<code>lo</code>	scalar	0	Lower bound of the valid texture intensity range.
<code>hi</code>	scalar	255	Upper bound of the valid texture intensity range.

#### B.6.4 Cost functions parameters

The fitting parameters, detailed at the previous section, are changed by the user to modify the behaviour of the fitting algorithm. These parameters are utilised by the functions `mff_fit` and `mff_oi` (page 92). There is another set of parameters called the *cost function parameters*. These parameters are used by the feature cost function. They are stored in a cell array referenced by `oi`. In fact, this cell array does not only store parameters values but also any data required by the cost functions to compute their residuals and Jacobi matrices.

The cell array `oi` is given as input argument to the Matlab optimisation function `lsqnonlin` that passes it on to the main cost function `cost_full`. `oi` includes one cell element per features fitted. This cell element is a structure that holds the information necessary to a feature cost function to perform its computation.

**Parameters common to all features** The following fields are required for all features. They are used by the function `cost_full`.

<i>name</i>	<i>type</i>	<i>description</i>
<code>fct</code>	function handle	Handle to the cost function.
<code>err_trsfm</code>	struct.	Error norm transformation structure described at page 96.
<code>npoint</code>	scalar	Length of the residual vector returned by the cost function.
<code>tau</code>	scalar	Weight of the feature (Equation 4.40 on page 60).
<code>f_name</code>	string	Name of the feature. This is a copy of the field name of fitting parameters structure <code>f_p</code> (page 96).

**Edges** The textured and contour edge cost function structure include the following fields:

---

<i>name</i>	<i>type</i>	<i>description</i>
<code>cdt</code>	struct.	Chamfer Distance Transform data structure. It is described hereafter.
<code>model</code>	struct.	Shape model including only either the vertices of the textured edges or the vertices of the contour. This sub-model is compiled from the full 3DMM by the function <code>submodel</code> .
<code>g_idx</code>	cell array	The number of elements in this cell array is equal to the number of edge orientation bins. <code>g_idx(i)</code> lists the indexes of the model edge vertexes at orientation <code>i</code> .

The Chamfer Distance Transform data structure, `cdt`, is a cell array. The number of elements in this cell array is equal to the number of edge orientation bins specified by the parameters `orient` of the feature structures `f_inner` and `f_ctr` (page 96). A cell element is a CDT that includes only the edges at a specific orientation. Each cell elements include the following three matrices:

<i>name</i>	<i>type</i>	<i>description</i>
<code>cdt</code>	$h \times w$	Chamfer Distance Transform for a particular orientation.
<code>cdt_dx</code>	$h \times w$	Derivative of <code>cdt</code> along X.
<code>cdt_dy</code>	$h \times w$	Derivative of <code>cdt</code> along Y.

**Anchor** The anchor cost function structure include the following fields:

<i>name</i>	<i>type</i>	<i>description</i>
<code>model</code>	struct.	Shape model including only the model vertices in correspondence with the anchor points. This sub-model is compiled from the full 3DMM by the function <code>submodel</code> .
<code>screen</code>	$2 \times N_a$	X-Y coordinates of the anchor points. $N_a$ refer to the number of anchor points.

**Pixel colour** The pixel colour cost function structure include the following fields:

<i>name</i>	<i>type</i>	<i>description</i>
<code>img</code>	$h \times w \times 3$ or $h \times w$	Input image. It is a 3D matrix if the image is a colour image and a 2D matrix if it is a grey-level image.
<code>dImg_dx</code>	same as <code>img</code>	Derivative along X of the input image.
<code>dImg_dy</code>	same as <code>img</code>	Derivative along Y of the input image.
<code>tri_idx</code>	<code>npoint</code> $\times$ 1	Indexes of the triangles to be fitted.
<code>ver_idx</code>	<code>3npoint</code> $\times$ 1	Indexes of the vertices of the selected triangles.
<code>isshad</code>	<code>npoint</code> $\times$ 1	<code>isshad(i)</code> is one if the triangle <code>i</code> is in a cast shadow.
<code>corner_model</code>	struct.	Shape model with the vertices listed in <code>ver_idx</code> .

center_model	struct.	Shape and texture model with <code>npoint</code> vertices. Each vertex of this model is a 'virtual' vertex located at the triangle incenters of the 3DMM. It is computed by averaging the shape and texture PC of the three corners of the model <code>corner_model</code> .
outlier_img	$h \times w$	An outlier image may be used by the fitting. If it is available it may be given as input to the function <code>mff_fit</code> . This is a binary image where a pixel is set to one if it is an outlier in the input image. In this case it is not used in the computation of the pixel colour cost function.
alpha	$N_S \times 1$	Value of the shape parameters, $\alpha$ , before the fitting. This is used used in case the shape is not fitted.
beta	$N_T \times 1$	Value of the texture parameters, $\beta$ , before the fitting. This is used used in case the texture is not fitted.

**Texture constraint** The texture constraint cost function structure include the following fields:

<i>name</i>	<i>type</i>	<i>description</i>
<code>img</code>	$h \times w \times 3$ or $h \times w$	Input image.
<code>center_model</code>	struct.	Same as the field <code>center_model</code> of the pixel colour cost function data structure.

**Specular highlight** The specular highlight cost function structure include the following fields:

<i>name</i>	<i>type</i>	<i>description</i>
<code>spec_img</code>	$h \times w$	Binary image with the pixels detected on the specular highlight.
<code>corner_model</code>	struct.	Shape model of the vertices in correspondence with the pixels on the specular highlight.
<code>center_model</code>	struct.	Shape model of the incenters of the triangles whose corners are in the <code>corner_model</code> .

**Prior probability** The prior probability cost function structure include the following fields:

<i>name</i>	<i>type</i>	<i>description</i>
<code>n_xp_sha_dim</code>	scalar	Number of expression shape components.

## Appendix C

# Reproducing figures and experiments

The algorithms detailed in this thesis should not be analysed only from their theoretical perspective, for this analysis can not fully predict the behaviour nor the relevance of these algorithms on specific images. Numerical experiments must be carried out and these experiments must be reproducible.

Buckheit and Donoho [15] report that Claerbout, a distinguished exploration geophysicist, has championed the concept of *really reproducible research* in the computational sciences. The goal of exploration seismology is to produce an image of the subsurface as accurate as possible. However, Claerbout [18] has pointed out that the research deliverable is not an image itself, but instead the software environment that, applied in the right way, produces the image, and which, hopefully, could be applied to other datasets to produce equally nice images. The scientific findings may turn out to be a *knowledge of parameters settings* for this complex software environment that seem to lead to good results on real datasets.

Donoho summarises Claerbout's standpoint as follows: *An article about computational science in a scientific publication is not the scholarship itself, it is merely advertising of the scholarship. The actual scholarship is the complete software development environment and the complete set of instructions which generated the figures.*

To reproduce the figures, the tables and the experiments reported in this thesis, the Morphable Model Toolbox (Appendix B on page 85) is required. Additionally, some Matlab files are necessary. Tables C.1 and C.2 list the files required to reproduce the figures and the tables, respectively.

Some figures are not reproducible. These figures and the reason why they are not reproducible are listed on Table C.3.

Figures	Matlab files
4.2	matlab/cost_strong_weak/strong_cost.m
4.3	matlab/slack_prior/bad_tex.m
4.4, 4.5, 4.6, 4.12, and 4.13	matlab/cost_features/cost_features.m
4.7, and 4.8	matlab/edge/cdt_example.m
4.9	matlab/edge/inner_edges.m
4.10, and 4.11	matlab/edge/contour.m
4.16	matlab/specularities/example.m
4.17	matlab/tex_constr/tc_example.m
4.18, 4.19, and 4.20	matlab/mestimator/mestimator_plot.m
4.21, and 4.22	matlab/noise/noise_cost.m
4.23, and 4.24	matlab/multi_stage/ms_example.m
4.25	matlab/fit_res/fit_res.m
4.26	matlab/tex_ext/tex_ext.m
5.3	matlab/id/id.m
5.5, and 5.8	matlab/xp/xp_transfer.m
5.7	matlab/xp/xp_model.m
5.9	matlab/tracking/tracking.m
6.1	matlab/model/rec_error.m

Table C.1: List of files that reproduces the figures of this thesis.

Tables	Matlab files
4.3	matlab/multi_stage/ms_example.m
5.1, and 5.2	matlab/id/id.m

Table C.2: List of files that reproduces the tables of this thesis.

Figures	Reason
4.1	This figure requires a fitting with the SNO algorithm, it was produced using the interactive program <i>Mouflon</i> developed by our research group. This program is not publicly available.
4.14	Figure made with xfig.
5.6	Training 3D scans for expression are not publicly available.

Table C.3: List of figures that are not reproducible and the reason why this is the case.

# List of Figures

1.1	Top-down approach work-flow. . . . .	10
1.2	Bottom-up approach work-flow. . . . .	10
1.3	Integrated top-down/bottom-up approach work-flow . . . . .	10
1.4	Overview of the proposed fitting algorithm . . . . .	11
2.1	Texture and shape in the reference space $(u, v)$ . . . . .	19
2.2	Plot of the shape and texture eigenvalues . . . . .	20
2.3	Image of the four segments . . . . .	21
2.4	Inverse shape function computed using the shape projection . . . . .	22
3.1	First order approximation of the inverse shape projection defined by Baker and Matthews in [1]. . . . .	29
4.1	Example of poor fitting yielded by the Stochastic Newton Optimisation algorithm . . . . .	35
4.2	Synthetic example of combining various cost functions . . . . .	38
4.3	Example of unlikely texture that has a high probability according to the PCA model . . . . .	38
4.4	Input image on which the plot of various cost functions are plotted. . . . .	41
4.5	Plot of the pixel colour cost function along variations of the azimuth angle . . . . .	41
4.6	Plot of the edge cost function along variations of the azimuth angle . . . . .	43
4.7	Edge map of the input image shown in Figure 4.4 and its Chamfer Distance Transform. . . . .	45
4.8	Derivatives of the Chamfer Distance Transform . . . . .	46
4.9	Textured edges . . . . .	47
4.10	Contours according to the normal-based definition . . . . .	47
4.11	Mode contour computation . . . . .	48
4.12	Plot of the thresholded edge cost function along variations of the azimuth angle . . . . .	50
4.13	Plot of the anchor cost function along variations of the azimuth angle . . . . .	51
4.14	Light reflection vector . . . . .	52
4.15	Result of the specular highlight pixel detection algorithm . . . . .	53
4.16	Example of the improvement yielded by using the specular highlight feature . . . . .	53
4.17	Comparison of fitting result with and without the texture constraint feature . . . . .	55
4.18	Quadratic error norm (left) and its influence function (right). . . . .	57
4.19	Huber function and its influence function . . . . .	58
4.20	Lorentzian function and its influence function . . . . .	58
4.21	Ground truth synthetic image used for the cost function comparison. . . . .	59
4.22	Fitting results obtained on input images affected by Gaussian and Cauchy noise using least square and Lorentzian cost functions. . . . .	60
4.23	Fitting results obtained after each step of the Multi-Features Fitting algorithm . . . . .	61
4.24	Plot of the pixel colour error at each iteration of the Multi-Features Fitting algorithm . . . . .	62
4.25	Example of fitting results obtained with the Multi-Features algorithm . . . . .	63
4.26	Texture extraction and illumination inversion . . . . .	64
5.1	FERET images included in the series $b_j$ omitted in the recognition experiments . . . . .	68

5.2	The 21 light directions of the CMU-PIE dataset . . . . .	68
5.3	ROC for a verification task on the FERET and PIE dataset . . . . .	69
5.4	Example of imaging condition that yields lower identification performance . . . . .	71
5.5	Example of facial expression transfer. The expression of Mr Bean is transferred to Tony Blair and vice versa. . . . .	72
5.6	Exemplars 3D scans used to built the shape expression model . . . . .	73
5.7	First 12 shape expression principal components . . . . .	74
5.8	Fitting of images with the identity and expression 3DMM . . . . .	75
5.9	Face tracking result . . . . .	77
6.1	Plot of the shape and texture reconstruction error . . . . .	81

# List of Tables

4.1	Summary of features used for fitting and their main characteristics. . . . .	56
4.2	List of features fitted at each stage of the Multi-Feature Fitting algorithm . . . . .	61
4.3	Multi-Features Fitting timings . . . . .	61
5.1	Percentage of correct identification on the FERET dataset . . . . .	69
5.2	Mean percentage of correct identification obtained on the PIE data set . . . . .	70
5.3	Details of the identification results in presence of combined pose and illumination variation . . . . .	70
B.1	The five layers of the implementation of the MFF algorithm in the MM Toolbox. . .	91
B.2	List of MMT cost functions and their related feature. . . . .	92
C.1	List of files that reproduces the figures of this thesis. . . . .	102
C.2	List of files that reproduces the tables of this thesis. . . . .	102
C.3	List of figures that are not reproducible and the reason why this is the case. . . . .	102



# Curriculum Vitae

Address 3b Rue du Sauvage  
68300 Saint-Louis,  
France  
Tel. + 33 8 70 27 06 34  
Email sami.romdhani@unibas.ch  
Date of birth 30 September 1972  
Marital status Single  
Citizenship Belgium

## Education

2003 - 2004 University of Basel, Switzerland - Philosophical Doctorate: 'A Multi-Features Algorithm to Fit a 3D Model to Facial Images'  
1995 - 1996 Master of Science in Electronics Engineering at the University of Glasgow (Honors) - part of the European student exchange program 'ERASMUS'. MSc thesis: 'Face Recognition using Principal Components Analysis'  
1991 - 1996 Electronics Engineer at the 'Université Libre de Bruxelles' ('Distinction')  
1984 - 1991 High school diploma ('Lycée Daschbeck' - Brussels)

## Employment

2000 - 2003 Researcher at the University of Freiburg, Germany  
1998 - 2000 Researcher at the University of Westminster, England  
Mar - Jun 2000 Internship at Microsoft Research Cambridge, development of the patent 'Pattern Detection Methods and Systems, and Face Detection Methods and Systems' with Andrew Blake, Philip Torr and Bernhard Schölkopf  
1996 - 1998 Software Engineer at Alcatel Telecom: I was responsible of the software development of a module of a telecommunication device (a multiplexer).

## Publications

S. Romdhani, P. Torr, B. Schölkopf, and A. Blake. Efficient face detection by a cascaded support-vector machine expansion. *Proceedings of The Royal Society A*, 460(2501):3283–3297, November 2004.

S. Romdhani, V. Blanz, C. Basso, and T. Vetter. Morphable Models of Faces. In S. Z. Li and A. Jain, editors, *Handbook of Face Recognition*. Springer, in press.

M. Raetsch, S. Romdhani, and T. Vetter. Efficient Face Detection by a Cascaded Support Vector Machine using Haar-like Features. In *DAGM*, 2004.

- S. Romdhani and T. Vetter. Efficient, robust and accurate fitting of a 3d morphable model. In *Proceedings of the International Conference on Computer Vision*, 2003.
- S. Romdhani, N. Canterakis, and T. Vetter. Selective vs. global recovery of rigid and non-rigid motion. Technical report, CS Dept, University of Basel, 2003.
- S. Romdhani, V. Blanz, and T. Vetter. Face identification by fitting a 3D morphable model using linear shape and texture error functions. In *Proc. European Conference on Computer Vision*, 2002.
- V. Blanz, S. Romdhani, and T. Vetter. Face identification across different poses and illuminations with a 3D morphable model. In *Auto. Face and Gesture Recognition*, 2002.
- S. Romdhani, P. Torr, B. Schlkopf, and A. Blake. Computationally efficient face detection. In *Proceedings of the 8th International Conference on Computer Vision*, 2001.
- S. Romdhani, A. Psarrou, and S. Gong. On utilising Template and Feature-based Correspondence in Multi-View Appearance Models. In *Computer Vision - ECCV 2000*, Vol I: 799-813.
- S. Romdhani, A. Psarrou, and S. Gong. A Generic Face Appearance Model of Shape and Texture under very large Pose Variations from Profile to Profile Views. In *Proceedings of the 15th International Conference on Pattern Recognition*, 2000, Vol I: 1060-1063.
- S. Romdhani, A. Psarrou, and S. Gong. Tracking Deformable Face Models across Multiple Views. In *Proceeding of the fourth asian conference on computer vision*, Vol II: 1022-1027.
- S. Romdhani, A. Psarrou, and S. Gong. Learning a single active shape model for faces across views. In *First International Workshop on Recognition, Analysis and Tracking of Faces and Gestures in Real-time Systems*, 1999.
- S. Romdhani, A. Psarrou, and S. Gong. Multi-View Nonlinear Active Shape Model using Kernel PCA. In *Proceedings of the tenth British Machine Vision Conference*, September 1999. **Best Scientific Paper Award**

# Bibliography

- [1] S. Baker and I. Matthews. Equivalence and efficiency of image alignment algorithms. In *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition*, 2001.
- [2] J. A. Barnett. Computational methods for a mathematical theory of evidence. In *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, 1981.
- [3] R. Basri and D. W. Jacobs. Lambertian reflectances and linear subspaces. In *Proceedings of the International Conference on Computer Vision*, 2001.
- [4] R. Basri and D. W. Jacobs. Lambertian reflectance and linear subspaces. *IEEE Trans. Pattern Anal. Mach. Intell.*, 25(2):218–233, 2003.
- [5] P. Belhumeur and D. Kriegman. What is the set of images of an object under all possible illumination conditions. *Int. J. Computer Vision*, 28(3):245–260, 1998.
- [6] J. R. Bergen and R. Hingorani. Hierarchical motion-based frame rate conversion. Technical report, David Sarnoff Research Center Princeton NJ 08540, 1990.
- [7] D. Beymer and T. Poggio. Image representations for visual learning. *Science*, 1996.
- [8] D. Beymer, A. Shashua, and T. Poggio. Example based image analysis and synthesis. Technical report, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 1993.
- [9] I. Biederman. Recognition by components: a theory of human image understanding. *Psychological Review*, 94:115–147, 1987.
- [10] V. Blanz. *Automatische Rekonstruktion der dreidimensionalen Form von Gesichtern aus einem Einzelbild*. PhD thesis, Universität Tübingen, 2001.
- [11] V. Blanz, S. Romdhani, and T. Vetter. Face identification across different poses and illuminations with a 3d morphable model. In *Auto. Face and Gesture Recognition*, 2002.
- [12] V. Blanz and T. Vetter. A morphable model for the synthesis of 3D-faces. In *SIGGRAPH 99*, 1999.
- [13] V. Blanz and T. Vetter. Face recognition based on fitting a 3d morphable model. *IEEE Trans. Pattern Anal. Mach. Intell.*, 2003.
- [14] G. Borgefors. Hierarchical chamfer matching: A parametric edge matching algorithm. *IEEE Trans. Pattern Anal. Mach. Intell.*, 10(6):849–865, 1988.
- [15] J. Buckheit and D. Donoho. Wavelab and reproducible research, 1995.
- [16] M. C. Burl, M. Weber, and P. Perona. A probabilistic approach to object recognition using local photometry and global geometry. *Lecture Notes in Computer Science*, 1407:628–641, 1998.
- [17] J. Canny. A computational approach to edge detection. *IEEE Trans. Pattern Anal. Mach. Intell.*, 8(6), 1986.

- [18] J. Claerbout. Hypertext documents about reproducible research, <http://sepwww.stanford.edu>, 1994.
- [19] T. Cootes, D. H. Cooper, C. J. Taylor, and J. Graham. A trainable method of parametric shape description. In *Proc. British Machine Vision Conference*, 1991.
- [20] T. Cootes, G. Edwards, and C. Taylor. Active appearance model. In *Proc. European Conference on Computer Vision*, volume 2, pages 484–498. Springer, 1998.
- [21] T. Cootes and C. Taylor. Active shape models- - smart snakes. In *Proc. British Machine Vision Conference*, 1992.
- [22] T. Cootes, C. Taylor, A. Lanitis, D. Cooper, and J. Graham. Building and using flexible models incorporating grey-level information. In *Proceedings of the 4th International Conference on Computer Vision*, pages 355–365, 1993.
- [23] T. Cootes, C. J. Taylor, D. H. Cooper, and J. Graham. Training models of shape from sets of examples. In *Proc. British Machine Vision Conference*, pages 266–275, Berlin, 1992. Springer.
- [24] T. Cootes, K. Walker, and C. Taylor. View-based active appearance models. In *Fourth International Conference on Automatic Face and Gesture Recognition*, pages 227–232, 2000.
- [25] I. Craw and P. Cameron. Parameterizing images for recognition and reconstruction. In *Proc. BMVC*, 1991.
- [26] P. Debevec. Rendering synthetic objects into real scenes: Bridging traditional and image-based graphics with global illumination and high dynamic range photography. In *Proc. ACM SIGGRAPH*, 1998.
- [27] P. Debevec, T. Hawkins, C. Tchou, H.-P. Duiker, W. Sarokin, and M. Sagar. Acquiring the reflectance field of a human face. In *Proc. ACM SIGGRAPH*, 2000.
- [28] K. I. Diamantaras and S. Y. Kung. *Principal Component Neural Networks*. Wiley, 1996.
- [29] P. Felzenszwalb and D. Huttenlocher. Distance transforms of sampled functions. Technical Report TR2004-1963, Cornell Computing and Information Science, 2004.
- [30] A. Fitzgibbon. Robust registration of 2d and 3d point sets. In *Proc. British Machine Vision Conference*, volume 2, pages 411–420, 2001.
- [31] J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes. *Computer Graphics: Principles and Practice*. Addison-Wesley, 1996.
- [32] A. S. Georghiades, P. N. Belhumeur, and D. J. Kriegman. From few to many: Illumination cone models for face recognition under variable lighting and pose. *IEEE Trans. Pattern Anal. Mach. Intell.*, 23(6):643–660, 2001.
- [33] P. E. Gill, W. Murray, and M. H. Wright. *Practical Optimization*. Academic Press, 1981.
- [34] M. Gleicher. Projective registration with difference decomposition. In *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition*, pages 331–337, 1997.
- [35] J. Gray. Computer technology forecast for virtual observatories. Technical Report MSR-TR-2000-102, Microsoft Research, 2000.
- [36] H. Groemer. *Geometric applications of Fourier series and spherical harmonics*. Cambridge University Press, 1996.
- [37] P. W. Hallinan. *A deformable model for the recognition of human faces under arbitrary illumination*. PhD thesis, Harvard University, 1995.

- [38] F. R. Hampel, Ronchetti E.M., Rousseeuw P. J., and Stahel W. A. *Robust Statistics: the approach based on influence functions*. John Wiley & Sons, Inc., 1986.
- [39] K. M. Hanson and D. R. Wolf. Estimators fro the cauchy distribution. In G. R. Heidbreder, editor, *Maximum Entropy and Bayesian Methods*, pages 255–263. Kluwer Academic Publishers, 1996.
- [40] R. M. Haralick and L. G. Shapiro. *Computer and robot vision*. Addison-Wesley, 1992.
- [41] B. Heisele, P. Ho, and T. Poggio. Face recognition with support vector machines: Global versus component-based approach. In *International Conference on Computer Vision (ICCV'01)*, volume 2, pages 688–694, 2001.
- [42] B. Heisele, T. Serre, M. Pontil, T. Vetter, and T. Poggio. Categorization by learning and combining object parts. In *Advances in Neural Information Processing Systems*, volume 2, pages 1239–1245, 2002.
- [43] B. K. Horn and B. G. Schunck. Determining optical flow. *Artificial Intelligence*, 1981.
- [44] P. J. Huber. *Robust Statistics*. John Wiley & Sons, Inc., 1981.
- [45] Expanding moore’s law. Technical report, Intel Research, 2002.
- [46] M. Kirby and L. Sirovich. Application of the karhunen-loewe procedure for characterization of human faces. *IEEE Trans. Pattern Anal. Mach. Intell.*, 1990.
- [47] A. Lanitis, C. J. Taylor, and T. Cootes. An automatic face identification system using flexible appearance models. In *Proc. British Machine Vision Conference*, 1994.
- [48] A. Lanitis, C.J. Taylor, and T.F. Cootes. Automatic face identification system using flexible appearance models. *Image and Vision Computing*, 13(5):393–401, June 1995.
- [49] B. D. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In *Proc. Intl Joint Conf. Artificial Intelligence*, 1981.
- [50] D. Marr. *Vision*. Freeman, 1982.
- [51] S. R. Marschner, S. H. Westin, E. P. F. Lafortune, K. E. Torrance, and D. P. Greenberg. Image-based brdf measurement including human skin. In *Eurographics rendering workshop*, 1999.
- [52] T. P. Minka. Old and new matrix algebra useful for statistics. <http://www.stat.cmu.edu/~minka/papers/matrix.html>, 2000.
- [53] G. Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8), 1965.
- [54] H. Murase and S. K. Nayar. Visual learning and recognition of 3d objects from appearance. *Int. J. Computer Vision*, 1995.
- [55] A. Pentland, B. Moghaddam, and T. Starner. View-based and modular eigenspaces for face recognition. In *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition*, 1994.
- [56] P. J. Phillips, P. Grother, R. J. Michaels, D. M. Blackburn, E. Tabassi, and M. Bone. Face recognition vendor test 2002: Evaluation report. NISTIR 6965, Nat. Inst. of Standards and Technology, 2003.
- [57] P. J. Phillips, P. Rauss, and S. Der. Feret (face recognition technology) recognition algorithm development and test report. Technical report, U.S. Army Research Laboratory, 1996.
- [58] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical recipes in C : the art of scientific computing*. Cambridge University Press, 1992.

- [59] R. Ramamoorthi and P. Hanrahan. An efficient representation for irradiance environment maps. In *Proc. ACM SIGGRAPH*, pages 497–500, 2001.
- [60] S. Romdhani, V. Blanz, C. Basso, and T. Vetter. Morphable models of faces. In S. Z. Li and A. Jain, editors, *Handbook of Face Recognition*. Springer, in press.
- [61] S. Romdhani, V. Blanz, and T. Vetter. Face identification by fitting a 3d morphable model using linear shape and texture error functions. In *Proc. European Conference on Computer Vision*, 2002.
- [62] S. Romdhani, P. Torr, B. Schlkopf, and A. Blake. Efficient face detection by a cascaded support-vector machine expansion. *Proceedings of The Royal Society A*, 460(2501):3283–3297, November 2004.
- [63] S. Romdhani and T. Vetter. Efficient, robust and accurate fitting of a 3d morphable model. In *Proceedings of the International Conference on Computer Vision*, 2003.
- [64] J. Schürmann. *Pattern classification: a unified view of statistical and neural approaches*. John Wiley & Sons, Inc., 1996.
- [65] S. Sclaroff and J. Isidoro. Active blobs. In *Proceedings of the 6th International Conference on Computer Vision*, 1998.
- [66] T. Sim, S. Baker, and M. Bsat. The cmu pose, illumination and expression (pie) database of human faces. Technical report, CMU, 2000.
- [67] M.R. Spiegel. *Theory and Problems of Probability and Statistics*. McGraw-Hill, 1992. pp. 114-115.
- [68] M. Turk and A. Pentland. Eigenfaces for recognition. *Journal of Cognitive Neuroscience*, 1991.
- [69] T. Vetter and N. Troje. Separation of texture and shape in images of faces for image coding and synthesis. *Journal of the Optical Society of America*, 1997.
- [70] J. Xiao, S. Baker, I. Matthews, R. Gross, and T. Kanade. Real-time combined 2d+3d active appearance model. In *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition*, pages 535–542, 2004.