Separation of Identity and Expression Information in 3D Scans of Human Faces

Inauguraldissertation

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

Curzio Basso

aus Albenga, Italien

Basel, 2006

Genehmigt von der
Philosophisch-Naturwissenschaftlichen Fakultät

auf Antrag von

Prof. Dr. Thomas Vetter, Universität Basel,
Dissertationsleiter
Prof.in Dr.in Paola Campadelli, Università degli Studi
di Milano, Korreferentin

Basel, den 31. Januar 2006

Prof. Dr. Hans-Jakob Wirz, Dekan

Abstract

Our work is motivated by the problem of automatic *face recognition*, a difficult task, still missing a general solution. Its complexity lies in the wide range of variations presents in the input data, due to different lightings, background scenes and head positions. Moreover, the face appearance is affected by internal sources of variations: on a long temporal scale aging and weight gain, and on a short scale the action of the facial muscles. An effective recognition algorithm should be insensitive to all these sources of variations.

During the last decade, good results to the recognition problem have been obtained using *3D Morphable Models* (3DMMs). Their use allowed to separate the data variations due to the identity from the ones due to external sources like the lighting conditions. However, other internal sources were not considered. Our goal is to include expressions as an additional source of internal variation in 3DMMs, enabling us to recognize faces not only under different illuminations and pose conditions, but also with different expressions.

In general, the construction of a 3DMM requires a corpus of training data; for our task we need a training set including examples of both identity and expression variations. Unfortunately, their acquisition alone is not sufficient, since they have to be previously registered with a reference 3D head model. The registration of 3D scans of expressions is a difficult problem, which could not be solved with the registration algorithm previously used. The main contribution of our work is a new registration algorithm which can cope with arbitrary expressions in the 3D data. Our algorithm is also capable of registering data with missing values, an important property since virtually no 3D acquisition devices is immune to holes and artifacts in the output.

Given the training set of registered 3D examples, we construct a 3DMM where identity and expression variations are represented with two separate linear Gaussian models. The two models are then linearly combined, yielding an expression-identity 3DMM which we apply to the problem of 3D face recognition. Although this modeling approach does not take into account the interdependency between expressions and identity, the recognition performance is not negatively affected.

Contents

Chapter 1

Introduction

It is well known to computer scientists that a number of problems, which can be easily solved by humans, are extremely difficult to solve for a computer. *Face recognition* is such a problem, one for which a general solution is still missing. Apparently, the task of an automatic recognition algorithm is easy. Given two sets of face data, typically images, the *gallery* and the *probe*, the algorithm has to match each image in the probe to the image in the gallery which represents the same face, thereby recognizing the individuals in the two images as the same. Unfortunately, automatic recognition is more complex than it may appear at first sight. The main reason for its complexity is that, in a real scenario, the images in the probe have been acquired under different conditions than the ones in the gallery: the lighting can be different, as well as the background scene and the head position with respect to the camera. Moreover, the appearance of the face is affected not only by these external sources of variations, but also and especially by what we might call *internal* sources of variations. In fact, the human face is subject to continuous and dramatic changes, both on a long temporal scale, due to aging or weight gain, and on a short scale, due to the action of the facial muscles. An effective recognition algorithm should be insensitive to all these sources of variations.

During the last decade, good results to the recognition problem have been obtained by using *generative models*, and in particular *3D Morphable Models* (3DMMs). Such models can generate synthetic 3D faces with different identities, which can then be rendered to images under different lighting conditions and head poses. In particular, the model and rendering parameters can be optimized so that the generated image approximates a given one; this allows us to represent any image in terms of the model parameters, which depend only on the face identity. Performing the recognition on these parameters rather than on the images themselves provides good identification results over a broad range of illumination and pose variations. However, other sources of variations might still affect the face appearance, in particular expressions (by which we denote all the non-rigid deformations caused by the contraction and relaxation of the facial muscles, either due to emotion or to speech). Our goal is to bring 3DMMs

one step further, allowing them to model both identity and expression as separate sources of variations. This will enables us to recognize faces not only under different illumination and pose conditions, but also with different expressions.

In general, the construction of a 3DMM requires a corpus of training data: a set of 3D scans of human faces, which are used to learn the space of possible shapes and textures of the human face. In order to model identity and expression variations with a 3DMM, the training set has to include examples of both. However, acquiring 3D scans of facial expressions is not enough, since before the examples can be used for building the 3DMM, they have to be *registered* with a reference 3D head model. The term registration denotes the task of transforming the reference 3D model in such a way that it approximates well a novel 3D scan, under the constraint that the features of the reference are kept fixed in the parameterization domain. This essentially means that a vertex, which in the reference represents a specific feature of the human face (e.g. the inner corner of the right eye), has to represent the same feature after the transformation. As it turns out, registering 3D scans of facial expressions is quite difficult, and the registration algorithm previously used did not provide useful results. The main contribution of our work is a new registration algorithm, described in ***chapter 4***, which can cope with arbitrary expressions in the 3D data. A further advantage of the new registration algorithm is the possibility of registering data with missing values. During registration, the eventual missing parts of the 3D scan are reconstructed via a combined statistical-variational approach which ensures accurate reconstructions. This is an important property, since virtually no 3D acquisition devices is immune to holes and artifacts in the output. Our reconstruction method can also be applied independently from the registration, as shown in ***chapter 3***.

The construction of a 3DMM incorporating both identity and expression as separate sources of variations raises the additional problem of how to combine them. Ideally, the model should be able to represent the inherent dependency between expressions and identities, and a natural choice would have been to use a bilinear model. However, our face recognition experiments showed no practical advantage in using such a model, rather a performance drop. Considering also the restrictions imposed to the training set composition, we decided to discard bilinear models and to employ a linear model. In such a model, the two sources of variations are represented with two linear Gaussian models, which are then linearly combined. Although this simpler model does not take into account the interdependency between expressions and identity, this does not seem to have an impact on face recognition applications. The theoretical foundations of the model we use, as well as its learning procedure and the inference rules, are presented in ***chapter 2***.

As mentioned above, face recognition is typically performed on images, and in principle the combined identity-expression 3DMM can also be used for image data, provided that an image fitting algorithm is able to handle it. However, in our work we experimented only with the recognition from 3D data, for which we obtained a very good identification performance. In fact, 3D face recognition has got more and more attention in recent years, since it is naturally indepen-

dent from illumination conditions and head pose. With respect to image data, although we did not carry out recognition experiments, we show how the new model can be used for image normalization tasks: given the image of a face with arbitrary pose, lighting and expression, we can synthesize a new image of the same face with standard pose and lighting, and neutral expression. The results of our experiments, both for 3D data and images, are reported in ***chapter 5***.

1.1 Related Work

As explained in the previous section, our approach is based on a 3D generative model of the face data, encompassing both identity and expression. It is important to note that these two sources of variations have to be modeled separately in order to obtain a face recognition system that is insensitive to expressions. As far as the 3DMMs are concerned, this separate modeling has never been tried. The original model ([BV99, BV03]) did not include expressions, and the 3DMM of expressions presented in [BBPV03] was neither combined with the identity variations nor applied to face recognition. However, the work of [CCET99] explored the possibility of incorporating expressions, as well as pose and illumination, as separate sources of linear variations in the context of an *active appearance model* (AAM, [CET98, ECT98]). A close relative of the 3DMM, the AAM is a generative model of face images, which, in its original form, did not separate between different sources of variations. In fact, at the classification stage a Linear Discriminant Analysis (LDA) was used to improve the recognition performance by isolating the variations due to the identity. With the explicit modeling of expressions as a separate source of variation, recognition can be performed by fitting the model to the probe and gallery data, and then comparing only the identity parameters of the model. From this respect then, our work follows an approach similar to [CCET99], with the important difference however that we apply it in the context of a 3DMM.

Recently there have also been many attempts to incorporate the different sources of variations in a generative framework via *multilinear* models, rather than superimposing them in a single linear model, but the applications to face recognition are relatively few. In [TF00], for instance, a bilinear model of face images has been applied to combine pose and identity variations; expressions variations, together with identity, pose and illumination, are considered in [VT02], although the authors do not treat the application of such a model for face recognition. Similarly, [CDB02] uses a bilinear model spanning identity and expressions to separate the two informations in video sequences, but again no recognition application is presented. To the best of our knowledge, there has been so far no attempt of applying 3D generative models to expression-invariant face recognition. However, some works have already presented explored the possibility of building such models. Both [WHL$^+$04] and [VBPP05], for instance, use multilinear models to track expressions in dynamic sequences of 3D data, and transfer them to other individuals. The idea of applying a multilinear model to the task of face recognition is certainly appealing, but its conversion

to practice seems to proceed slowly. One of the reasons might be that a multilinear model requires a much larger training set in order to achieve the same generalization power of a linear model.

To conclude our overview of the available methods for expression-insensitive face recognition, we should also mention that generative models are only one possible approach. A whole class of methods works by extracting and comparing certain features from the face images or 3D scans, without relying on such models. These methods relies on the assumption that the chosen features are invariant, for each individual, under changes of illumination, pose, expressions, or any other variation present in the data. However, few of them address the problem of facial expressions in 3D face recognition. One notable exception is [BBK05], where they extract from the 3D scans what is called the canonical form of the 3D surface. As shown in the paper, the canonical form is invariant under isometric transformations, that is transformations that keep the geodesic distance constant. If expressions are isometric transformations, an ideal expression-insensitive recognition method is offered by surface matching of the canonized 3D surfaces. Other recognition methods for 3D data are compared in [CBF05], together with a novel one which uses as features three local regions (nose tip, nose bridge and eyes sockets), automatically detected in the probe and registered via ICP with the same regions in the gallery. Clearly, such methods are restricted to the type of data they are designed for; this is an important difference with respect to recognition based on 3DMMs, which can be applied to both images and 3D data, and would be therefore particularly well suited for multimodal recognition.

1.2 Definitions and Notation

We conclude this chapter by introducing some of the basic notions required in the rest of the work. In order to avoid confusion, we also summarized the notation used in the table 1.1.

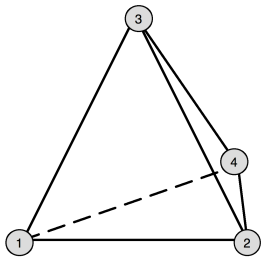
Most of our work deals with 3D objects, more specifically triangular meshes, which can be defined by a structure $(\mathcal{M}, \boldsymbol{S})$. The topology of the mesh is stored in the graph $\mathcal{M} = (V, E, F)$, defined by the vertices V, the edges E, and the faces (in this case only triangles) F (see figure 1.1). In general, however, we do not use the graph $\mathcal{M}$ but only the neighboring information, that is the indices of the vertices which share an edge with a given vertex. We denote by $\mathcal{N}_i$ the set of indices of the neighbors of vertex i.

The shape of the mesh is stored in $\boldsymbol{S}$, which holds the 3D positions of the vertices. In practice, if the mesh has n vertices with positions $(x_i, y_i, z_i)_{i=1\dots n}$, $\boldsymbol{S}$ will be an $n \times 3$ matrix stacking their coordinates:

$$\boldsymbol{S} = \begin{pmatrix} x_1 & y_1 & z_1 \\ & \vdots & \\ x_n & y_n & z_n \end{pmatrix}$$

a, A, γ	Scalars are denoted by letters with normal typeface.
$\boldsymbol{a}, \boldsymbol{\gamma}$	Vectors are denoted by lowercase letters with bold typeface.
$\boldsymbol{a}_i$	Element i of the vector $\boldsymbol{a}$.
$\boldsymbol{a}^T$	Transpose of the column-vector $\boldsymbol{a}$. If $\boldsymbol{a}$ is n-dimensional, $\boldsymbol{a}^T$ has dimensions $1 \times n$.
$\|\boldsymbol{a}\|$	L_2-norm of the vector $\boldsymbol{a}$.
$\boldsymbol{A}, \boldsymbol{\Gamma}$	Matrices and tensors are denoted by bold uppercase letters.
$\boldsymbol{A}_{ij}$	Element (i, j) of the matrix $\boldsymbol{A}$.
$\|\boldsymbol{A}\|$	Determinant of the matrix $\boldsymbol{A}$.
$tr\ \boldsymbol{A}$	Trace of the matrix $\boldsymbol{A}$.
$\boldsymbol{A}^T$	Transpose of the matrix $\boldsymbol{A}$.
$vec(\boldsymbol{A})$	Vectorization of the matrix $\boldsymbol{A}$. If $\boldsymbol{A}$ is $m \times n$, $vec(\boldsymbol{A})$ is $mn \times 1$.

Table 1.1: A summary of the mathematical notation used in our work.



$(\mathcal{M}, \boldsymbol{S})$

$$V = \{1, 2, 3, 4\}$$

$$E = \left\{ \begin{array}{c} (1,2), (1,3), (1,4) \\ (2,3), (2,4), (3,4) \end{array} \right\}$$

$$F = \left\{ \begin{array}{c} (1,2,3), (1,2,4) \\ (1,3,4), (2,3,4)) \end{array} \right\}$$

Figure 1.1: The topology of a 3D mesh, in this example a tetrahedron, is defined by a graph $\mathcal{M} = (V, E, F)$. On the right the sets of vertices V, of edges E and of triangles F are explicitly written.

Sometimes we will also use a vectorial representation of $\boldsymbol{S}$, obtained by concatenating its rows, and we will denote it by $\boldsymbol{s}$:

$$\boldsymbol{s} = (x_1, y_1, z_1, \ldots, x_n, y_n, z_n)^T .$$

The transformation from $\boldsymbol{S}$ to $\boldsymbol{s}$ is represented by the $vec(\cdot)$ operator, so that we can also write:

$$\boldsymbol{s} = vec(\boldsymbol{S}).$$

If the 3D mesh is textured, as it is in our case, then it also has to include the texture informations, which we denote by $\boldsymbol{T}$. The structure of $\boldsymbol{T}$ will depend on the type of texturing: if the mesh is vertex colored, $\boldsymbol{T}$ will be an $n \times 3$ matrix just like $\boldsymbol{S}$; if the mesh is texture mapped $\boldsymbol{T}$ will be an image and $\mathcal{M}$ will also store the texture coordinates for each corner of the triangles.

The 3D faces synthesized by a 3DMM are assumed to be the result of a stochastic process, governed by random variables with a multivariate Gaussian

distribution. It is therefore useful to recall that a multivariate Gaussian distribution of an n-dimensional random variable $\boldsymbol{x}$, which we denote by $\mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$, is parameterized by its mean $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$, and is defined as

$$\begin{aligned} p(\boldsymbol{x}) &= \mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) \\ &= (2\pi)^{-n/2} |\boldsymbol{\Sigma}|^{-1/2} \exp\left\{ -\frac{1}{2} (\boldsymbol{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\boldsymbol{x} - \boldsymbol{\mu}) \right\} \end{aligned}$$

We conclude by recalling some derivation rules for vector and matrix functions which we will use especially in chapters 2 and 3; for a more complete treatment, you can refer to [MN02]. Two basic rules are:

$$\frac{\partial \boldsymbol{A}\boldsymbol{x}}{\partial \boldsymbol{x}} = \boldsymbol{A} \quad \text{and} \quad \frac{\partial \boldsymbol{x}^T \boldsymbol{A}\boldsymbol{x}}{\partial \boldsymbol{x}} = \boldsymbol{x}^T (\boldsymbol{A} + \boldsymbol{A}^T)$$

In particular, setting $\boldsymbol{A} = \boldsymbol{I}$ in the second we derive

$$\frac{\partial \|\boldsymbol{x}\|^2}{\partial \boldsymbol{x}} = 2\boldsymbol{x}^T.$$

The chain rule holds also for functions of vectors and matrices:

$$\frac{\partial f(g(\boldsymbol{x}))}{\partial \boldsymbol{x}} = \frac{\partial f}{\partial g} \frac{\partial g}{\partial \boldsymbol{x}},$$

and therefore

$$\begin{aligned} \frac{\partial \|\boldsymbol{a} + \boldsymbol{B} \cdot \boldsymbol{x}\|^2}{\partial \boldsymbol{x}} &= 2(\boldsymbol{a} + \boldsymbol{B} \cdot \boldsymbol{x})^T \frac{\partial \boldsymbol{a} + \boldsymbol{B} \cdot \boldsymbol{x}}{\partial \boldsymbol{x}} \\ &= 2(\boldsymbol{a} + \boldsymbol{B} \cdot \boldsymbol{x})^T \boldsymbol{B} \end{aligned}$$

Finally, the derivative of the trace of a matrix is given by:

$$\frac{\partial tr\ \boldsymbol{X}^T \boldsymbol{A}\boldsymbol{X}}{\partial \boldsymbol{X}} = (\boldsymbol{A} + \boldsymbol{A}^T)\boldsymbol{X}.$$

Chapter 2

3D Morphable Models

In order to model the variations of three-dimensional scans of human faces, it is necessary to represent the scans as vectors in a space. Since in practice 3D scans are 3D meshes, a natural representation as vectors is offered by stacking together the attributes (position and color) of all vertices. However, such a representation has a relatively high dimensionality (typically $\sim 10^5$): a low-dimensional, equivalent representation would be advantageous both computationally and qualitatively. This equivalent representation can be achieved by a generative model, which we can imagine as a function $\boldsymbol{f}$ from a subspace $\mathcal{F} \subseteq \mathbb{R}^m$ (the parameter space) to the space of all possible 3D objects, such that:

- for any choice of parameters $\boldsymbol{x} \in \mathcal{F}$, $\boldsymbol{f}(\boldsymbol{x})$ yields a 3D object belonging to the class of human faces;
- for any human face with arbitrary identity and expression, there exists a $\boldsymbol{x} \in \mathcal{F}$ such that $\boldsymbol{f}(\boldsymbol{x})$ approximates it well;
- a probabilistic model for $\boldsymbol{f}$ is defined, and in particular the *posterior probability* $p(\boldsymbol{x}|\boldsymbol{y})$.

The first two conditions require that the image $\boldsymbol{f}(\mathcal{F})$ of the parameter space covers exactly the space of all possible 3D faces, and not more. The third condition plays a key role when we need to find the actual point $\boldsymbol{x}$ in the parameter space corresponding to a given face, as it is the case for instance when applying the model to tasks of image analysis. Since $\boldsymbol{f}$ is in general not invertible, $\boldsymbol{x}$ is found by using the probabilistic model to *infer* it from the data – that is looking for the most likely values of the parameters given the face.

Let us assume for the moment that any 3D object has always n vertices (you can for instance imagine to subdivide or decimate an appropriate set of triangles/vertices). Then, the space of all possible 3D shapes will be $\mathbb{R}^{n\times 3}$, and all possible human faces will lie in a (relatively small) subspace of $\mathbb{R}^{n\times 3}$. In general, a model for this subspace can be learned from a set of examples of

human faces (a so-called *training* set), but without any assumptions on the form of the subspace this can be a challenging task, due to the high dimensionality of the data w.r.t. the number of examples (typically $n \sim 10^4$).

The method we will describe is based on the 3D Morphable Model, presented in [BV99] for modeling human faces with varying identity. Such models are based on the key observation that given two 3D faces, if they are previously registered, their linear interpolation (also known as *morph*) will still describe a human face. That is, if $\boldsymbol{S}_1$ and $\boldsymbol{S}_2$ are the shapes of such two examples after registration, then their interpolation

$$\boldsymbol{S}(a) = (1-a) \cdot \boldsymbol{S}_1 + a \cdot \boldsymbol{S}_2 \quad \text{with} \quad a, b \in [0,1] \tag{2.1}$$

will belong to the subspace of human faces (see figure 2.1 for an example of shape and texture interpolations). The equation (2.1) can be generalized to the case of m examples:

$$\boldsymbol{S} = \sum a_i \cdot \boldsymbol{S}_i \quad \text{with} \quad \sum a_i = 1,\ a_i \in [0,1],$$

and it is reasonable to assume that the registered examples lie on a subspace which is, at least approximately, linear.

In the following sections we will extend the previous concept of 3D Morphable Models in two directions: on the one hand, we will apply it for modeling both identity and expressions, and on the other hand we will present a scheme to deal with missing values in the face data. Note that for the rest of the chapter we will assume that the examples have been previously registered; the registration process is explained in detail in chapter 4.

2.1 Linear Gaussian Models

Let us consider the shapes of the registered 3D faces. For the rest of this chapter we will use for the 3D shape not the matrix representation $\boldsymbol{S}$, but rather its vectorial representation in $\mathbb{R}^{3n}$, obtained by flattening the matrix through the $vec(\cdot)$ operator:

$$\boldsymbol{s} = vec(\boldsymbol{S}) = (x_1, y_1, z_1, \ldots, x_n, y_n, z_n)^T$$

This change of representation is needed in order to express in matrix form the assumption that the space of the face shapes can be approximated by a linear subspace of $\mathbb{R}^{3n}$. A linear subspace of $\mathbb{R}^{3n}$ is defined by a vector $\bar{\boldsymbol{s}} \in \mathbb{R}^{3n}$ and a matrix $\boldsymbol{C} \in \mathbb{R}^{3n \times k}$ with $k < 3n$. Just like a line in $\mathbb{R}^3$ is defined by a point and a direction, $\bar{\boldsymbol{s}}$ is a point lying on the subspace, while the columns of $\boldsymbol{C}$ are the directions spanned by the subspace. A generic shape vector $\boldsymbol{s}$ can then be written as

$$\boldsymbol{s} = \bar{\boldsymbol{s}} + \boldsymbol{C} \cdot \boldsymbol{\alpha} + \boldsymbol{\epsilon}. \tag{2.2}$$

That is, any shape vector $\boldsymbol{s}$ is decomposed into a point lying on the linear subspace and a residual displacement. The point on the subspace is specified by

Figure 2.1: Interpolation between two examples. The originals are on the top-left and bottom-right corner; in the horizontal direction the shape is interpolated, while the texture is interpolated in the vertical direction.

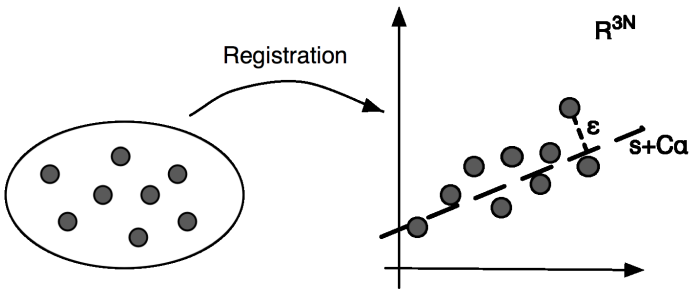


Figure 2.2: Learning a linear model of 3D human faces. The original data, on the left, cannot be linearly combined. By registering them, they are mapped to a subspace of $\mathbb{R}^{3n}$ which can be approximated by the linear subspace $\bar{\boldsymbol{s}} + \boldsymbol{C} \cdot \boldsymbol{\alpha}$ (represented as a dashed line). The approximation error $\boldsymbol{\epsilon}$ is modeled as a Gaussian noise.

the vector $\boldsymbol{\alpha} \in \mathbb{R}^k$, while the residual is defined by $\boldsymbol{\epsilon} \in \mathbb{R}^{3n}$, which is in practice the error made by the linear approximation (see figure 2.2).

A statistical model can be derived if we assign to the latent variables $\boldsymbol{\alpha}$ and $\boldsymbol{\epsilon}$ a probability distribution. By assuming for both $\boldsymbol{\alpha}$ and $\boldsymbol{\epsilon}$ a Gaussian distribution with zero mean and diagonal covariance:

$$\begin{aligned} p(\boldsymbol{\alpha}) = \mathcal{N}(0, \boldsymbol{I}) &= (2\pi)^{-k/2} \exp\left\{-\frac{1}{2}\|\boldsymbol{\alpha}\|^2\right\}, \\ p(\boldsymbol{\epsilon}) = \mathcal{N}(0, \sigma^2\boldsymbol{I}) &= (2\pi\sigma^2)^{-3n/2} \exp\left\{-\frac{1}{2\sigma^2}\|\boldsymbol{\epsilon}\|^2\right\}. \end{aligned} \tag{2.3}$$

we will obtain what is known as a *linear Gaussian model*. With this model, it can be shown that the shape vector $\boldsymbol{s}$ will also have a Gaussian distribution, centered on $\bar{\boldsymbol{s}}$ and with covariance $\boldsymbol{M} = \boldsymbol{C}\boldsymbol{C}^T + \sigma^2\boldsymbol{I}$, that is:

$$p(\boldsymbol{s}) = \mathcal{N}(\bar{\boldsymbol{s}}, \boldsymbol{M}),$$

The conditional probability of $\boldsymbol{s}$ given $\boldsymbol{\alpha}$ can be also explicitly computed, and is again a Gaussian distribution:

$$p(\boldsymbol{s}|\boldsymbol{\alpha}) = \mathcal{N}(\bar{\boldsymbol{s}} + \boldsymbol{C} \cdot \boldsymbol{\alpha}, \sigma^2\boldsymbol{I}). \tag{2.4}$$

Note that if σ^2 were zero, $p(\boldsymbol{s}|\boldsymbol{\alpha})$ would degenerate to a singularity, which is what one would expect in absence of noise.

The values of the model parameters $\bar{\boldsymbol{s}}$, $\boldsymbol{C}$ and σ^2 are not known from the start, and they have to be learned from a training set of exemplar shapes $\boldsymbol{s}_1$, $\ldots$, $\boldsymbol{s}_m$. The idea is that the training set constitutes a finite sample drawn from the distribution $p(\boldsymbol{s})$, and if the sample is representative enough a good estimate

of the model parameters can be obtained by maximizing the likelihood data. Denoting by $\bar{\boldsymbol{s}}$ the sample mean

$$\bar{\boldsymbol{s}} = \frac{1}{m}\sum_{i=1}^{m} \boldsymbol{s}_i,$$

and by $\boldsymbol{\Sigma}$ the sample covariance of the training set

$$\boldsymbol{\Sigma} = \frac{1}{m}\sum_{i=1}^{m} (\boldsymbol{s}_i - \bar{\boldsymbol{s}})(\boldsymbol{s}_i - \bar{\boldsymbol{s}})^T,$$

the log-likelihood can be written as

$$\begin{aligned} \mathcal{L} &= \sum_{i=1}^{m} \log p(\boldsymbol{s}_i) \\ &= -\frac{m}{2}\left\{3n\log(2\pi) + \log|\boldsymbol{M}| + tr(\boldsymbol{M}^{-1}\boldsymbol{\Sigma})\right\} \end{aligned}$$

As shown in [TB99] (to which we refer for details), the maximization of $\mathcal{L}$ has a close solution, intimately related to the *Principal Component Analysis* (PCA). The optimal estimates of $\boldsymbol{C}$ and σ^2 can be computed from the *Singular Value Decomposition* (SVD) of the *centered* data matrix

$$\boldsymbol{A} = (\boldsymbol{s}_1 - \bar{\boldsymbol{s}}, \ldots, \boldsymbol{s}_m - \bar{\boldsymbol{s}}) \in \mathbb{R}^{3n\times m},$$

which results in

$$\boldsymbol{A} = \boldsymbol{U}\cdot\boldsymbol{W}\cdot\boldsymbol{V}^T,$$

where $\boldsymbol{U} = (\boldsymbol{u}_1, \ldots, \boldsymbol{u}_{m-1})$ is a column-orthogonal matrix ($\boldsymbol{U}^T\boldsymbol{U} = \boldsymbol{I}$), $\boldsymbol{W}$ is a diagonal matrix, and $\boldsymbol{V}$ is orthogonal ($\boldsymbol{V}\boldsymbol{V}^T = \boldsymbol{V}^T\boldsymbol{V} = \boldsymbol{I}$). It is also easy to verify that $\boldsymbol{U}$ and $\boldsymbol{\Lambda} = \boldsymbol{W}^2/m$ hold respectively the eigenvectors and eigenvalues of the sample covariance matrix $\boldsymbol{\Sigma}$, since:

$$\begin{aligned} \boldsymbol{\Sigma}\boldsymbol{U} &= \frac{1}{m}\boldsymbol{A}\boldsymbol{A}^T\boldsymbol{U} \\ &= \frac{1}{m}\boldsymbol{U}\boldsymbol{W}^2\boldsymbol{U}^T\boldsymbol{U} \\ &= \frac{1}{m}\boldsymbol{W}^2\boldsymbol{U} \end{aligned}$$

Denoting by w_i the diagonal elements of $\boldsymbol{W}$, the optimal estimates of $\boldsymbol{C}$ and σ^2 are given by

$$\begin{aligned} \sigma^2 &= \frac{1}{m(3n-k)}\sum_{i=k+1}^{m-1} w_i^2 \\ \boldsymbol{C} &= (\boldsymbol{u}_1\sqrt{\frac{w_1^2}{m} - \sigma^2}, \ldots, \boldsymbol{u}_k\sqrt{\frac{w_k^2}{m} - \sigma^2}) \in \mathbb{R}^{3n\times k} \\ &= \boldsymbol{U}_k\cdot\left(\boldsymbol{\Lambda}_k - \sigma^2\boldsymbol{I}\right)^{1/2}, \end{aligned} \tag{2.5}$$

where k is the number of principal directions which are retained (for $k = m - 1$ the noise estimate is null and we obtain a standard PCA model). To summarize, the model is built by

1. computing the eigenvectors $\boldsymbol{u}_i$ and eigenvalues w_i of the sample covariance matrix via SVD of the data matrix $\boldsymbol{A}$;
2. fixing the number k of eigenvectors/eigenvalues which we want to retain in the model;
3. defining the noise variance σ^2 as the sum of the discarded eigenvalues;
4. defining the generative matrix $\boldsymbol{C}$ as in equation (2.5), using the retained pairs of eigenvectors and eigenvalues.

The difference from this model and the one obtained from PCA lies in the last two steps. Discarding some of the higher components $\boldsymbol{u}_i$, their contributions to the total sample variance accumulates in the model noise and scales down the variance of the retained components. Therefore, although the directions of the retained components are the same as in the PCA model, their magnitude is typically smaller, which accounts for the fact that some of the sample variance in those directions is due to the (isotropic) noise (see figure 2.3).

2.1.1 Combined Model

The model of the previous section is appropriate when there is a single type of variations in the data, for instance only the identity or the expression. However, if both type of variations were present in the data, a unique linear model would not allow us to discriminate variations due to expression from the ones due to identity. Since this capability is essential to perform tasks such as expression-independent recognition, we cannot use the linear model of the previous section as is. In order to manage this bi-modal face variations, we assume a generic face vector to be a sum of an identity vector and an expression vector:

$$\boldsymbol{s} = \boldsymbol{s}_{id} + \boldsymbol{s}_{xp},$$

where $\boldsymbol{s}_{id}$ represents the face with neutral expression while $\boldsymbol{s}_{xp}$ holds the vertices displacements due to the expression. Assigning to each of them a separate linear Gaussian model, a generic face is modeled as a linear superposition of two linear Gaussian models:

$$\boldsymbol{s} = \bar{\boldsymbol{s}}_{id} + \boldsymbol{C}_{id} \cdot \boldsymbol{\alpha}_{id} + \bar{\boldsymbol{s}}_{xp} + \boldsymbol{C}_{xp} \cdot \boldsymbol{\alpha}_{xp} + \boldsymbol{\epsilon},$$

with the usual Gaussian prior for the latent variables $\boldsymbol{\alpha}_{id}$, $\boldsymbol{\alpha}_{xp}$ and $\boldsymbol{\epsilon}$. Clearly, once the model parameters are fixed, this is nearly equivalent to the model of equation (2.2):

$$\boldsymbol{s} = \underbrace{\bar{\boldsymbol{s}}_{id} + \bar{\boldsymbol{s}}_{xp}}_{\bar{\boldsymbol{s}}} + \underbrace{\begin{pmatrix} \boldsymbol{C}_{id} & \boldsymbol{C}_{xp} \end{pmatrix}}_{\boldsymbol{C}} \cdot \underbrace{\begin{pmatrix} \boldsymbol{\alpha}_{id} \\ \boldsymbol{\alpha}_{xp} \end{pmatrix}}_{\boldsymbol{\alpha}} + \boldsymbol{\epsilon} \tag{2.6}$$

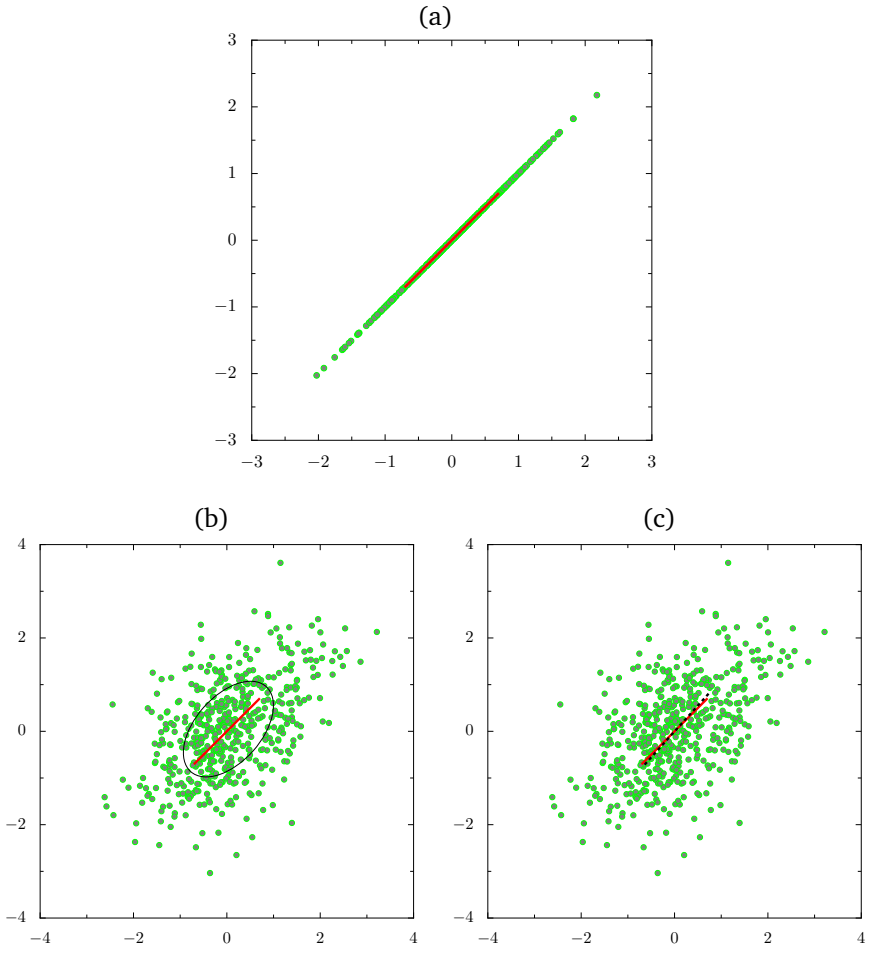


Figure 2.3: Comparison of PCA and PPCA estimations with a toy example. (a) Data points generated with a one-dimensional normal distribution, the red line marks the region within one sigma from the mean. (b) PCA performed on the data points to which Gaussian noise has been added, the black ellipsoid marks the region within one sigma from the sample mean. Even discarding the second principal component, the PCA overestimates the variance along the first axis. (c) PPCA scales down the variance along the first axis and results in a more precise estimate.

with the only difference that now the matrix $\boldsymbol{C}$ is not column-orthogonal anymore.

In order to learn the distinct model parameters for the identity and expressions components we use two training sets. A first set of examples with neutral expression (that is with all facial muscles relaxed) and varying identity is used to estimate the identity parameters $\bar{\boldsymbol{s}}_{id}$ and $\boldsymbol{C}_{id}$, as outlined in the previous section. A second set of expressions examples from a smaller set of persons is used to estimate the expression parameters $\bar{\boldsymbol{s}}_{xp}$ and $\boldsymbol{C}_{xp}$, by first removing from the examples the corresponding neutral expression and then applying the method of the previous section. Assume the expressions examples are acquired from p different persons. Then, for the subset of expressions acquired from the i-th person, we have her neutral expression $\boldsymbol{n}^i$ and m_i examples $\boldsymbol{s}^i_j$, from which we build a matrix

$$\boldsymbol{B}_i = (\boldsymbol{s}^i_1 - \boldsymbol{n}^i, \ldots, \boldsymbol{s}^i_{m_i} - \boldsymbol{n}^i) \in \mathbb{R}^{3n \times m_i}.$$

All the person-specific matrices $\boldsymbol{B}_i$ are then put together into a matrix

$$\boldsymbol{B} = (\boldsymbol{B}_1 \ldots \boldsymbol{B}_p) \in \mathbb{R}^{3x \times \sum m_i},$$

which is then recentered, obtaining a matrix $\boldsymbol{A}$ which can be used as input of the same learning algorithm used for the identity-only data.

As explained in the previous section the estimate of the noise variance σ^2 depends on the number of eigenvectors of the sample covariance which are retained in the model. Due to the fact that, in the case of an identity-expression combined model, the model components are learned separately, we could have in principle two different estimates of σ^2. To avoid this, we let the number of expression components k_{xp} depend on the σ^2 estimated from the selection of k_{id} identity components. Once σ^2 is fixed, k_{xp} can be chosen so that the estimate of the noise variance obtained by the discarded expression components is as closest as possible to σ^2.

2.1.2 Inference

As emphasized in the introduction, there are applications of the model for which given a novel face vector $\boldsymbol{s}$ a corresponding parameters vector $\boldsymbol{x}$ has to be found. Since $\boldsymbol{C}$ is in general non-square and therefore non-invertible, $\boldsymbol{x}$ is found not by analytical inversion of equation (2.2), but rather by statistical inference, that is minimizing the log-inverse of the posterior probability of the model coefficients

$$\begin{aligned} -\log p(\boldsymbol{\alpha}|\boldsymbol{s}) &= -\log \frac{p(\boldsymbol{s}|\boldsymbol{\alpha}) \cdot p(\boldsymbol{\alpha})}{p(\boldsymbol{s})} \\ &= -\log p(\boldsymbol{s}|\boldsymbol{\alpha}) - \log p(\boldsymbol{\alpha}) + \log p(\boldsymbol{s}) \end{aligned}$$

If the shape vectors are independent and identically distributed (iid) the last term is constant, and plugging in the above equation the probabilities (2.4) and (2.3), the log-inverse is

$$-\log p(\boldsymbol{\alpha}|\boldsymbol{s}) = \frac{\|\boldsymbol{s} - \bar{\boldsymbol{s}} - \boldsymbol{C} \cdot \boldsymbol{\alpha}\|^2}{2\sigma^2} + \frac{\|\boldsymbol{\alpha}\|^2}{2} + const.$$

Its global minimum is obtained by setting to zero its derivative w.r.t. the model coefficients $\boldsymbol{\alpha}$:

$$\begin{aligned} -\frac{\partial}{\partial \boldsymbol{\alpha}} \log p(\boldsymbol{\alpha}|\boldsymbol{s}) &= -\frac{1}{\sigma^2}\boldsymbol{C}^T \cdot (\boldsymbol{s} - \bar{\boldsymbol{s}} - \boldsymbol{C} \cdot \boldsymbol{\alpha}) + \boldsymbol{\alpha} = 0 \\ \Rightarrow \left(\boldsymbol{C}^T\boldsymbol{C} + \sigma^2\boldsymbol{I}\right) \cdot \boldsymbol{\alpha} &= \boldsymbol{C}^T \cdot (\boldsymbol{s} - \bar{\boldsymbol{s}}) \end{aligned}$$

If the left-hand matrix of the above equation is non-singular, then the optimal coefficients can be recovered as

$$\boldsymbol{\alpha} = \left(\boldsymbol{C}^T\boldsymbol{C} + \sigma^2\boldsymbol{I}\right)^{-1} \cdot \boldsymbol{C}^T \cdot (\boldsymbol{s} - \bar{\boldsymbol{s}}), \tag{2.7}$$

and the optimal reconstruction of the vector $\boldsymbol{s}$ is

$$\tilde{\boldsymbol{s}} = \bar{\boldsymbol{s}} + \boldsymbol{C} \cdot \left(\boldsymbol{C}^T\boldsymbol{C} + \sigma^2\boldsymbol{I}\right)^{-1} \cdot \boldsymbol{C}^T \cdot (\boldsymbol{s} - \bar{\boldsymbol{s}}).$$

If the matrix $\boldsymbol{C}$ were defined as in equation (2.5), the solution of (2.7) is simple to find, since

$$\begin{aligned} \boldsymbol{C}^T\boldsymbol{C} + \sigma^2\boldsymbol{I} &= \left(\boldsymbol{\Lambda}_k - \sigma^2\boldsymbol{I}\right)^{1/2} \boldsymbol{U}_k^T\boldsymbol{U}_k \left(\boldsymbol{\Lambda}_k - \sigma^2\boldsymbol{I}\right)^{1/2} + \sigma^2\boldsymbol{I} \\ &= \boldsymbol{\Lambda}_k - \sigma^2\boldsymbol{I} + \sigma^2\boldsymbol{I} \\ &= \boldsymbol{\Lambda}_k \end{aligned}$$

In this particular case, the expression for the reconstruction $\tilde{\boldsymbol{s}}$ can be further simplified to

$$\tilde{\boldsymbol{s}} = \bar{\boldsymbol{s}} + \boldsymbol{U}_k \cdot \left(\boldsymbol{I} - \sigma^2\boldsymbol{\Lambda}_k^{-1}\right) \cdot \boldsymbol{U}_k^T \cdot (\boldsymbol{s} - \bar{\boldsymbol{s}}). \tag{2.8}$$

From the above equation it is then clear that the optimal reconstruction, in statistical terms, is an orthogonal projection only if the noise is zero. Otherwise, the projection along a principal direction $\boldsymbol{u}_i$ is scaled down by a factor $1 - (m-1)\sigma^2/w_i^2$ (see figure 2.4).

In the case of the combined model, however, we have

$$\boldsymbol{C}^T\boldsymbol{C} + \sigma^2\boldsymbol{I} = \begin{pmatrix} \boldsymbol{\Lambda}_{id} & \boldsymbol{C}_{id}^T\boldsymbol{C}_{xp} \\ \boldsymbol{C}_{xp}^T\boldsymbol{C}_{id} & \boldsymbol{\Lambda}_{xp} \end{pmatrix}.$$

Since the two matrices $\boldsymbol{C}_{id}$ and $\boldsymbol{C}_{xp}$ are not orthogonal, the above matrix is in general not diagonal, and we cannot simplify the reconstruction as in equation (2.8). However, we can decompose $\boldsymbol{C}$ by SVD (note that the matrices $\boldsymbol{U}$ and $\boldsymbol{W}$ are not the same resulting from the SVD of the data matrix $\boldsymbol{A}$):

$$\boldsymbol{C} = \boldsymbol{U}\boldsymbol{W}\boldsymbol{V}^T,$$

which results in the following equation for the optimal reconstruction

$$\tilde{\boldsymbol{s}} = \bar{\boldsymbol{s}} + \boldsymbol{U} \cdot \boldsymbol{W}^2 \left(\boldsymbol{W}^2 + \sigma^2\boldsymbol{I}\right)^{-1} \cdot \boldsymbol{U}^T \cdot (\boldsymbol{s} - \bar{\boldsymbol{s}}).$$

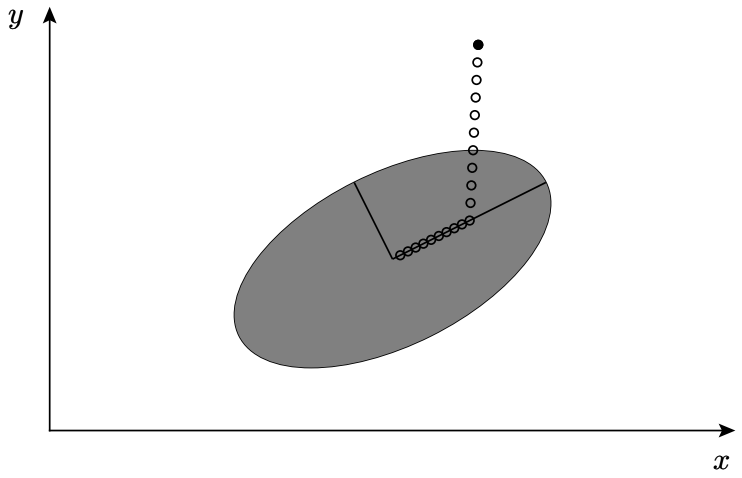


Figure 2.4: Example of PPCA reconstruction. The solid point is reconstructed with different values of noise. When the noise is null, the reconstruction is perfect, since the model has two components, as many as the dimensions of the data. Increasing the noise, the reconstruction moves more and more towards the mean. When the noise variance is equal to the variance of the smallest component, the reconstruction collapses on the axis of the largest component. At this point the smallest component cannot be included in the model any more, and the model reduces to one component. Increasing the noise again, the reconstruction will eventually coincide with the mean.

2.2 Missing Data

The estimation of the model parameters as described in the previous section relies on the fact that the centered data matrix $\boldsymbol{A}$ is complete, that is all its values are known. In our context however, it often occurs that the examples $\boldsymbol{s}_i$ have missing values, and a different method is needed.

As shown in [BVB03], it is possible to build a 3D Morphable Model from incomplete data by applying an *Expectation-Maximization* (EM) algorithm (see [DLR77]) for the estimation of the linear Gaussian model (see [Row97] and [TB99]). The EM algorithm estimates the model parameters iteratively, by computing at each iteration:

- the expected values of the latent variables $\boldsymbol{\alpha}$, given the current parameters $\bar{\boldsymbol{s}}$, $\boldsymbol{C}$ and σ^2 (Expectation- or E-step);
- the most likely model parameters given the previously estimated values of the latent variables (Maximization- or M-step).

In case of complete data, the algorithm converges to the close solution given in the previous section, so that the two methods are equivalent. However, the algorithm can be applied also to the case of incomplete data, using a generalized E-step which estimates both the latent variables and a *complete* reconstruction of the observed variables $\boldsymbol{s}_i$. Although statistically sensible, this approach has the disadvantage of yielding principal components which might present discontinuities at the boundaries between areas present in the examples and areas of missing values. We use therefore a different strategy.

In our approach, the reconstruction of the missing values is done *before* the estimation of the model parameters, during the registration. Therefore, the registered examples in the training set are always complete, and can be used to build a complete data matrix $\boldsymbol{A}$. The model parameters are then estimated from $\boldsymbol{A}$ with the close form solution presented in the previous sections. The advantage of this approach is twofold: first, the model learning is done in one step rather than iteratively as with the EM-algorithm; second, the reconstructions are continuous and have a lower generalization error than the purely statistical estimate used in the EM-algorithm.

It can however happen that some of the vertices have no or few observed values in the examples, so that one might decide to exclude them from the estimation of the model. In this case, a reduced model $(\bar{\boldsymbol{s}}, \boldsymbol{C}^\star, \sigma^2)$ can be built from $\boldsymbol{A}^\star$, a matrix obtained from the data matrix $\boldsymbol{A}$ by removing the rows corresponding to the vertices we want to exclude from the model estimation. In such a case it is still useful to have a full-dimensional generative matrix $\boldsymbol{C}$, which can be derived from the reduced generative matrix $\boldsymbol{C}^\star$ estimated from $\boldsymbol{A}^\star$. Observing that $\boldsymbol{C}^\star$ can be expressed in terms of the reduced data matrix $\boldsymbol{A}^\star$ (with

$\boldsymbol{A}^\star = \boldsymbol{U}^\star \boldsymbol{W}^\star \boldsymbol{V}^{\star T}$):

$$\begin{aligned} \boldsymbol{C}^\star &= \boldsymbol{U}^\star \cdot \left(\frac{1}{m} \boldsymbol{W}^{\star 2} - \sigma^2 \boldsymbol{I} \right)^{1/2} \\ &= \boldsymbol{A}^\star \boldsymbol{V}^\star \boldsymbol{W}^{\star -1} \cdot \left(\frac{1}{m} \boldsymbol{W}^{\star 2} - \sigma^2 \boldsymbol{I} \right)^{1/2} \\ &= \boldsymbol{A}^\star \boldsymbol{V}^\star \cdot \left(\frac{1}{m} \boldsymbol{I} - \sigma^2 \boldsymbol{W}^{\star -2} \right)^{1/2} \end{aligned}$$

we can define the full-dimensional matrix $\boldsymbol{C}$ as a linear combination of the examples in $\boldsymbol{A}$:

$$\boldsymbol{C} = \boldsymbol{A} \boldsymbol{V}^\star \cdot \left(\frac{1}{m} \boldsymbol{I} - \sigma^2 \boldsymbol{W}^{\star -2} \right)^{1/2},$$

with the coefficients of the combination given by the SVD on $\boldsymbol{A}^\star$.

Chapter 3

Surface Reconstruction

As discussed in section 2.1.2, it may occur that the the examples in the training set used to build a morphable model have missing values. This poses a problem, since the learning algorithm assumes that the training examples are complete. Although it is possible to use an iterative learning algorithm which builds the model from incomplete data by estimating the statistically optimal positions of the missing vertices, this approach does not ensure that the reconstructed examples are continuous. We adopt therefore a different approach, performing the reconstruction of the missing values during the registration, so that the training examples are in fact complete, and the standard learning algorithm can be applied.

We describe here the reconstruction method independently from its use in the registration algorithm, which will be treated in the next chapter. After a summary of the related works, we describe the statistical reconstruction method applied to registered 3D faces, already presented in [BMVS04]. This approach uses the same inference rule exploited in the iterative learning algorithm mentioned above, and presents the same discontinuity problems. Our method combines the statistical reconstruction with a variational approach, presented in [PGB03] for image editing. As we will show, the result of the statistical reconstruction can be incorporated as a sort of guidance for the solution of the variational problem, which will approximate the gradient of the former while at the same time ensuring the continuity at the boundaries of the reconstructed area. Moreover, a comparison of our method's results with the ones of the statistical reconstruction shows that our method performs better, in terms of generalization error, than the statistical method.

3.1 Related Work

A wide range of methods for surface reconstruction are based on a variational approach. With this type of approach, a certain functional is defined over the whole domain of the surface, both where the surface is known and where it is

missing. The reconstruction is defined as the surface minimizing the functional under a set of boundary constraints, therefore yielding a variational problem. The variational problem is then transformed to a partial differential equation (PDE), which is discretized over a polygonal mesh and solved as a sparse (typically non-symmetric) linear system. Typical choices for the functional of the surface are the membrane and the thin-plate energy, as in [Lie03], which are coupled to, respectively, C^0 and C^1-continuity. Other, more exotic choices of the functional can be found in the literature, as the Willmore energy in [CDD$^+$04], which also ensure C^1-continuity at the boundaries. Note that the discretization of the PDE requires the topology of the polygonal mesh to be defined also in the missing areas, which is not the case if they correspond to actual holes in the data; it might be therefore necessary to preliminarily identify and triangulate the holes. However, this is not necessary if the surface has been registered against a template surface as in our method or in [SK02].

The problem of the missing topology can also be avoided by defining the surface implicitly rather than explicitly, as a level set of a function defined in $\mathbb{R}^3$. Although the invalid regions have still to be identified, no triangulation is needed, since the reconstruction of the surface yields both its shape and its topology. In [DMGL02] the surface is implicitly defined using a clamped signed distance function, and a binary-valued function discriminates the valid regions from the invalid ones. The reconstruction is obtained by a diffusion of both functions in the invalid regions. A more sophisticate approach uses anisotropic diffusion ([VCBS03], an extension to surfaces of the image inpainting method presented in [BSCB00]). Also using implicit functions is the method of [ZOF01], which minimizes the L_1-norm, measured on the implicit surface, of a distance function from the valid data. The method is in fact intended for surface reconstruction, but hole filling descends as a side effect.

Independently from the functional used and the type of surface representation (implicit or explicit), all the methods discussed so far share a common problem. The energy they minimize is not necessarily related to the reconstruction error in the invalid region, and in fact, the solution depends only on a small region surrounding the hole, if not simply on its boundary. Since all other information about the surface is discarded, it is difficult for such a method to obtain convincing results if the invalid surface has a complex structure. Sharf et al. ([SACO04]) propose a method that uses the valid surface to predict the structure of the invalid region: using an implicit representation for the surface, the invalid voxels are filled in a multi-resolution approach with examples extracted from the available surface. The choice depends on the context of the voxel, defined by its valid neighboring voxels. However interesting, this method can still fail to reconstruct a surface patch whose features are not present elsewhere in the known surface.

3.2 Statistical Reconstruction

We begin now by describing the statistical reconstruction method, which has been already presented in [BMVS04]. As customary, we denote by $\boldsymbol{S} \in \mathbb{R}^{n\times 3}$ the matrix holding the vertices positions of an example, and by $\boldsymbol{s} = vec(\boldsymbol{S}) \in \mathbb{R}^{3n}$ its vector representation. Let us assume now that only p vertices of the example mesh are known, and that we want to reconstruct the positions of the remaining vertices as accurately as possible. The positions of the known vertices are held by the shape vector

$$\boldsymbol{s}^\star = vec(\boldsymbol{P}\cdot\boldsymbol{S}),$$

where $\boldsymbol{P} \in \mathbb{R}^{p\times n}$ is a matrix which selects the rows of $\boldsymbol{S}$ corresponding to the known vertices[1]. Since the mesh has been registered, the selection matrix $\boldsymbol{P}$ is known. The reconstruction problem consists of finding an estimate $\tilde{\boldsymbol{s}}$ of the complete shape which, given the incomplete shape $\boldsymbol{s}^\star$ and the selection matrix $\boldsymbol{P}$, is as close as possible to the true $\boldsymbol{s}$.

If we assume the shape vector $\boldsymbol{s}$ has been generated by a linear Gaussian model as in chapter 2, the optimal reconstruction can be obtained by finding the model coefficients $\boldsymbol{\alpha}$ which maximize the posterior probability $p(\boldsymbol{\alpha}|\boldsymbol{s}^\star)$. They can be computed in closed form in essentially the same way shown in section 2.1.2 for complete data, by setting to zero the derivative of $-\log p(\boldsymbol{\alpha}|\boldsymbol{s}^\star)$ with respect to $\boldsymbol{\alpha}$:

$$\begin{aligned}
-\frac{\partial}{\partial\boldsymbol{\alpha}}\log P(\boldsymbol{\alpha}|\boldsymbol{s}^\star) &= \frac{\partial}{\partial\boldsymbol{\alpha}}\left\{\frac{\|\boldsymbol{s}^\star-\boldsymbol{P}\cdot(\bar{\boldsymbol{s}}+\boldsymbol{C}\cdot\boldsymbol{\alpha})\|^2}{2\sigma^2}+\frac{\|\boldsymbol{\alpha}\|^2}{2}+const.\right\}\\
&= -\frac{1}{\sigma^2}\boldsymbol{C}^T\boldsymbol{P}^T\cdot(\boldsymbol{s}^\star-\boldsymbol{P}\cdot\bar{\boldsymbol{s}}-\boldsymbol{P}\boldsymbol{C}\cdot\boldsymbol{\alpha})+\boldsymbol{\alpha}=0\\
\Rightarrow\boldsymbol{\alpha} &= \left(\boldsymbol{C}^T\boldsymbol{P}^T\boldsymbol{P}\boldsymbol{C}+\sigma^2\boldsymbol{I}\right)^{-1}\cdot\boldsymbol{C}^T\boldsymbol{P}^T\cdot(\boldsymbol{s}^\star-\boldsymbol{P}\cdot\bar{\boldsymbol{s}}) \quad (3.1)
\end{aligned}$$

Letting $\boldsymbol{Q}=\boldsymbol{P}\boldsymbol{C}$ and $\bar{\boldsymbol{s}}^\star=\boldsymbol{P}\cdot\bar{\boldsymbol{s}}$, we can rewrite the above equation in exactly the same form as equation (2.7):

$$\boldsymbol{\alpha}=\left(\boldsymbol{Q}^T\boldsymbol{Q}+\sigma^2\boldsymbol{I}\right)^{-1}\cdot\boldsymbol{Q}^T\cdot(\boldsymbol{s}^\star-\bar{\boldsymbol{s}}^\star).$$

Since $\boldsymbol{Q}$ is not orthogonal, the equation is solved as in section 2.1.2 by using SVD. The matrix $\boldsymbol{Q}$ is decomposed as

$$\boldsymbol{Q}=\boldsymbol{U}\boldsymbol{W}\boldsymbol{V}^T$$

and then substituted in equation (3.1), obtaining

$$\boldsymbol{\alpha}=\boldsymbol{V}\cdot\boldsymbol{W}\left(\boldsymbol{W}^2+\sigma^2\boldsymbol{I}\right)^{-1}\cdot\boldsymbol{U}^T\cdot(\boldsymbol{s}^\star-\bar{\boldsymbol{s}}^\star).$$

The optimal model coefficients can therefore be found in exactly the same way shown in 2.1.2, provided that the SVD is applied to $\boldsymbol{Q}$ rather than to $\boldsymbol{C}$. Once

[1] $\boldsymbol{P}$ is a matrix whose elements are either zero or one; for each row and each column only one element is different from zero.

the optimal model coefficients have been found, the shape can be reconstructed as $\tilde{\boldsymbol{s}} = \bar{\boldsymbol{s}} + \boldsymbol{C}\boldsymbol{\alpha}$. We should stress again the fact that this is not an orthogonal projection, that is it will not minimize the reconstruction error of the examples used for training the data. However, as we will show in the results, such a projection does yield a lower generalization error (the expected reconstruction error over the whole data distribution).

Although statistically well-founded, this solution presents a problem due to the fact that the reconstructed shape $\tilde{\boldsymbol{s}}$ lies in the principal subspace spanned by the columns of $\boldsymbol{C}$. Therefore in general the reconstruction error $\|\boldsymbol{S}^\star - \boldsymbol{P} \cdot \tilde{\boldsymbol{S}}\|^2$ will not be zero (see figure 3.1(c)). Since our goal is to reconstruct only the missing values, we could of course combine $\tilde{\boldsymbol{S}}$ and $\boldsymbol{S}^\star$, using the former for defining the positions of the missing vertices and the latter for the known ones:

$$\tilde{\boldsymbol{S}}' = \left(\boldsymbol{I} - \boldsymbol{P}^T\boldsymbol{P}\right)\tilde{\boldsymbol{S}} + \boldsymbol{P}^T\boldsymbol{P}\boldsymbol{S}^\star.$$

Unfortunately, this raises another problem, already mentioned in chapter 2: because of the residual reconstruction error, the resulting surface might present discontinuities at the boundaries between known and missing vertices (see figure 3.1(d)). In the following sections we will discuss a different reconstruction method which overcomes this problem.

3.3 Laplace Reconstruction

It is not surprising to discover that the statistical reconstruction cannot ensure continuity, since it uses a model based only on the covariance of the vertices, while no information on the neighborhood of a vertex is used. We derive now a method which uses this information by considering instead of the vertices the continuous 2D surface $\boldsymbol{f} : \mathcal{S} \subset \mathbb{R}^2 \rightarrow \mathbb{R}^3$ on which they lie, where we denote by $\mathcal{S}$ the two-dimensional parameterization domain. We will first use this continuous setting to motivate our method, and then show how it is applied to a discrete mesh.

Like $\boldsymbol{S}$, the surface $\boldsymbol{f}$ is not completely defined. Let us denote by $\Omega \subset \mathcal{S}$ the subset of the parameterization domain where the surface is undefined, and by $\partial\Omega = S - \Omega$ its complement. Given the known surface $\boldsymbol{f}^\star : \partial\Omega \rightarrow \mathbb{R}^3$, equivalent to $\boldsymbol{S}^\star$ in the discrete setting, we define the reconstruction problem as the one of finding a surface $\boldsymbol{f}$ which satisfies the constraint $\boldsymbol{f}|_{\partial\Omega} = \boldsymbol{f}^\star$ and minimizes a given cost function. In theory, it would be desirable to minimize the generalization error, but in practice we will choose a simpler cost function, and be satisfied with showing that the results yield a lower generalization error than the statistical reconstruction.

Let us begin with the following variational problem:

$$\boldsymbol{f} = \min_{\boldsymbol{f}} \iint_\Omega \|\nabla \boldsymbol{f}\|^2 \quad \text{with} \quad \boldsymbol{f}|_{\partial\Omega} = \boldsymbol{f}^\star, \tag{3.2}$$

where we minimize in Ω the *membrane energy* of the surface. Note that, without the constraints, the optimal solution will have in each point a gradient equal to

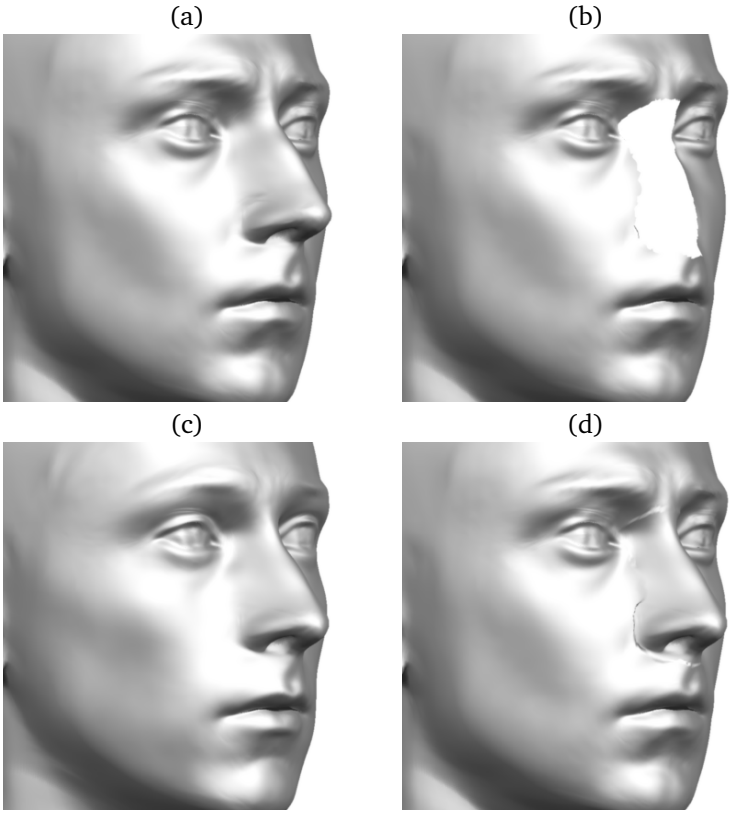


Figure 3.1: Statistical reconstruction of missing vertices. Given the registered head in (a), we removed its nose (b) and reconstructed it via the statistical method described in section 3.2. The reconstruction of the whole head is shown in image (c), while in image (d) we show the combination of the known vertices from (b) and the reconstruction of the missing ones from (c). The discontinuities at the boundary are evident.

zero, that is it will be a plane; the constraints in fact forces this planar surface to stretch such that it fits $\boldsymbol{f}^\star$ at the boundaries between Ω and $\partial\Omega$. The problem above can be shown to be equivalent to its corresponding Euler-Lagrange equation, which in this case is a Laplace PDE with Dirichelet boundary conditions:

$$\Delta \boldsymbol{f}|_\Omega = 0 \quad \text{with} \quad \boldsymbol{f}|_{\partial\Omega} = \boldsymbol{f}^\star \tag{3.3}$$

where $\Delta = \nabla \cdot \nabla = \partial_x^2 + \partial_y^2$ is the Laplace operator.

Since in practice we do not work with the continuous surface $\boldsymbol{f}$, but rather with a 3D mesh approximating it, we have to define a discrete approximation of equation (3.3). To this aim, we use what is known in Computer Graphics as the *umbrella* operator ([KCVS98]), which approximates the Laplace operator at each vertex $\boldsymbol{p}_i$ of the 3D mesh as:

$$\Delta \boldsymbol{f} \sim \mathcal{U}(\boldsymbol{p}_i) = \frac{1}{|\mathcal{N}_i|} \sum_{j \in \mathcal{N}_i} (\boldsymbol{p}_j - \boldsymbol{p}_i)$$

where $\mathcal{N}_i$ are the indices of the neighbors of $\boldsymbol{p}_i$ and $|\mathcal{N}_i|$ is their number. We define now a sparse $n \times n$ matrix $\boldsymbol{K}$, with values different from zero only if they correspond to an edge of the mesh. In this case, if $\{i, j\}$ is an edge of the mesh, we set $K_{ij} = 1/|\mathcal{N}_i|$. It is easy to verify that with $\boldsymbol{K}$ defined in this way, we can approximate the action of the Laplace operator on the whole surface as

$$\mathcal{U}(\boldsymbol{S}) = (\boldsymbol{K} - \boldsymbol{I}) \cdot \boldsymbol{S}.$$

If we denote by $\boldsymbol{\Lambda}$ a diagonal $n \times n$ matrix, with $\Lambda_{ii} = 0$ if the i-th vertex is missing and $\Lambda_{ii} \gg 1$ otherwise, we can write the discrete approximation of equation (3.3) as

$$(\boldsymbol{K} - \boldsymbol{I}) \cdot \boldsymbol{S} + \boldsymbol{\Lambda} \cdot (\boldsymbol{S} - \boldsymbol{S}^\star) = 0. \tag{3.4}$$

The above equation defines a sparse linear system, which can be efficiently solved with standard algorithms (in our implementation we used the UMFPACK library, [Dav04]).

The reconstruction obtained by solving the system (3.4) is still not satisfactory, as shown in the left image of figure 3.2, since the missing surface is obtained only minimizing the membrane energy. In the next section, we explain how to improve the solution (and obtain the reconstruction in right image of figure 3.2) by incorporating the result of the statistical reconstruction into a variational problem similar to (3.2).

3.4 Poisson Reconstruction

As we already remarked, the variational problem of equation (3.2) looks for a solution which in Ω is as close as possible to a plane, since it minimizes $\|\nabla \boldsymbol{f}\|^2$. As a result, we saw in figure 3.2 that the shape of the nose is not correctly reconstructed. Let us now assume that we have a better guess for the gradient

Figure 3.2: Variational reconstructions of missing vertices. On the left, the Laplacian reconstruction cannot recover the shape of the nose. The Poisson reconstruction on the right, however, yields a good approximation of the original, while at the same time ensuring continuity at the boundaries.

of $\boldsymbol{f}$, given by a *guidance field* $\boldsymbol{g}$ defined on Ω. We can then modify equation (3.2) as follows

$$\boldsymbol{f} = \min_{\boldsymbol{f}} \iint_{\Omega} \|\nabla \boldsymbol{f} - \boldsymbol{g}\|^2 \quad \text{with} \quad \boldsymbol{f}|_{\partial\Omega} = \boldsymbol{f}^\star,$$

in order to obtain a solution which minimizes the difference between its gradient and $\boldsymbol{g}$, while at the same time satisfying the boundary constraints. The Euler-Lagrange equation of the variational problem above is similar to equation (3.3), and is a Poisson PDE with Dirichelet boundary conditions:

$$\Delta \boldsymbol{f}|_{\Omega} = \nabla \cdot \boldsymbol{g} \quad \text{with} \quad \boldsymbol{f}|_{\partial\Omega} = \boldsymbol{f}^\star \tag{3.5}$$

where $\nabla \cdot \boldsymbol{g} = \partial_x g_x + \partial_y g_y$ is the divergence of the guidance field. A further simplification occur if the guidance field is itself obtained as gradient of a surface $\tilde{\boldsymbol{f}}$ defined on $\mathcal{S}$, that is $\boldsymbol{g} = \nabla \tilde{\boldsymbol{f}}$. In this case the PDE of (3.5) becomes

$$\Delta \boldsymbol{f}|_{\Omega} = \nabla \cdot \nabla \tilde{\boldsymbol{f}} = \Delta \tilde{\boldsymbol{f}},$$

and if we let $\boldsymbol{d} = \boldsymbol{f} - \tilde{\boldsymbol{f}}$, the continuous problem becomes

$$\Delta \boldsymbol{d}|_{\Omega} = 0 \quad \text{with} \quad \boldsymbol{d}|_{\partial\Omega} = \boldsymbol{f}^\star - \tilde{\boldsymbol{f}}.$$

Denoting by $\tilde{\boldsymbol{S}}$ the discrete equivalent of $\tilde{\boldsymbol{f}}$, and setting $\boldsymbol{D} = \boldsymbol{S} - \tilde{\boldsymbol{S}}$, the discrete approximation is

$$(\boldsymbol{K} - \boldsymbol{I}) \cdot \boldsymbol{D} + \boldsymbol{\Lambda} \cdot \left(\boldsymbol{D} + \tilde{\boldsymbol{S}} - \boldsymbol{S}^\star \right) = 0. \tag{3.6}$$

Equation (3.6) is interesting because a good approximation $\tilde{\boldsymbol{S}}$ of the unknown surface can be obtained from the statistical reconstruction of section

3.2. In practice, what we do by plugging the statistical reconstruction in equation (3.6), is to estimate its residuals; the residuals can then be added to $\tilde{\boldsymbol{S}}$, yielding a surface (see the right image of figure 3.2) without discontinuities and, as proved in the next section, smaller generalization error.

3.4.1 Laplace Operator

We conclude the description of the reconstruction method with the discussion of an alternative discrete approximation of the Laplace operator. In section 3.3 we defined the approximation at a vertex of a 3D mesh with the umbrella operator

$$\Delta \boldsymbol{f} \sim \mathcal{U}(\boldsymbol{p}_i) = \frac{1}{|\mathcal{N}_i|} \sum_{j \in \mathcal{N}_i} (\boldsymbol{p}_j - \boldsymbol{p}_i) .$$

However, a more general discrete approximation can be defined ([Tau95]) as

$$\Delta \boldsymbol{f} \sim \sum_{j \in \mathcal{N}_i} w_{ij} (\boldsymbol{p}_j - \boldsymbol{p}_i) \tag{3.7}$$

where the coefficients w_{ij} are positive numbers which sum up to one for a given i. The matrix $\boldsymbol{K}$ can be defined exactly as shown for the umbrella operator, by setting $K_{ij} = w_{ij}$ for each edge $\{i, j\}$.

A general way of choosing the coefficients is by means of a set of positive scalars $\phi_{ij} = \phi_{ji}$ defined on the edges of the mesh:

$$w_{ij} = \frac{\phi_{ij}}{\sum_{k \in \mathcal{N}_\backslash} \phi_{ik}}.$$

The simplest choice is to set $\phi_{ij} = 1$, by which the generalized definition (3.7) reduces to the umbrella operator. Another common choice is to set

$$\phi_{ij} = \|\boldsymbol{p}_i - \boldsymbol{p}_j\|^{-1},$$

so that close neighbors are weighted more than distant ones in the approximation. Yet another possibility is to set ϕ_{ij} to the inverse of the variance, over the set of training examples, of the length of the edge $\{i, j\}$. In the following section we will compare the results obtained with such a choice w.r.t. the use of the umbrella operator.

3.5 Results

As we have seen, the Poisson reconstruction is continuous by construction. In order to ensure that the method does have a generalization error at least comparable with the statistical reconstruction, we tested it on a set of 200 registered heads. We designed the experiment to be similar to a cross-validation test: we iteratively split the set of examples in a training set and a test set, removed the

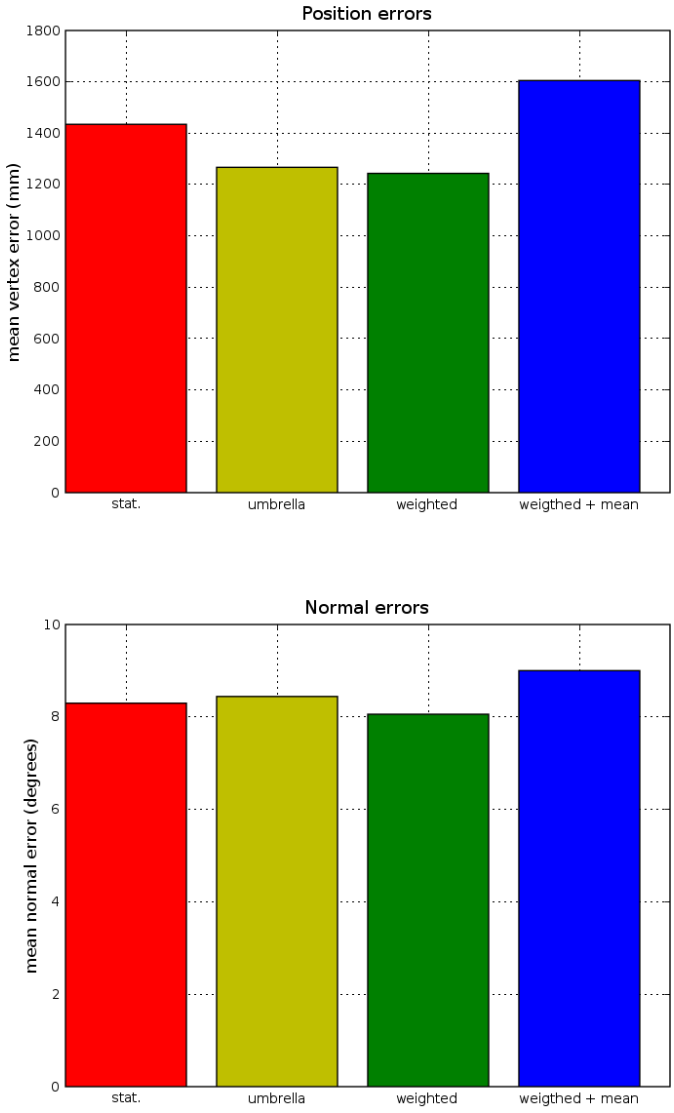


Figure 3.3: Histograms of the reconstruction errors for different methods.

noses of the heads in the test set, and reconstructed them using the model built with the (complete) examples in the training set.

We ran the experiment on four different reconstruction methods:

- the statistical reconstruction of section 3.2;
- the Poisson reconstruction of section 3.4, with the discrete Laplace operator defined using the variance of the edges lengths;
- the Poisson reconstruction using the umbrella operator;
- the Poisson reconstruction using the mean shape rather than the statistical reconstruction as guidance.

The results are summarized in the histograms of figure 3.3. First, we can conclude that the the Poisson reconstruction method yields a lower generalization error than the statistical reconstruction, both in terms of vertices positions and in terms of normals directions. Second, the results show that using the statistical reconstruction as guiding surface does contribute significantly to the result. Finally, the Laplacian operator defined in 3.4.1 does improve the results, but only marginally.

In figure 3.4 we show the worst and best results, in term of vertex position error and normal direction error, and although the latter are of course nearly indistinguishable from the originals, the former show clear differences. The different performance is probably related to the different likelihoods of the test data with respects to the distribution estimated from the training sets. Clearly, if a test head deviates significantly from the distribution estimated from the training data, its reconstruction will be worse than for a more normal head. It should however be noted how the reconstructed noses, even in the worse cases, fits perfectly in the rest of the face.

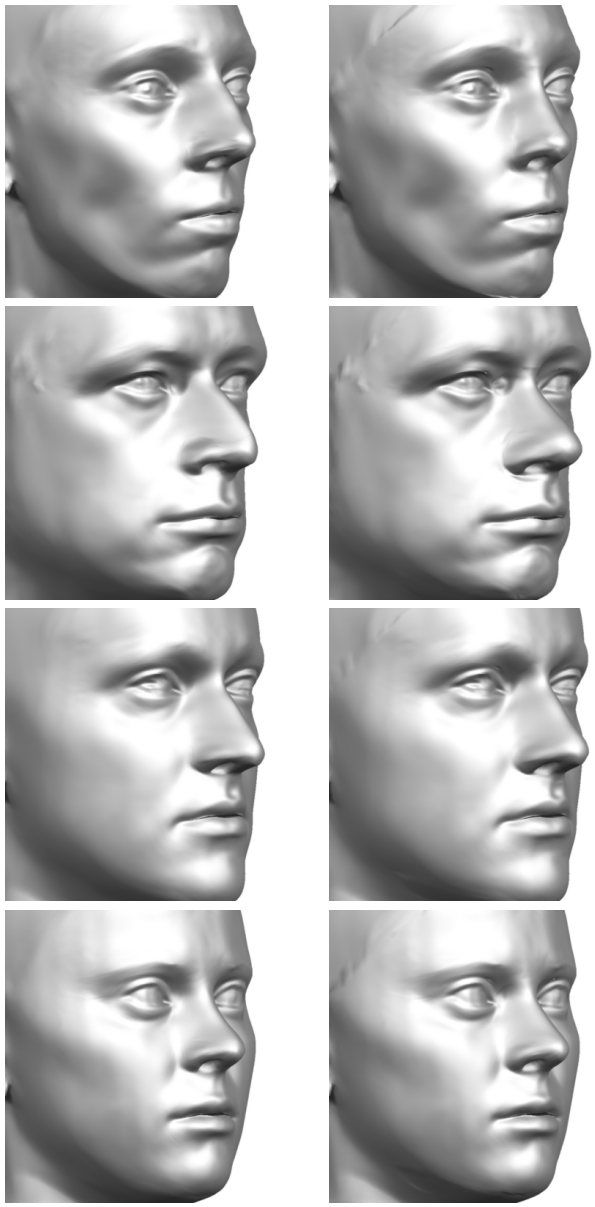


Figure 3.4: Reconstruction results with maximum/minimum errors. On the left column the originals, on the right the reconstructions. The first two rows show the results with maximum error on the vertices position (first row) and the normals directions (second row). The last two rows shows the results with minimum error on the vertices position (third row) and the normals directions (fourth row).

Chapter 4

Registration

In this chapter we present the algorithm we use registering 3D scans of human faces with arbitrary identity and expression. Our algorithm presents the following novel characteristics:

- A unified processing of faces with arbitrary identity and expressions.
- A statistically consistent reconstruction of the missing data.
- Robustness with respect to errors in the correspondence.

Unified Processing. Some very efficient methods for registering 3D scans of human faces have already been published, e.g. [BV99] and [ZSCS04], as well as for registering scans of full bodies ([ACP03]). However, the registration of data with varying identities and of data with varying expressions are typically treated separately. The method of [ZSCS04], for instance, is applied to dynamic sequences of expressions scans acquired from a single subject. On the other hand, the registration algorithm of [BV99] has been originally applied to scans with varying identity; a modified version of it ([BBPV03]) has been later used to register scans with varying expressions only. Our algorithm can be applied to 3D face scans with arbitrary identity and expressions, which makes it suitable for the applications where no such prior knowledge is available (e.g. recognition).

Reconstruction of Missing Data. The input data of the registration algorithm are typically incomplete, in two senses: first, it is quite normal that they present holes in the surface, and second, they might not cover completely the area represented by the reference model. In previous methods this problem is either not considered or it is addressed from a purely geometric point of view. In the first case, the data are pre-processed in order to fill the holes, and eventually the reference model is chosen in such a way as to be sure that it represents only the area present in the input data. In the second case, the registered surface in the missing areas is recovered by imposing on it some geometric constraint (e.g. smoothness). In both cases the recovered surface is not necessarily the most likely reconstruction of the original, missing one, and this is a clear drawback if the results of the registration have to be used to build statistical models

of human faces. In our registration algorithm the reconstruction of the missing areas takes into account not only its geometric properties but also its likelihood w.r.t. the available data.

Robustness. A further novelty of our algorithm is related to the estimation of the *correspondence*. This term, which we will often encounter in the rest of the chapter, refers to the correspondence established between vertices of the reference and points on the novel surface. Such a relation is necessary to ensure that the features of the reference match the features of the registration result, and its estimation is central for the whole registration process. In previous methods, the relative importance in the registration process of the correspondence – for instance with respect to the smoothness of the registration result – is globally fixed. As a result, in areas where the quality of the correspondence is locally good, its relative weight might be too low, and consequently important information might be discarded; or the converse might occur where the correspondence quality is locally bad but its globally fixed weight too high, therefore introducing errors in the registration results. By setting the relative importance of the correspondence locally, depending on an assessment of its quality, our algorithm retains as much correspondence information as possible, while at the same time being robust with respect to errors in its estimation.

After a brief review of the related work in this area, we will give an overview of the whole registration process. This is followed by a detailed description of the two main tasks involved: the estimation of the correspondence between the reference and the novel input, and the computation of the registration result as the solution of an optimization problem. The chapter is closed by a section reporting the registration results.

4.1 Related Work

We begin our review with the algorithm proposed by [SL94], used for registering arbitrary 3D surfaces. The authors modeled the surface deformation with an extension to 3D of the B-splines, and framed the registration problem as one of finding the optimal deformation parameters minimizing an energy made up of two terms. A first term took into account the distance between the transformed point of the novel mesh (in this paper is the novel mesh that is deformed to match the reference) from the reference surface. A second term accounted for the probability of the parameters of the transformation, in order to regularize the deformation.

This approach to the registration problem as the minimization of an energy including a regularization term is quite common. The use of the regularization term provides the natural advantage of handling missing data and inconsistencies in the correspondence. We will now briefly describe some of the papers using this approach for the registration of 3D face scans.

In [MGR00], the authors use a similar scheme to adapt a coarse reference head to novel 3D scans. In this case the transformation is modeled via a displaced subdivision, but the energy minimized is essentially the same as in

[SL94], including a distance term and a regularization term, plus a term depending on the positions of manually placed landmarks. Naturally the exact forms the energy terms are different from the one of [SL94], and in particular the regularization term is designed to favor planar surfaces.

A more recent example is the work of [ACP03], where the registration is applied to 3D scans of full human bodies with fixed pose. In this case the transformation is modeled as a set of affine transformations of the reference vertices, and the energy is again made up of a distance term, a regularization term, and a term depending on manually placed landmarks. In this case, however a correspondence is explicitly estimated using an iterative closest point (ICP) algorithm which is embedded in the optimization scheme. The methods of [SP04, VBPP05] follow a similar scheme.

Another algorithm applied to 3D scans of full bodies has been presented in [ASP+05], where the authors take a quite different approach by framing the registration problem in a probabilistic context. The correspondence is formally defined as a discrete assignment of each point of the novel mesh to a point in the reference with a probability given by the joint probability of all the vertex correspondences. On their turn, the probability of a single vertex correspondence depends on two factors. First, the probability of a vertex to correspondence to another is related to how well their signatures (an encoding of the features of the surface patch around the vertices) match. Second, the probability is conditioned on the correspondences of the neighboring vertices; this penalizes the correspondences that map neighboring vertices to vertices which are far away in terms of geodesic distance. Finally, a given correspondence defines a non-rigid deformation which also has a certain probability, defined in such a way as to penalize the stretch and twist induced by the transformation on the edges of the model. Although the algorithm has been applied to full body scans, the fact that it could register data with different poses hints to the possibility of applying it to face expressions data.

The first method dealing explicitly with the problem of registering a large database of 3D scans of human faces has been [BV99], which takes an approach different from the energy minimization adopted by the papers mentioned above. In this case the correspondence was explicitly estimated, and the deformed reference was defined by resampling the novel mesh at the corresponding points. Here the core problem was the correspondence estimation, and it was solved by exploiting a 2D representation of the 3D data which allowed to estimate the correspondence using a modified optical flow algorithm. Compared to the methods previously mentioned, registering a novel mesh by resampling it at the corresponding points has the disadvantages of relying completely on the correspondence and of being unable to cope with missing data. On the other hand, the explicit estimation of a dense correspondence provides a better matching of the face features.

Another type of approach to the registration problem is based on the idea of using an initial sparse correspondence, typically manually defined, to set up a mapping between the parameterizations of the reference and of the novel mesh. A good example is the work of [PSS01], which has been applied to

different types of 3D models, faces included. This method works by first defining a common set of feature points on a set of novel meshes. The feature points define the topology of a common base mesh, and the algorithm maps each edge of it to a path of edges in the novel meshes. In this way each face of the base mesh is mapped to a patch on the novel meshes, and this establishes a natural correspondence among all the novel meshes: two points correspond it they lie in the same patch and have the same barycentric coordinates. The registration is then only a matter of refining the base mesh to the desired resolution and then resampling the novel meshes at the corresponding points.

In [KHYS02] a reference mesh for human heads is transformed using thin-plate splines to register novel 3D face scans (without expressions). The method is initialized with a sparse correspondence, then it iteratively deforms the reference with a transformation learned by the sparse correspondence and refines automatically the sparse correspondence, until convergence. The refinement of the correspondence is obtained by subdividing the mesh defined by the current landmarks and projecting the new vertices onto the novel mesh. These projections define a set of new landmarks, whose barycentric coordinates and displacements w.r.t. the surface can be used to add corresponding landmarks to the reference. Refining in this way the sparse correspondence, a more precise deformation of the reference can be computed. What the two methods have in common is the fact that the correspondence is not explicitly estimated, rather implicitly derived from the correspondence between the parameterizations which are manually defined at the beginning.

The methods to register expression data usually adopt approaches similar to the ones outlined above. In [BBPV03] the method of [BV99] is adapted to the case of expressions by using a semi-automatic bootstrapping procedure (see following section), where the reference is deformed before correspondence estimation to better fit the novel mesh. Although the deformation is learned by previously registered examples, the amount of deformation is defined by the user. The method of [ZSCS04] is applied to time sequences of 3D scans and uses therefore time-coherence information. However, the registration of the first frame of the sequences, where no such information is used, is related to our problem. As in other methods previously outlined, the authors adopt an energy minimization approach, with the energy made up of a data term and a smoothness term, and no explicit estimation of the correspondence.

A possibly interesting work is also [BOP98], where a morphable model of lips is built from a sequence of exemplar 3D points sets. The training data are sparse, obtained by tracking and reconstructing a set of markers across a sequence of images acquired with a multiple camera system, and a vertex-to-vertex correspondence is manually set. However, since the correspondence is not available for all model vertices, they also incur in the problem of missing information. In their work the problem is solved by finding the minimum-strain solution.

4.2 Method Overview

The registration problem has been defined in the introduction as the one of deforming a reference 3D mesh so that it approximates well a novel 3D mesh while at the same time matching its features. The reference 3D mesh and the novel 3D mesh are denoted as $\mathcal{R}_0$ and $\mathcal{N}$, respectively. Recalling that a 3D mesh is defined by a structure $(\mathcal{M}, \boldsymbol{S}, \boldsymbol{T})$ (see section 1.2), we can write the reference as $\mathcal{R}_0 = (\mathcal{M}, \boldsymbol{S}_0, \boldsymbol{T}_0)$, and a deformation of the reference will be any transformation of its shape and texture which leaves the topology untouched. Since we do not impose any constraint on the form of the deformations, we can think of them as displacements $\Delta\boldsymbol{S}$ and $\Delta\boldsymbol{T}$ applied to the shape and texture. The deformed reference will be therefore $(\mathcal{M}, \boldsymbol{S}_0 + \Delta\boldsymbol{S}, \boldsymbol{T}_0 + \Delta\boldsymbol{T})$.

We restate the goal of the registration as the one of finding the optimal transformation

$$\mathcal{R}_0 \to \mathcal{R}(\Delta\boldsymbol{S}, \Delta\boldsymbol{T}) = (\mathcal{M}, \boldsymbol{S}_0 + \Delta\boldsymbol{S}, \boldsymbol{T}_0 + \Delta\boldsymbol{T}),$$

which satisfies the requirements mentioned at the beginning, that is:

- the deformed reference $\mathcal{R}(\Delta\boldsymbol{S}, \Delta\boldsymbol{T})$ approximates well the novel mesh $\mathcal{N}$;
- features in $\mathcal{R}(\Delta\boldsymbol{S}, \Delta\boldsymbol{T})$ corresponds to features in $\mathcal{R}_0$.

The registration process (see also the diagram of figure 4.1) is split into three distinct steps: first, the morphable model defined in chapter 2 is used to find an approximation of the novel mesh $\mathcal{N}$; then, the approximation is used to estimate the correspondence between the reference $\mathcal{R}_0$ and the novel mesh $\mathcal{N}$; finally, the optimal deformations $\Delta\boldsymbol{S}$ and $\Delta\boldsymbol{T}$ are computed minimizing an energy which depends on the correspondence information and the approximation. Although the first two steps are performed once for both shape and texture registration, the third step differs; we will describe first the procedure for the shape registration and postpone the description of the texture registration to a later section (4.6).

Approximation. In order to obtain a more accurate estimation of the correspondence we employ a strategy known as *bootstrapping* (see [VJP97]). It is based on the idea of exploiting previous registration results to build a statistical model of the data, which can then be used to find a first approximation of the optimal deformations $\Delta\boldsymbol{S}$ and $\Delta\boldsymbol{T}$ for the current input $\mathcal{N}$. If the shape and texture of the data are modeled with the linear Gaussian models introduced in chapter 2, that is as

$$\boldsymbol{S}(\boldsymbol{\alpha}) = \boldsymbol{S}_0 + \sum_{i=1}^{k} \alpha_i \boldsymbol{S}_i \quad \text{and} \quad \boldsymbol{T}(\boldsymbol{\beta}) = \boldsymbol{T}_0 + \sum_{i=1}^{k} \beta_i \boldsymbol{T}_i. \tag{4.1}$$

a Newton descent method can be used (as described in section 4.3) to find the coefficients of the model such that $\boldsymbol{S}(\boldsymbol{\alpha})$ and $\boldsymbol{T}(\boldsymbol{\beta})$ optimally fit the shape and

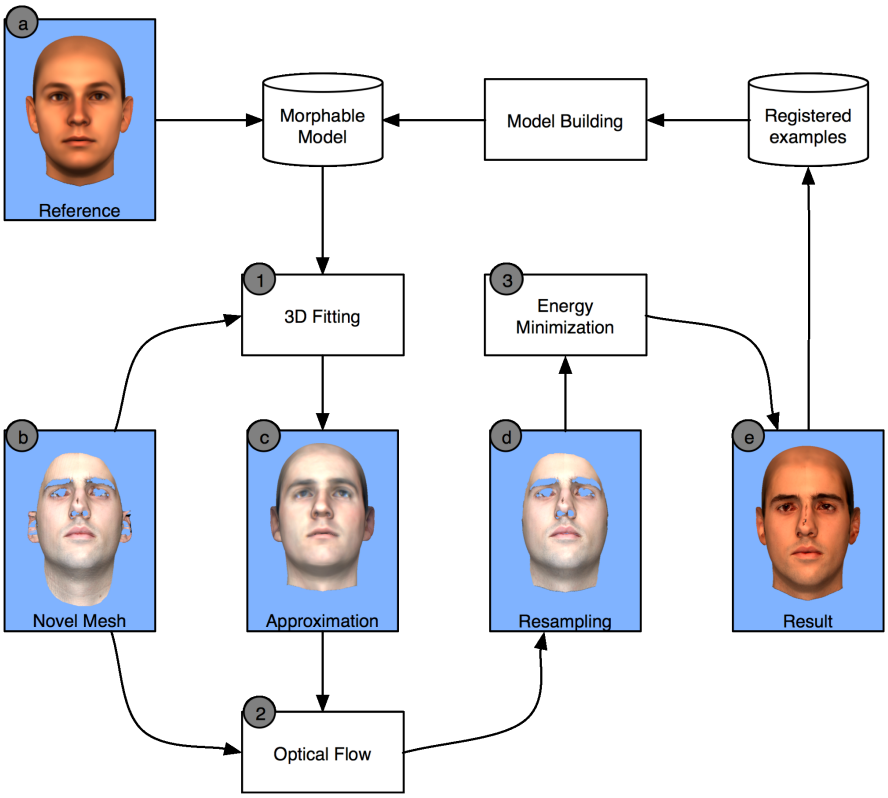


Figure 4.1: Flow diagram of the registration method. (1) The morphable model is fitted to the novel mesh (b), yielding the approximation (c). (2) Both the approximation and the input are projected to 2D and a correspondence is estimated via optical flow. Using the correspondence, the input is resampled yielding the incomplete surface (d). (3) The registration result (e) is obtained by minimizing an energy which depends on the resampling of the input. Each result of the registration increases the set of examples used to build the morphable model.

texture of $\mathcal{N}$. An example of such an approximation is shown in figure 4.1(c) (the novel mesh is shown in figure 4.1(b) and the reference in figure 4.1(a)).

Correspondence Estimation. Both the novel mesh $\mathcal{N}$ and the approximation are projected to a 2D representation, and a correspondence between them is estimated as shown in [BV99]. The correspondence between the two 2D projections defines in fact also a correspondence between the reference and the novel input. By resampling the latter at the points corresponding to the vertices of the reference we could in principle register the input, as done in [BV99]. However, as shown in figure 4.1(d), the registration result might be in general incomplete. Moreover, as we mentioned in the introduction, any error in the correspondence would be transferred to the registration result.

Energy minimization. For these reasons, the estimated correspondence is not used to directly resample the novel surface, but rather to define an energy, whose minimization yield the optimal deformation of the reference. As most of the other methods which frames the registration as an energy minimization problem, we define a cost function with a data and a smoothness term:

$$E = E_d(\mathcal{N}, \Delta\boldsymbol{S}) + E_s(\Delta\boldsymbol{S}) \tag{4.2}$$

The data term E_d measures the distance between the vertices of the deformed reference $\mathcal{R}(\Delta\boldsymbol{S})$ and their corresponding points on $\mathcal{N}$, and the smoothness term E_s penalizes deformations $\Delta\boldsymbol{S}$ for which neighboring vertices have different displacements. The smoothing induced by the second term, however, is adaptive, in the sense that its relative weight in the equation changes locally depending on the expected stiffness of the edges and the estimated accuracy of the correspondence (see section 4.5.2 for details). As stressed in the introduction, setting the relative importance of the smoothing term on a per-vertex basis improves the robustness of the registration without loosing too much of the input's details. Minimizing the energy of equation 4.2 (which corresponds to solving a sparse linear system) and registering the texture (see section 4.6), we obtain the result shown in figure 4.1(e).

4.2.1 Pre-processing

Initially the shape of the novel mesh $\mathcal{N}$ is defined in terms of the measure units and coordinate system of the particular acquisition device used. In order to make it consistent with the measure units and coordinates system of the reference, a global scaling is performed followed by a coarse alignment to the reference. The alignment is achieved by finding the optimal rigid transformation, in a least square sense, which maps a few (less than 10), manually-placed landmarks of the novel mesh $\mathcal{N}$ to the corresponding points in the reference mesh $\mathcal{R}_0$. The algorithm used (presented in [Ume91]) is described in the appendix A.1.

Figure 4.2: Projection of a 3D mesh to a 2D domain. On the left, the 3D mesh. The middle image is the projection of the texture, while the right image is a grey-coded representation of the range values. The range values are enough to recover the geometry, since the image coordinates provide the angle ϕ and the height y. The black areas in both images are the void pixels.

After the alignment, a 2D representation of the novel mesh $\mathcal{N}$ is computed. For each vertex of $\mathcal{N}$ after the alignment, its position $\boldsymbol{w} = (w_x, w_y, w_z)$ is transformed into cylindrical coordinates:

$$\begin{aligned} \phi &= \arctan \frac{w_x}{w_z} + \pi \\ y &= w_y \\ \rho &= (w_x^2 + w_z^2)^{1/2} \end{aligned}$$

The cylindrical coordinates (ϕ, y) are used to define a point (u, v) in a domain $[0, W] \times [0, H] \subset \mathbb{R}^2$ by means of the following transformation:

$$(u, v) = \left(\frac{W}{2\pi}\phi, \frac{H}{2} - ky \right)$$

where the factor k influences the vertical resolution of the 2D domain. In this way each vertex of $\mathcal{N}$ is mapped to a 2D position (u, v) and each triangle of $\mathcal{N}$ is mapped onto a triangle on the domain $[0, W] \times [0, H]$. The 2D domain results partitioned in the area covered by the triangles of $\mathcal{N}$, and the uncovered areas, also called *void* region since it does not correspond to a surface.

Each non-void pixel of the 2D domain corresponds then to a point on the surface of $\mathcal{N}$. Since in general this point will not coincide with a vertex, its shape and color are defined by a linear interpolation of their values at the vertices of the triangle containing it. The 2D representation (an example is shown in figure 4.2) is obtained by storing for each non-void pixel all the information needed to represent its corresponding point of $\mathcal{N}$. This consists of 4 scalars: the three texture colors plus the value of ρ, since ϕ and y are computed from the position (u, v) of the point.

In the following, we denote by $\boldsymbol{I}$ the $4 \times W \times H$ structure holding the shape and color information of $\mathcal{N}$ mapped to the 2D domain. The structure $\boldsymbol{I}$ is essentially a four-layered image; by $\boldsymbol{I}_{uv}$ we denote the 4D vector corresponding

to the position (u, v) in the 2D domain, and to denote the i-th value of this 4D vector we use the notation $I_{uv,i}$. Let us also define a norm on this type of structures as

$$\|\boldsymbol{I}_{uv}\|_w = \left(\sum_i w_i I_{uv,i}^2 \right)^{1/2}, \tag{4.3}$$

where the weights w_i take into account the fact that we are combining heterogeneous quantities.

Unfortunately, the data as they are obtained from the projection to 2D could still need an additional processing step, due to the fact that they will typically include structures which could disturb the registration, e.g. clothes or hairs. In the case of scans with open mouth, it is especially problematic all information of the mouth interior, which is not explicitly modeled by our reference. All these structures have to be removed at this stage.

An additional remark about the texture is needed. By representing with $\boldsymbol{I}$ the texture of $\mathcal{N}$ one is bound to the resolution of the former. Since increasing its resolution has a big impact on the computational load of the whole registration, it would not be a good idea to use high resolution when faced with high-resolution textures. A better solution is to keep the texture images, and make an additional projection to the 2D domain of the texture coordinates of $\mathcal{N}$. As I will show in a later section, in this way one can reduce to the minimum the loss in texture detail.

4.3 Approximation

Two types of transformations are applied to the refence in order to fit the input. We already introduced the first type in the overview of the method: it is the deformation induced by the morphable model, whose shape and texture are parameterized by the k-dimensional vectors $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ (equations (4.1)), where k is the number of principal components retained in the model. These linear deformations of the shape and textures are followed by a geometric transformation of the shape, parameterized by the vector $\boldsymbol{\rho}$, and a similar transformation in the color space for the texture, parameterized by the vector $\boldsymbol{\gamma}$.

The shape $\boldsymbol{S}(\boldsymbol{\alpha})$ is transformed by the combination of an anisotropic scaling with a rigid transformation, defined by a rotation matrix $\boldsymbol{R}(\boldsymbol{\rho})$ and a translation vector $\boldsymbol{t}(\boldsymbol{\rho})$. The scaling is parameterized by two scalars, say ρ_1 and ρ_2, and is defined by a matrix $\boldsymbol{U}(\boldsymbol{\rho})$ of the form

$$\boldsymbol{U}(\boldsymbol{\rho}) = \begin{pmatrix} \rho_1 & 0 & 0 \\ 0 & \rho_2 & 0 \\ 0 & 0 & \frac{\rho_1+\rho_2}{2} \end{pmatrix}.$$

The z-axis scaling factor is set to the average of the other two factors because of the relative small range of depth values in a typical acquisition, which does not allow for a precise estimation of the optimal scaling factor. Given the geometric

transformations defined above, the final shape $\boldsymbol{S}'$ is defined as

$$\boldsymbol{S}'(\boldsymbol{\alpha}, \boldsymbol{\rho}) = \boldsymbol{S}(\boldsymbol{\alpha}) \cdot \boldsymbol{R}(\boldsymbol{\rho})^T \cdot \boldsymbol{U}(\boldsymbol{\rho}) + \boldsymbol{t}(\boldsymbol{\rho})^T. \tag{4.4}$$

Note that, since $\boldsymbol{S}(\boldsymbol{\alpha})$ is a $N \times 3$ matrix, it must be right-multiplied with the geometric transformation matrices, and that (with an abuse of notation) the addition of the row-vector $\boldsymbol{t}^T$ denotes its addition to each of the N rows of the result of the matrix multiplications.

The texture $\boldsymbol{T}(\boldsymbol{\beta})$ is subject to a similar transformation in the color space, parameterized by the vector $\boldsymbol{\gamma}$:

$$\boldsymbol{T}'(\boldsymbol{\beta}, \boldsymbol{\gamma}) = \boldsymbol{T}(\boldsymbol{\beta}) \cdot \boldsymbol{M}(\boldsymbol{\gamma})^T \cdot \boldsymbol{U}(\boldsymbol{\gamma}) + \boldsymbol{a}(\boldsymbol{\gamma})^T.$$

The translation is replaced by what is called a color *offset*, defined by the vector $\boldsymbol{a}(\boldsymbol{\gamma})$; instead of the rotation we have a transformation controlling the color contrast, defined by the (non-orthogonal) matrix $\boldsymbol{M}(\boldsymbol{\gamma})$:

$$\boldsymbol{M}(\boldsymbol{\gamma}) = \gamma_1 \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} + (1 - \gamma_1) \begin{pmatrix} 0.3 & 0.59 & 0.11 \\ 0.3 & 0.59 & 0.11 \\ 0.3 & 0.59 & 0.11 \end{pmatrix};$$

and finally we denote by $\boldsymbol{U}(\boldsymbol{\gamma})$ a diagonal matrix which scales the values of the color channels – the color *gains*.

If we denote the model parameters by $\theta = (\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\rho}, \boldsymbol{\gamma})$, the goal of the fitting is to find an optimal $\theta^\star$ such that the result of the transformations above, that is the 3D mesh $(\mathcal{M}, \boldsymbol{S}', \boldsymbol{T}')$, best approximates the input $\mathcal{N}$. The optimal approximation is found by maximization of the posterior probability of the model parameters given the input $\mathcal{N}$. This is equivalent to the minimization of its log-inverse:

$$\begin{aligned} -\log P(\theta|\mathcal{N}) &= -\log \frac{P(\mathcal{N}|\theta)P(\theta)}{P(\mathcal{N})} \\ &= -\log P(\mathcal{N}|\theta) - \log P(\theta) + const. \\ &= E_d(\theta) + E_p(\theta) + const. \end{aligned}$$

where we denoted by $E_d(\theta)$ the term of the log-inverse depending on the approximation error, and by $E_p(\theta)$ the term depending on the likelihood of the model coefficients.

In practice, it is convenient to define the data term E_d in terms of the 2D projections of the approximation and of the input rather than in terms of the 3D meshes. Recalling that $\boldsymbol{I}$ is the 2D projection of $\mathcal{N}$, we will therefore define the approximation error as

$$E_d(\theta) = P(\boldsymbol{I}|\theta);$$

since the relation between $\mathcal{N}$ and $\boldsymbol{I}$ is deterministic, we can safely assume that this is equivalent to $P(\mathcal{N}|\theta)$. Denoting by $\tilde{\boldsymbol{I}}$ the 2D projection of the approximation, the approximation error is defined as a sum over all the non-void pixels:

$$E_d(\theta) = \frac{1}{\sigma_I^2} \sum_{u,v} \|\boldsymbol{I}_{uv} - \tilde{\boldsymbol{I}}_{uv}\|_w^2.$$

The squared norm $\|\cdot\|_w^2$ is the weighted sum described by equation (4.3), and the noise variance σ_I^2 has been derived from the noise variance of the model σ^2.

As we have seen in chapter 2, the prior probabilities of the model parameters $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are Gaussian distributions with mean zero. Assuming a Gaussian distribution also for the geometric and color transformations, the term E_p can be written as:

$$E_p(\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\rho}, \boldsymbol{\gamma}) = \sum_i \frac{\alpha_i^2}{\sigma_{S,i}^2} + \sum_i \frac{\beta_i^2}{\sigma_{T,i}^2} + \sum_i \frac{(\rho_i - \mu_{R,i})^2}{\sigma_{R,i}^2} + \sum_i \frac{(\gamma_i - \mu_{C,i})^2}{\sigma_{C,i}^2}$$

where the coefficients $\sigma_{\bullet,i}$ and $\mu_{\bullet,i}$ denote the standard deviations and the eventual non-zero means of the i-th element of the different parameters vectors. The coefficients $\sigma_{S,i}$ and $\sigma_{T,i}$ can be derived from the construction of the linear models for the shape and texture, but the coefficients $\sigma_{R,i}$ and $\sigma_{C,i}$ have to be chosen empirically.

Although this is not explicitly motivated by the maximization of the posterior probability of the model parameters, if some landmarks have been previously defined it is advisable to add a term $E_l(\theta)$ to the cost function, which depends on the error on the landmarks positions. This additional energy is defined in terms of the distances in 3D (that is, before the mapping to the 2D domain) between the landmarks positions in the input and in the approximation. Denoting by $\mathcal{L}$ the set of indices of the landmarks in the reference, and by $\boldsymbol{l}_i$ their corresponding positions in the input, the energy is simply the sum of squared distances:

$$E_l(\theta) = \sum_{i \in \mathcal{L}} \|\boldsymbol{s}_i'(\boldsymbol{\alpha}, \boldsymbol{\rho}) - \boldsymbol{l}_i\|^2,$$

where $\boldsymbol{s}_i'$ is the vector of elements of $\boldsymbol{S}'$ corresponding to the i-th vertex.

The minimum of the cost function is achieved employing a stochastic quasi-Newton descent. The algorithm is *stochastic* in the sense that we are not evaluating the approximation term and its derivatives for all the vertices, but only on a random sample of them (the other two terms of the cost function, however, are not affected by this). That is, at each iteration a subset of the vertices of the model is randomly sampled, and the approximation error is estimated by

$$E_d = \sum_{(u,v) \in \mathcal{I}} \|\boldsymbol{I}_{uv} - \tilde{\boldsymbol{I}}_{uv}\|^2,$$

where we denoted by $\mathcal{I}$ the set of the coordinates in the 2D domain of the sampled vertices. In order for the stochastic estimation of E_d to be unbiased, we have to sample the vertices with a probability proportional to the projected area of the adjoining triangles.

Given $\mathcal{I}$, the first derivatives of the cost function with respect to the parameters $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$, $\boldsymbol{\rho}$ and $\boldsymbol{\gamma}$ are computed and the update is computed as in a standard Newton descent algorithm, which we describe in appendix A.2. Note that for the computation of the updates a diagonal approximation of the Hessian matrix of the cost function is also needed. This is not computed on $\mathcal{I}$ at each iteration,

but it is rather estimated on a larger sample of vertices every few thousands iterations. At the same time when the Hessian is estimated, also the visibilities of the vertices are checked: since it can occur that in the mapping to 2D some of the vertices are occluded, these have not to be taken into account when estimating the Hessian or choosing the sample $\mathcal{I}$.

4.4 Correspondence Estimation

The result of the approximation implicitly defines a correspondence between the model and the input $\mathcal{N}$. Each vertex of the model is projected to a point in the 2D domain depending on the shape coefficients $\boldsymbol{\alpha}$ and the transformation parameters $\boldsymbol{\rho}$ and is therefore in correspondence with the point on the surface of $\mathcal{N}$ which has the same position in the 2D domain. However, in general the output $\tilde{\boldsymbol{I}}$ is not a perfect approximation of $\boldsymbol{I}$ (because $\mathcal{N}$ might not be in the linear span of the model), and therefore the implied correspondence is only an approximation.

As previously anticipated, in order to estimate the correspondence we apply to the input $\boldsymbol{I}$ and its approximation $\tilde{\boldsymbol{I}}$ an algorithm similar to the one used in [BV99], that is a modified version of the optical flow algorithm. The advantage of using the approximation in place of a fixed reference, as it was done in the original paper, lies in the assumptions on which the optical flow algorithm relies. As it will be clear in short, the algorithm tries to establish a correspondence between pixels in two images under the assumption of constant intensity, that is assuming that corresponding pixels have the same intensity in both views. In many applications, and ours is one of them, this is not true, and in fact different methods have been developed to relax the assumption; computing the approximation of $\boldsymbol{I}$ is a way to further reduce the problem.

The output of the optical flow algorithm will be a flow field defined over the 2D domain:

$$\Delta : [0, W] \times [0, H] \rightarrow \mathbb{R}^2$$

which will be combined with the correspondence implicitly defined by the approximation. In practice, if the i-th vertex of the approximation is projected to the point $p_i = (u_i, v_i)$ of the 2D domain, then by combining this position with the result of the optical flow we obtain as refined corresponding point:

$$p'_i = (u_i + \Delta_u(u_i, v_i), v_i + \Delta_v(u_i, v_i))$$

This array of 2D positions is used to sample the input $\boldsymbol{I}$ (and also the 2D mapping of the texture coordinates), so that for each vertex in the model we obtain an estimation of its corresponding position and color in the input $\mathcal{N}$. As explained in section 4.5.2, these estimated values are used to solve the registration problem of equation 4.2. In the next two sections we are going to describe briefly the fundamentals of the optical flow algorithm and in more details the modifications we implemented for applying the method to 3D data.

Basic Optical Flow Algorithm

The optical flow algorithm was originally designed and is still used in Computer Vision for the task of computing the motion field (or flow field) between consecutive frames of a sequence. We will first consider the continuous case of a temporal sequence of images, denoted by $I(u, v, t)$, in which the same physical point has the same intensity at different instants in time (see [HS81]). Given the assumption of constant intensity, if we denote by $(u(t), v(t))$ the trajectory of a point in the image domain, the following equation must hold:

$$I(u(t), v(t), t) = I(u(t_0), v(t_0), t_0)$$

By derivation with respect to the time, and keeping in mind that the right hand term is constant in time, one obtains the following partial differential equation (called the *optical flow constraint* equation):

$$\frac{dI}{dt} = \frac{\partial I}{\partial u}\frac{du}{dt} + \frac{\partial I}{\partial v}\frac{dv}{dt} + \frac{\partial I}{\partial t} = 0 \tag{4.5}$$

The terms du/dt and dv/dt are the components of the velocity of the physical point along its trajectory in the image domain. One can rewrite the above notation in vectorial form by denoting the velocity as $\boldsymbol{v}$:

$$\boldsymbol{v} \cdot \nabla I = -\frac{\partial I}{\partial t}. \tag{4.6}$$

This latter form puts in evidence the fact that the equation is underconstrained, since the component of the velocity field $\boldsymbol{v}$ perpendicular to the gradient ∇I is arbitrary. Different approaches are available to deal with this problem, known as *aperture* problem: a smoothness term can be introduced in the optical flow equation, or the image function $I(u, v)$ can be developed in a series of higher order terms. Another approach is to introduce a *spatial coherency* assumption, that is to assume that in the neighborhood of a pixel the velocity field will be constant (see [LK81]). With this assumption, rather than solving the equation (4.6) one minimizes for each pixel the following energy

$$E = \sum_{u,v \in R} \left(\boldsymbol{v}(u, v) \cdot \nabla I|_{(u,v)} + \frac{\partial I}{\partial t}|_{(u,v)} \right)^2, \tag{4.7}$$

where R is a neighborhood of the pixel. Deriving the above energy w.r.t. the velocity $\boldsymbol{v}$ one obtains the following linear system:

$$\boldsymbol{W} \cdot \boldsymbol{v} = \boldsymbol{b},$$

where the matrix $\boldsymbol{W}$ and the vector $\boldsymbol{b}$ are defined as

$$\boldsymbol{W} = \begin{pmatrix} \sum_R (\partial_u I)^2 & \sum_R \partial_u I \cdot \partial_v I \\ \sum_R \partial_u I \cdot \partial_v I & \sum_R (\partial_v I)^2 \end{pmatrix} \quad \boldsymbol{b} = \begin{pmatrix} \sum_R \partial_u I \cdot \partial_t I \\ \sum_R \partial_v I \cdot \partial_t I \end{pmatrix}$$

The equation (4.6) holds instantaneously at any given moment t, but when confronting the actual case of computing the motion field between two frames, we have to consider the integral over the interval of time. That is, given two frames $I_1 = I|_{t=t_0}$ and $I_2 = I|_{t=t_0+\Delta t}$, one has to solve the equation

$$\int_{t_0}^{t_0+\Delta t} \boldsymbol{v} \cdot \nabla I = -\int_{t_0}^{t_0+\Delta t} \frac{\partial I}{\partial t}.$$

By developing I in a first-order Taylor series around the point $t_0 + \Delta t/2$, one obtains the following equation:

$$\boldsymbol{s} \cdot \frac{1}{2}\left[\nabla I_1 + \nabla I_2\right] = -\Delta t \left.\frac{\partial I}{\partial t}\right|_{t_0+\Delta t/2} \approx I_1 - I_2, \tag{4.8}$$

where $\boldsymbol{s}$ is the spatial motion of a point from one frame to the other. From the equation above we can derive an optimization problem similar to the one of equation (4.7):

$$E = \sum_{u,v \in R} \left(\boldsymbol{s}(u,v) \cdot \left.\frac{\nabla I_1 + \nabla I_2}{2}\right|_{(u,v)} + I_2(u,v) - I_1(u,v) \right)^2,$$

where the gradients of the images are defined by the following finite differences:

$$\begin{aligned} \partial_u I_i &\rightarrow \frac{1}{2}\left[I_i(u+1,v) - I_i(u-1,v)\right], \\ \partial_v I_i &\rightarrow \frac{1}{2}\left[I_i(u,v+1) - I_i(u,v-1)\right]. \end{aligned}$$

The first order approximation of the time derivative of the image function I at the time $t_0 + \Delta t/2$ used in equation 4.8 is of course valid only insofar the time interval Δt is small with respect to the changes in the image. In general however, it can occur that the differences between the two frames have a larger scale than what this approximation can accommodate for. In order to overcome the problem, we adopt a *coarse-to-fine* approach (see [BAHH92]), in which the optical flow is iteratively computed at different resolutions. One starts by computing the optical flow on a low resolution version of the inputs (the coarse level), then the solution is expanded to the next higher resolution level and used to pre-warp one of the images before computing the flow again. This steps are repeated until one reaches the topmost, original resolution. We will describe the procedure in more detail in the next section.

Application to 3D Data

The algorithm for computing the optical flow on 3D data is a modification of the algorithm described in the previous section. Its pseudocode is shown at page 45.

Algorithm 1: Optical flow algorithm for 3D data. See section 4.4 for a detailed description of the single lines.

```
   Input: projections I and Ĩ
   Output: flow field Δ
1  begin
2      J, J̃ ← init features of I, Ĩ;
3      J^i, J̃^i with i = l_max ... 0 ← init pyramids of J, J̃;
4      Δ^{l_max} ← 0;
5      for l ← l_max to 0 do
6          if l < l_max then
7              ψ ← expand Δ^{l+1};
8          else
9              ψ ← 0
10         K ← warp J̃^l with ψ;
11         φ ← flow from K to J^l;
12         Δ^l ← smooth φ ∘ ψ;
13     Δ ← Δ^0;
14 end
```

Initially (line 2), the features of the input are computed. These are two 6-valued images made up of the normals and texture values. That is, for the input $\boldsymbol{I}$ (and similarly for $\tilde{\boldsymbol{I}}$) we define

$$\boldsymbol{J}(u,v) = \begin{pmatrix} \boldsymbol{n}(u,v) \\ \boldsymbol{c}(u,v) \end{pmatrix},$$

where $\boldsymbol{n}(u,v)$ is the vector field of normals, and $\boldsymbol{c}(u,v)$ the vector field of texture colors. In order to compute the normals for the 2D projections, recall that given a 3D surface defined parametrically as $\boldsymbol{s}(u,v) = (x(u,v), y(u,v), z(u,v))$ the unit normal vector is defined at each point as

$$\boldsymbol{n} = \frac{\boldsymbol{a} \times \boldsymbol{b}}{\sqrt{\|\boldsymbol{a}\|^2 \|\boldsymbol{b}\|^2 - \|\boldsymbol{a} \cdot \boldsymbol{b}\|^2}} \qquad (4.9)$$

$$\text{with} \qquad \boldsymbol{a} = \frac{\partial \boldsymbol{s}}{\partial u} \text{ and } \boldsymbol{b} = \frac{\partial \boldsymbol{s}}{\partial v}$$

The vector field $\boldsymbol{n}(u,v)$ is therefore computed by projecting the Cartesian coordinates of the 3D surface to 2D, computing (by finite differences) their first derivatives with respect to the two axis of the 2D domain, and finally applying the equation (4.9).

Given the features images $\boldsymbol{J}$ and $\tilde{\boldsymbol{J}}$, we compute the Laplacian pyramids (see [BA83]) up to a certain depth l_{max} (line 3). First, a Gaussian pyramid with l_{max} resolution levels is built, where the i-th level is obtained by Gaussian

smoothing of the $(i-1)$-level and nearest-neighbor resampling to half its size. After the Gaussian pyramid has been built, the i-th level of the Laplacian pyramid is obtained from it computing the difference between the i-th level and an expansion of the $(i+1)$-th level to the double of its size. See figure 4.4 for an example of the Laplacian and related pyramids. Note that when building the Laplacian pyramids, the normals will lose their geometric meaning; this is not a problem, since the normals are used as surface features and not for their geometric properties.

After the initialization steps, the coarse-to-fine loop is entered. At each iteration the flow field computed at the previous iteration is expanded to the current resolution (lines 6-9), and the resulting flow field ψ is used to pre-warp the reference of the current level, $\tilde{\boldsymbol{J}}^l$ (line 10). Then, a flow field φ is computed from the result of the warp, $\boldsymbol{K}$, to the input $\boldsymbol{J}^l$ (line 11). The computation of the flow field is performed by minimizing an energy with essentially the same form of equation (4.7), but since the images are vector-valued the addenda of the sum are squared norms:

$$E = \sum_{u,v \in R} \left\| \boldsymbol{v}(u,v) \cdot \nabla J|_{(u,v)} + \frac{\partial J}{\partial t}|_{(u,v)} \right\|^2 .$$

The linear system to be solved is defined therefore by scalar products, denoted by $\langle \cdot, \cdot \rangle$, between the partial derivatives:

$$\boldsymbol{W} = \begin{pmatrix} \sum_R \|\partial_u \boldsymbol{J}\|^2 & \sum_R \langle \partial_u \boldsymbol{J}, \partial_v \boldsymbol{J} \rangle \\ \sum_R \langle \partial_u \boldsymbol{J}, \partial_v \boldsymbol{J} \rangle & \sum_R \|\partial_v \boldsymbol{J}\|^2 \end{pmatrix} \quad \boldsymbol{b} = \begin{pmatrix} \sum_R \langle \partial_u \boldsymbol{J}, \partial_t \boldsymbol{J} \rangle \\ \sum_R \langle \partial_v \boldsymbol{J}, \partial_t \boldsymbol{J} \rangle \end{pmatrix}$$

The scalar product has to take into account the inhomogeneity of the elements of $\boldsymbol{J}$, and it is therefore defined as

$$\langle \partial_a \boldsymbol{J}, \partial_b \boldsymbol{J} \rangle = w_n (\partial_a \boldsymbol{n} \cdot \partial_b \boldsymbol{n}) + w_c (\partial_a \boldsymbol{c} \cdot \partial_b \boldsymbol{c})$$

where a, b are any of u, v, t, and the coefficients w_n, w_c compensate for the different measure units.

Finally, the flow fields φ (resulting from the solution of the optical flow equation) and ψ (the expansion of the flow field of the previous resolution level) are concatenated and smoothed (line 12). A smoothing of the solution to the optical flow equation proves necessary because of the high sensibility of the optical flow solution to noise in areas where there the contrast in the features is low. This effect can be easily understood by looking at the optical flow equation in one dimension, where the solution is given by:

$$v_u = -\frac{\partial_t J(u,t)}{\partial_u J(u,t)}$$

It is clear from this one-dimensional example that the smaller the values of $\partial_u J$ (that is the less contrast), the stronger the effect of the noise in it. Details of the smoothing step are given in the appendix A.3.

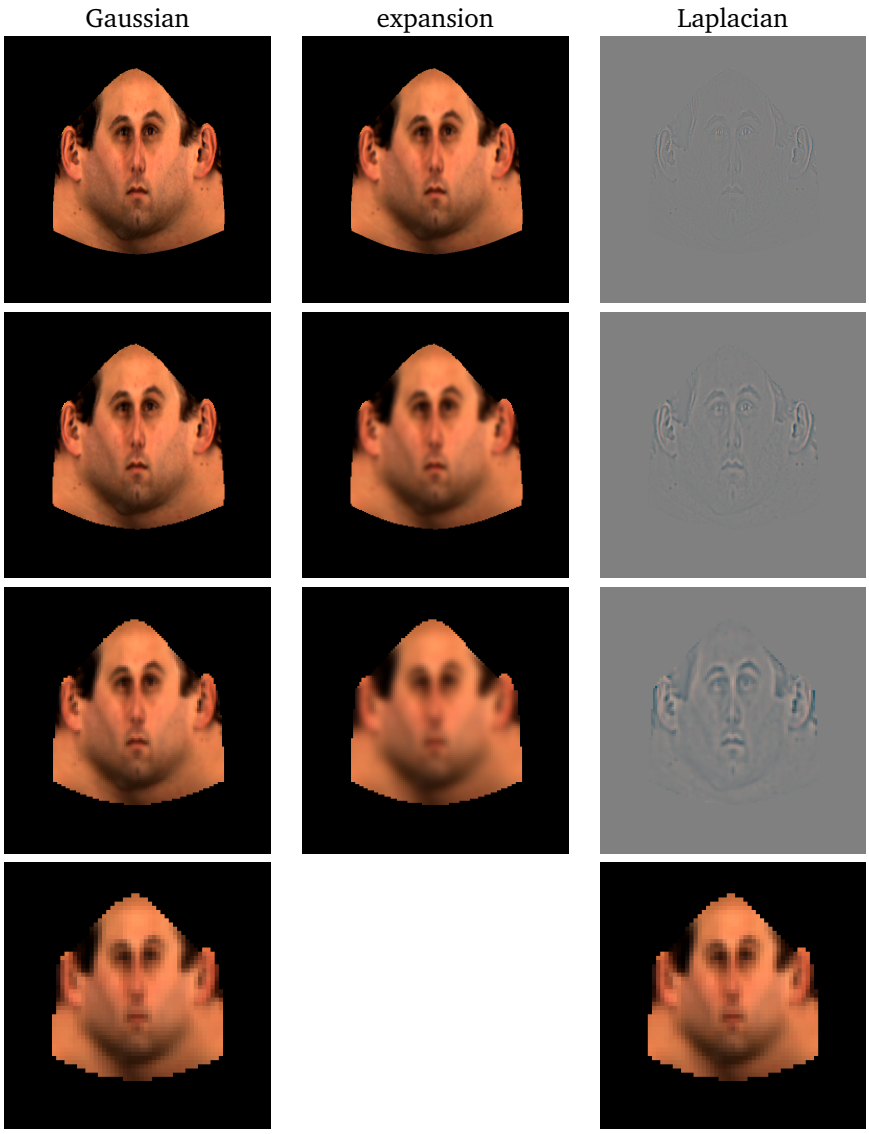


Figure 4.3: Building the Laplacian pyramid of a texture image. On the leftmost column, the Gaussian pyramid at different resolutions (all images in this figure have been scaled up to full resolution), from the highest (that is the original) 512×512 on the top row down to the lowest 64×64 on the bottom row. Each of the Gaussian images at a resolution lower than the original are then expanded to the next higher resolution (middle column). Subtracting these expanded images from the Gaussian images we obtain the Laplacian pyramid, on the rightmost column.

We conclude this section with a technical remark about a kind of numerical computation used at many steps of the algorithm: the convolution of an image with a given kernel, in order to compute its derivatives or its Laplacian pyramid. One should be aware that in the case of 3D data we have pixels in the inputs $\boldsymbol{J}$ and $\tilde{\boldsymbol{J}}$ which do not describe a surface point (void pixels), and which should not be taken into account during such type of computation. In the case of derivatives, for instance, one can substitute the central difference kernel with a forward or backward difference kernel if the pixel is on a boundary between a void and a non-void region. Then, if in J_1 the pixel (u, v) is non-void but $(u-1, v)$ is void, one can use the forward difference:

$$\partial_u J \to \frac{1}{4}\left[J_1(u+1,v) - J_1(u,v) + J_2(u+1,v) - J_2(u-1,v)\right].$$

By applying similar modifications to all kernels, one can obtain reliable results also at the boundaries between void and non-void regions.

4.5 Registration

The estimation of the correspondence between $\tilde{\boldsymbol{I}}$ and $\boldsymbol{I}$ allows us to estimate for the vertices of the model the corresponding positions on the novel input $\mathcal{N}$. In the method of [BV99] this information was directly used as output of the registration algorithm, where the registered shape and texture were sampled from $\mathcal{N}$ at the given positions. This approach has two disadvantages:

- in general, some vertex of the model might have no corresponding point in $\mathcal{N}$ and therefore the direct use of the sampled values is impossible;
- any error in the estimated correspondence can freely propagate to the registered shape and texture.

By framing the registration problem as a minimization of the energy of equation (4.2) we can address both issues.

4.5.1 Compensation of Rigid Motion

As described in the section about the pre-processing, the novel data are aligned with the reference. This alignment however is coarse and estimated only from the positions of a small set of landmarks (< 10), so that a residual rigid motion from the reference to the input will still be present. It is important to compensate for as much of this residual as possible, in order to exclude spurious rigid transformations from the morphable model built with the registration results.

An accurate estimation of the residual rigid motion can be extracted from the results of the approximation and of the correspondence estimation. The result of the 3D fitting provides a first approximation of the residual in the from of the estimated rotation $\boldsymbol{R}(\boldsymbol{\rho})$ and translation $\boldsymbol{t}(\boldsymbol{\rho})$. This approximation can be refined by extracting from the estimated flow field Δ its eventual rigid

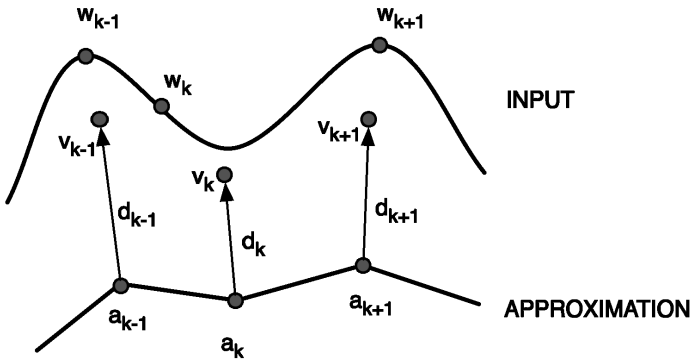


Figure 4.4: Notation used for defining the energy minimized during registration. The positions of the vertices in the approximation are denoted by $\boldsymbol{a}_i$, the corresponding sampled positions on the input by $\boldsymbol{w}_i$, and the unknown vertices positions in the solution by $\boldsymbol{v}_i$. The displacements from the approximation to the solution are denoted by $\boldsymbol{d}_i$.

components $\boldsymbol{R}(\Delta)$ (rotation) and $\boldsymbol{t}(\Delta)$ (translation). They can be estimated by applying the same alignment algorithm used in the pre-processing, but this time using as inputs the positions of the vertices in the approximation and their corresponding points on $\mathcal{N}$. Given the transformations above, the residual rigid motion is estimated as a transformation

$$\boldsymbol{v} \rightarrow \boldsymbol{R}(\Delta) \cdot [\boldsymbol{R}(\boldsymbol{\rho}) \cdot \boldsymbol{v} + \boldsymbol{t}(\boldsymbol{\rho})] + \boldsymbol{t}(\Delta)$$

In the rest of the chapter we will denote by $\boldsymbol{a}_i$ the position of the i-th vertex in the approximation without the rigid transformations (note that we are not compensating for the scaling factors), that is (see equation (4.4)):

$$\boldsymbol{a}_i = \boldsymbol{s}_i(\boldsymbol{\alpha}) \cdot \boldsymbol{U}(\boldsymbol{\rho}).$$

Each vertex $\boldsymbol{a}_i$ of the approximation corresponds to a point $\boldsymbol{p}_i$ of the novel input $\mathcal{N}$, whose position $\boldsymbol{w}_i$ after alignment with the reference is obtained as

$$\boldsymbol{w}_i = \boldsymbol{R}^T(\boldsymbol{\rho}) \cdot \left[\boldsymbol{R}^T(\Delta) \cdot (\boldsymbol{p}_i - \boldsymbol{t}(\Delta)) - \boldsymbol{t}(\boldsymbol{\rho})\right].$$

4.5.2 Energy Minimization

As anticipated in the overview of the method, the positions of the vertices in the registered output are obtained minimizing an energy depending on two terms, as in equation (4.2). The first component of the energy is a data term, depending on the distance between the solutions $\boldsymbol{v}_i$ and the sampled vertices

positions $\boldsymbol{w}_i$. Since the positions $\boldsymbol{w}_i$ are defined only for the vertices with a correspondence to the input $\mathcal{N}$, the data term is defined only for the subset $\mathcal{C}$ of such vertices:

$$E_d(\mathcal{N}, \Delta \boldsymbol{S}) = \sum_{i \in \mathcal{C}} \|\boldsymbol{v}_i - \boldsymbol{w}_i\|^2$$

The second component of the energy is a regularization term, depending on the smoothness of the deformations applied to the reference. We define it in terms of the displacements with respect to the approximation $\boldsymbol{d}_i = \boldsymbol{v}_i - \boldsymbol{a}_i$ (see figure 4.4):

$$E_s(\Delta \boldsymbol{S}) = \sum_i \sum_{j \in \mathcal{N}_i} w_{ij} \|\boldsymbol{d}_j - \boldsymbol{d}_i\|^2,$$

where the coefficients w_{ij} weights the relative importance of each edge to the smoothness energy of a vertex.

When combined together, the adaptive smoothing is achieved by weighting each term of E_d with a coefficient λ_i:

$$E = \frac{1}{2} \sum_{i \in \mathcal{C}} \lambda_i \|\boldsymbol{v}_i - \boldsymbol{w}_i\|^2 + \frac{1}{2} \sum_i \sum_{j \in \mathcal{N}_i} w_{ij} \|\boldsymbol{d}_j - \boldsymbol{d}_i\|^2 \tag{4.10}$$

Assuming for the moment that the coefficients λ_i and w_{ij} are known, the energy of the above equation can be minimized by solving a sparse linear system. Let us rewrite the equation in matrix form. In the following $\boldsymbol{V}$ denotes the $N \times 3$ matrix holding the row vectors $\boldsymbol{v}_i$, $\boldsymbol{A}$ the one holding the $\boldsymbol{a}_i$, and similarly for $\boldsymbol{W}$ and $\boldsymbol{D}$. Defining the diagonal $N \times N$ matrix

$$\boldsymbol{\Lambda} = \frac{1}{2} \begin{pmatrix} \lambda_1 & & 0 \\ & \ddots & \\ 0 & & \lambda_N \end{pmatrix} \quad \text{with} \quad \lambda_i = 0 \; \forall i \notin \mathcal{C},$$

we can write the first energy term as the trace of a matrix:

$$\begin{aligned} \frac{1}{2} \sum_{i \in \mathcal{C}} \lambda_i \|\boldsymbol{v}_i - \boldsymbol{w}_i\|^2 &= tr\left[(\boldsymbol{V} - \boldsymbol{W})^T \cdot \boldsymbol{\Lambda} \cdot (\boldsymbol{V} - \boldsymbol{W})\right] \\ &= tr\left[(\boldsymbol{A} + \boldsymbol{D} - \boldsymbol{W})^T \cdot \boldsymbol{\Lambda} \cdot (\boldsymbol{A} + \boldsymbol{D} - \boldsymbol{W})\right] \end{aligned}$$

The second term of the energy can be written in matrix form by using the same sparse $N \times N$ matrix $\boldsymbol{K}$ defined in chapter 3 for the reconstruction problem. Recall that if $\{i, j\}$ is an edge of the mesh then $K_{ij} = w_{ij}$, and $K_{ij} = 0$ otherwise. With the matrix $\boldsymbol{K}$ so defined, the second term of the energy can be written as

$$\frac{1}{2} \sum_i \sum_{j \in \Omega_i} w_{ij} \|\boldsymbol{d}_j - \boldsymbol{d}_i\|^2 = tr\left[\boldsymbol{D}^T \cdot (\boldsymbol{I} - \boldsymbol{K}) \cdot \boldsymbol{D}\right].$$

The energy of equation (4.10) can be rewritten in matrix form as

$$E = tr\left[(\boldsymbol{A} + \boldsymbol{D} - \boldsymbol{W})^T \cdot \boldsymbol{\Lambda} \cdot (\boldsymbol{A} + \boldsymbol{D} - \boldsymbol{W}) + \boldsymbol{D}^T \cdot (\boldsymbol{I} - \boldsymbol{K}) \cdot \boldsymbol{D}\right]$$

and derived with respect to $\boldsymbol{D}$ in order to find the condition for the global minimum:

$$\frac{\partial E}{\partial \boldsymbol{D}} = 2\boldsymbol{\Lambda} \cdot (\boldsymbol{A} + \boldsymbol{D} - \boldsymbol{W}) + (2\boldsymbol{I} - \boldsymbol{K} - \boldsymbol{K}^T) \cdot \boldsymbol{D} = 0$$

The above equation can be rewritten as

$$(\boldsymbol{\Lambda} + \boldsymbol{I} - (\boldsymbol{K} + \boldsymbol{K}^T)/2) \cdot \boldsymbol{D} = -\boldsymbol{\Lambda} \cdot (\boldsymbol{A} - \boldsymbol{W})$$

which is a more general form of the sparse linear system derived in chapter 3 for the reconstruction problem. It is more general because the diagonal elements of the matrix $\boldsymbol{\Lambda}$ can be set to different values for different vertices, in order to smooth adaptively the result.

Choice of the Coefficients

Deriving the linear system from the energy (4.10) makes even more sensible the choice mentioned in section 3.4.1 of setting the coefficients w_{ij} to

$$w_{ij} = \frac{\sigma_{ij}^{-2}}{\sum_{\mathcal{N}_i} \sigma_{ij}^{-2}},$$

where the σ_{ij} are the standard deviations of the edges lengths over a set of examples. For each vertex the smoothness term would become

$$\sum_{j \in \mathcal{N}_i} w_{ij} \|\boldsymbol{d}_j - \boldsymbol{d}_i\|^2 \propto \sum_{j \in \mathcal{N}_i} \frac{\|\boldsymbol{d}_j - \boldsymbol{d}_i\|^2}{\sigma_{ij}^2},$$

so that the influence of each edge on the vertex energy would be weighted by its deformations in the available examples. Such a choice can be physically motivated by the assumption that not all edges have the same stiffness: some edges will be more elastic than others, and an estimate of their elasticity can be obtained observing how they deform in the examples already available.

We conclude giving a criteria to set the values of the coefficients λ_i, which, as already observed, weight the relative importance of the data term with respect to the smoothness term. If a vertex has no correspondence ($i \notin \mathcal{C}$), we set $\lambda_i = \lambda_{min}$, with $\lambda_{min} \ll 1$. For the vertices with correspondence it would be a sensible choice to set λ_i to high values where the estimated correspondence is correct and to a lower value where it is wrong. Since the quality of the correspondence cannot be directly evaluated, we propose to measure it indirectly through another quantity. To this aim we define a measure of the smoothness of the displacement field $\boldsymbol{w}_i - \boldsymbol{a}_i$ as

$$s_i = \sum_{\mathcal{N}_i} \frac{\|(\boldsymbol{w}_j - \boldsymbol{a}_j) - (\boldsymbol{w}_i - \boldsymbol{a}_i)\|^2}{\|\boldsymbol{a}_j - \boldsymbol{a}_i\|^2}.$$

Note that the displacements $\boldsymbol{w}_i - \boldsymbol{a}_i$ are different from the displacements $\boldsymbol{d}_i$: the latter are in fact the solutions of the sparse linear system, while the former

are the displacements directly defined by the correspondence. For $i \in \mathcal{C}$, the set the coefficients λ_i to three different values depending on the values of s_i: if s_i is lower than a certain threshold, we set λ_i to a large value, so that the data term is dominant; if s_i is higher than a certain threshold, we set λ_i to λ_{min}, so that the smoothness term is dominant; if s_i is in between, we set λ_i to an intermediate value. Of course more fine-grained choices of λ_i are possible, but in the experiments we made this choice, although simple, proved to be sufficiently flexible.

4.6 Texture Processing

The process described in the previous section concerns only the registration of the shape of $\mathcal{N}$. For registering the texture we follow a different procedure, due to the different nature of the data.

With current 3D acquisition technologies, one can often obtain a high resolution texture of the scanned surface. In order to loose as less texture information as possible during the registration process, at the pre-processing stage (see section 4.2.1) we mapped the texture coordinates to the 2D domain just as the shape information. The 2D projections of the texture coordinates can be sampled exactly as the mapping of the shape, so that we can associate a texture coordinate to every vertex for which a correspondence to the novel surface is defined. The sampled texture coordinates refers to the original texture image, so that part of the registration result can be textured with the original images, without any loss of information (see figure 4.5).

However, only part of the result can be textured in this way, since in general the texture information will be incomplete like the shape information. Moreover, a consistent texturing of all the registration results is needed for generating new textures via linear combinations. Therefore two additional steps are required: first the original texture is warped to a fixed texture map, and then the missing texture is reconstructed with a diffusion algorithm.

Texture Warping.

The warping of the original texture is performed by considering the triangles of the model for which all the vertices have a correspondence to the novel surface. Each of these triangles is mapped by its vertices texture coordinates, denoted by $p_i = (u_i, v_i)_{i \in \{0,1,2\}}$, to a triangle in the model texture map; and since all its vertices have a correspondence to $\mathcal{N}$, the points p_i will correspond to three points p'_i in the original image.

To register the texture we have to find, for each pixel p of the output $\boldsymbol{T}$ contained in the triangle defined by the p_i, a corresponding position, with subpixel resolution, in the original image $\boldsymbol{T}'$. For each p in the triangle defined by p_i, we can compute its barycentric coordinates (b_1, b_2), which by definition will satisfy the following equation:

$$p = p_0 + b_1 p_1 + b_2 p_2.$$

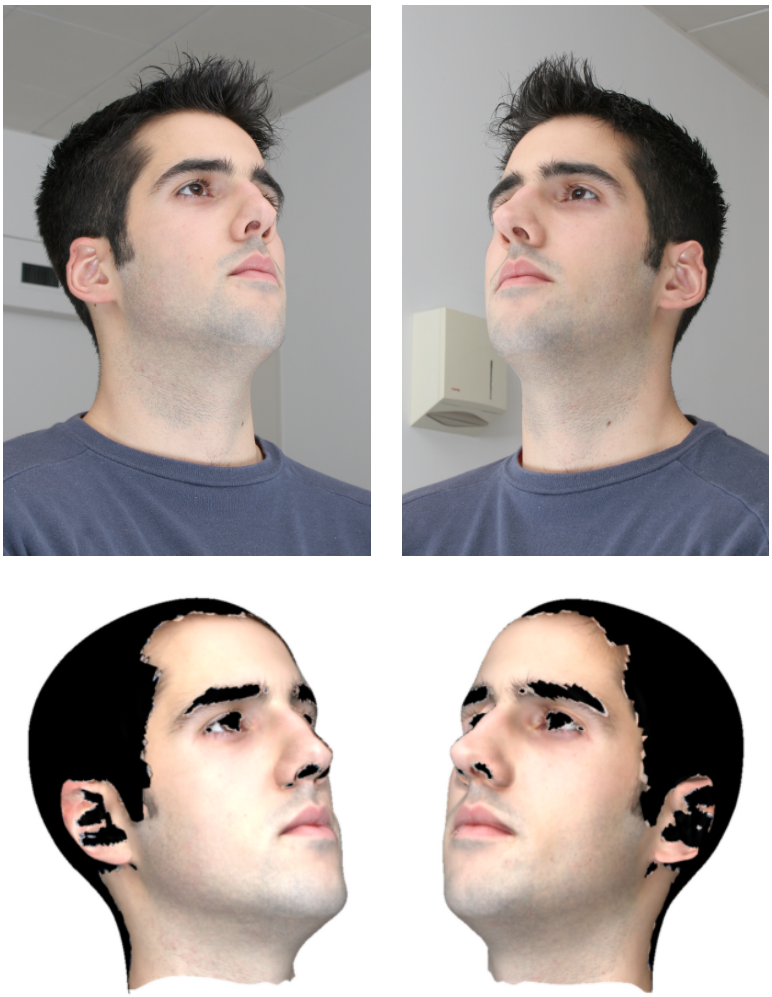


Figure 4.5: The registration result can be partially textured with the original image, as in this case where the two original texture image (top row) can be mapped on the surface of the registered shape (bottom row). The texturing however will be partial, since not all regions are present in the original images (black areas in the bottom images).

If we assume that the point $p' \in \boldsymbol{T}'$ corresponding to p has the same barycentric coordinates (b_1, b_2) with respect to the points p'_i, the registered texture is defined as:

$$\boldsymbol{T}(p) = \boldsymbol{T}'(p'_0 + b_1 p'_1 + b_2 p'_2)$$

Texture Reconstruction.

The warping of the original image defines only a part of the model texture; the rest of it has to be reconstructed.

To this aim we use a method based on a push-pull ([GGSC96]) algorithm presented in [DCOY03]. The algorithm reconstructs the missing areas of an image by iteratively downsampling it to a given scale (the *push* step) and then upsampling it back to the original scale (*pull* step), while keeping constant the known area. Repeating these two steps until convergence effectively achieves a diffusion of the known color values through the missing regions. By repeating the process with different and increasing bottom-scale resolutions, one obtains a fast and smooth approximation of the missing areas only based on the known pixel colors.

Differently than in the original setting, in our case the areas to reconstruct are not completely without information, since we have the approximation $\tilde{\boldsymbol{T}}$ previously obtained via the 3D fitting (section 4.3). It makes then sense to apply the above algorithm to the residual image $\boldsymbol{T} - \tilde{\boldsymbol{T}}$, and then add the result to $\tilde{\boldsymbol{T}}$, because in this way any structure in the approximating texture is retained.

Texture Coordinates Reconstruction.

A final issue is that some of the missing information in the warped texture might not be missing in the original texture image. This might occur if there are holes in the surface of $\mathcal{N}$, as it is the case for the eyes in figure 4.6. In this case the surface of the eyes has not been acquired, however we have its texture information and we would be interested in using it rather than having to approximate it by the method described above.

We apply the push-pull method described above to the 2D projections of the texture coordinates, in order to reconstruct their values in the areas we are interested. By reconstructing the texture coordinates before the warping, we can use the original texture information also for areas where the surface could not be reconstructed.

4.7 Results

In order to demonstrate the performance of our algorithm, we applied it to three different sets of 3D human scans: a set of 200 scans with varying identity used in [BV99]; a set of 68 scans of various expressions from 2 individuals, a subset of which has been used in [BBPV03]; and a new set of 217 scans from 31 individuals, 7 scans per subject (for each subject we took a scan without

Figure 4.6: Reconstruction of texture coordinates for the eyes. In the top image we show the texture resulting from the push-pull algorithm in the eyes region. A much better result can be obtained by applying the push-pull algorithm to the texture coordinates data, and then using the reconstructed coordinates to sample the original texture images (middle). The bottom image shows the resolution improvement when doubling the size of the registered texture.

expression and six with the basic emotional expressions). The first two sets, which we denote by D_1 and D_2, have been acquired with a Cyberware scanner, while the last set, D_3, has been acquired with a phase shift system.

We initialized the morphable model using as training examples two hand-designed 3D models of a reference human head, with closed and open mouth. The head model is complete, apart from the teeth which are not modeled, and the mouth interior is coarsely modeled as a box. We registered the data sets D_1, D_2 and D_3 one after the other, and we did not explicitly use any knowledge on the type of data during the registration. The first two datasets, however, have also been used to expand the initial morphable model, and at that stage information on the dataset class has been used. Figure 4.7 shows different examples of the training data and the results of their registration; in figure 4.8 we also show examples of heads randomly generated by the model built with the registration results from D_1 and D_2.

A quantitative evaluation of the registration results is not trivial, since we have no ground truth with which to compare them. However, one can compare the registration results of different algorithms, both by directly measuring certain properties of the results (e.g. smoothness of the surface) and of the morphable models built with them (e.g. by cross-validation); and by indirectly looking at the performance of the morphable models in specific applications (e.g. image fitting). In the following section we discuss such a comparison between the registration results of D_1 and the previous results obtained in [BV99]. The same was not possible for the results from D_2 and D_3, and we will therefore evaluate them indirectly in the context of an application, in chapter 5.1.

4.7.1 Identity Model

The evaluation of the registration results of the data set D_1 is split in three parts: (1) direct evaluation of the registered data taken singularly; (2) direct evaluation of the results as a whole, by looking at certain measures of the morphable model built from them; (3) indirect evaluation of the results as a whole, by looking at the morphable model performance in a typical application, the image fitting.

Evaluation of the Registered Heads.

In the introduction to the chapter we claimed that our algorithm offers an improved robustness to errors in the correspondence estimation, thanks to the smoothness term in the minimized energy. We first verify this claim by looking at the smoothness of the displacement field between the registered results and their average shape. Although this is not exactly the quantity minimized during registration, where we minimize the smoothness of the displacements w.r.t. to the approximation rather than the average, we reasonably expect our algorithm to show an improvement of this measure. This expectation is in fact confirmed, both for single scans which had registration problems with the pre-

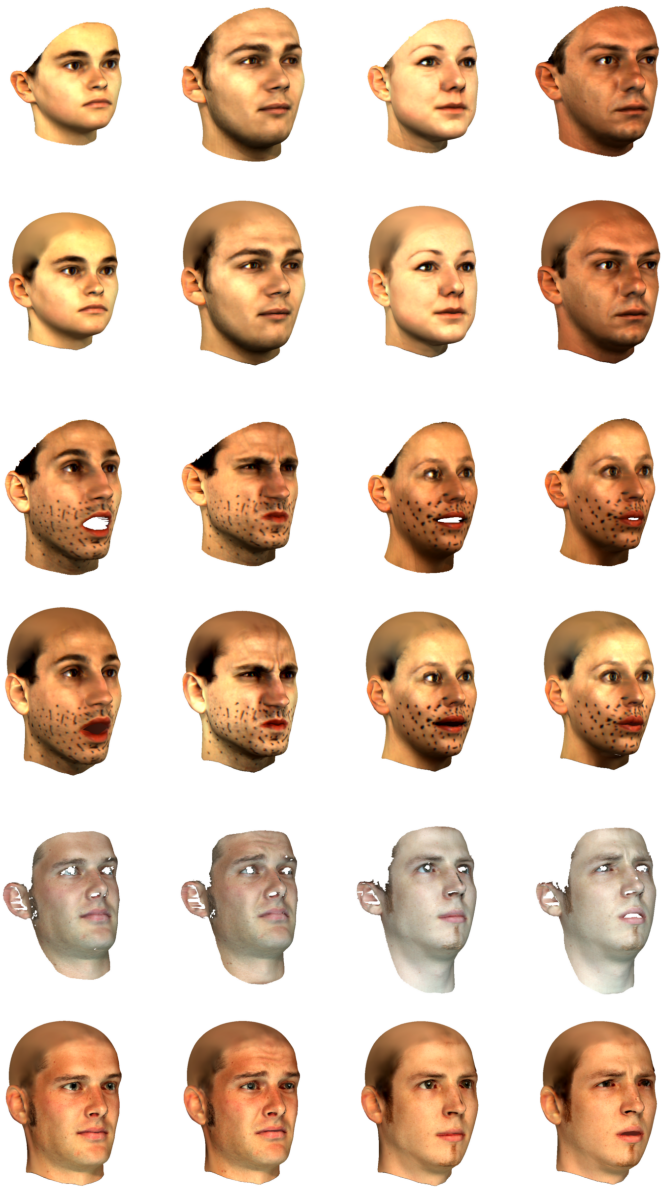


Figure 4.7: Training data and registration results. The two top rows are examples from D_1, the middle two from D_2, and the bottom two from D_3. For each pair of rows, the first shows the originals and the second the registered results.

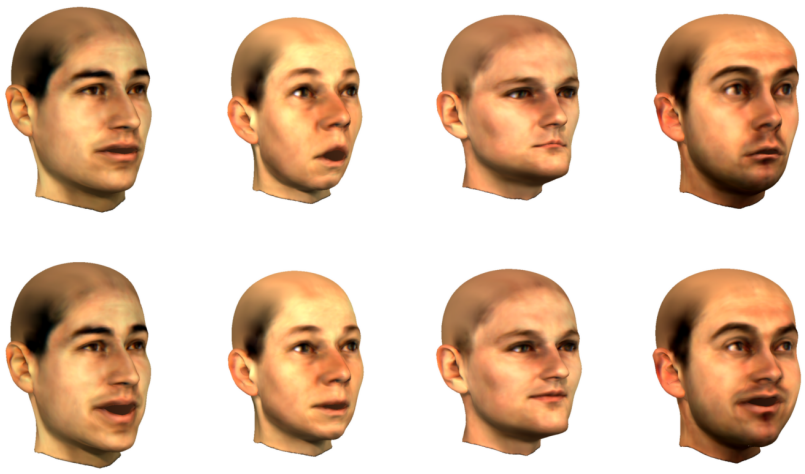


Figure 4.8: Examples of heads randomly generated by the model built from D_1 and D_2.

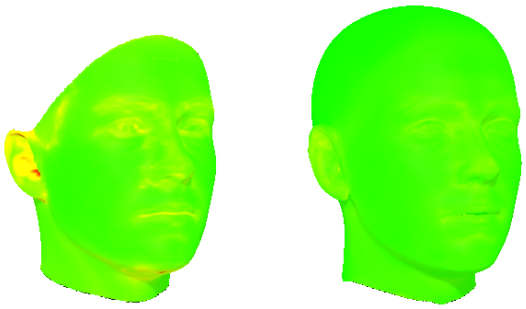


Figure 4.9: Comparison of the mean displacement field smoothness. The images are a color-coded rendering (red is lower, green higher smoothness) of the mean per-vertex smoothness values. On the left the result from the registration of [BV99], and on the the right from our algorithm. The averages over all vertices are respectively 0.204 and 0.138, with standard deviations of 0.182 and 0.086.

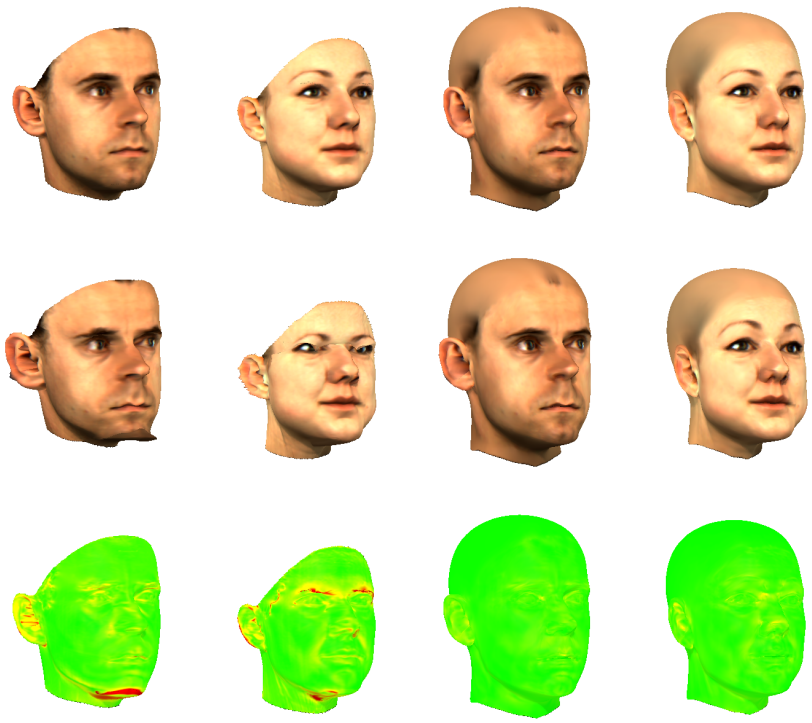


Figure 4.10: Comparison of the displacement field smoothness for two examples. In the top row are the registered results, in the middle row their caricatures to evidence the problems in the correspondence, and in the bottom row a color-coded rendering (red is lower, green higher smoothness) of the per-vertex smoothness values. The results on the left are from [BV99], the ones on the right from our algorithm.

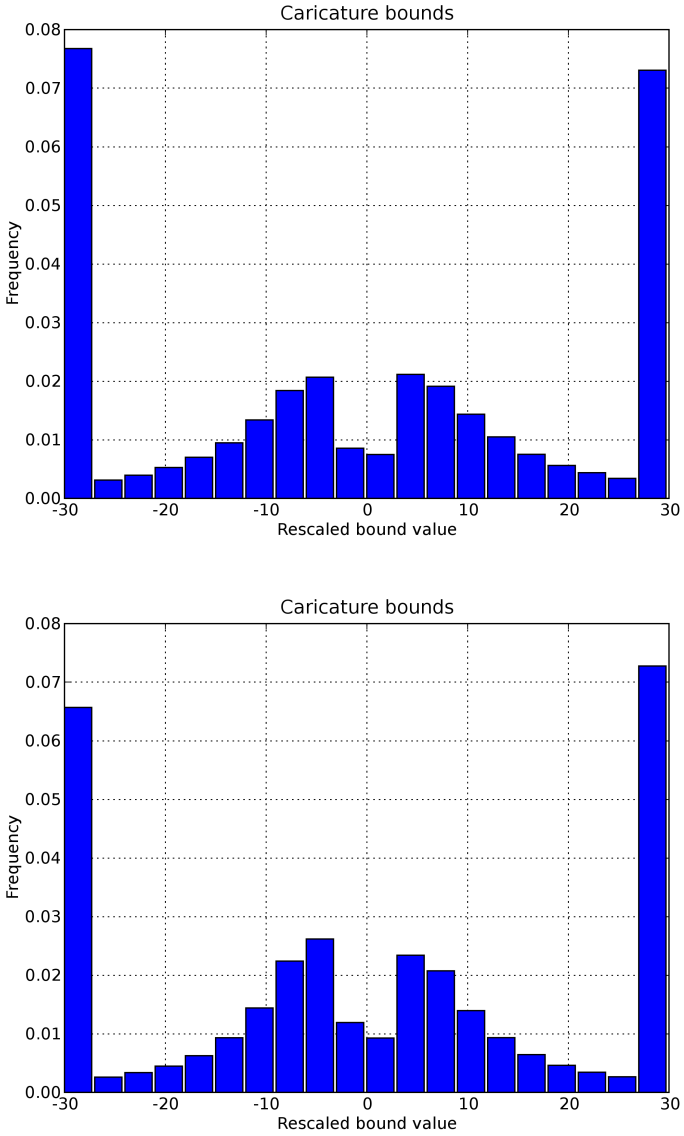


Figure 4.11: Distributions of the caricature bounds for the results of registration on the set D_1 (values clipped to $[-30, 30]$). In the top histogram the results of our algorithm, in the bottom histogram for the algorithm of [BV99]. Both lower and upper bounds are in the same histogram, on the negative and the positive quadrant, respectively. In comparison with the other algorithm, ours shows a distribution of the caricature bounds which is flatter in the middle and more skewed towards the tails, indicating a larger interpolation range.

Figure 4.12: Average distance of the vertices from the original surface. The distance is computed only for the vertices with values for all examples; on the right is the scale. Most of the vertices are close to the surface, with a distance well within the limit of 1.0 mm. The black areas correspond to vertices which are missing in at least one example.

vious method, as shown in figure 4.10, and on average over all the registration results, as shown in figure 4.9.

The smoothness of the displacement field however is not the only measure available for comparison of the registration results. A second measure is what we call *caricature bounds*. When we interpolate between a registered head and the average, either in one direction – away from the average, that is we are extrapolating or making a *caricature* of the registered head – or in the other – towards the average and eventually beyond, to obtain what is called an *anti-face* – a point could be reached where a given triangle will flip, that is the angle between its normals in the interpolation and in the original head will become greater than 90 degrees. This point is a limit to the interpolation: going further will produce artifacts in the mesh. For each results of the registration, we can compute the upper and lower interpolation bounds for each triangle; clearly, a bigger range of interpolation will signal a more regular result. Also in this respect the new algorithm shows an improvement with respect to the one of [BV99], as shown in figure 4.11

Proving that the new results are smoother or more regular does not necessarily prove that they are a better representation of the original novel meshes which were registered. In particular, the smoothing term in the minimized energy implies the risk of smoothing so much the result, that at the end it does not approximate well the input. We can check if this is the case by considering the distances of the vertices of the registered results from the surfaces of the inputs. We do this by projecting the vertices of each result to cylindrical coordinates and then looking at the radial difference with respect to the points which have the same 2D position the inputs. The results, summarized in figure 4.12, shows that the smoothing really affects the distance from the original surface only in

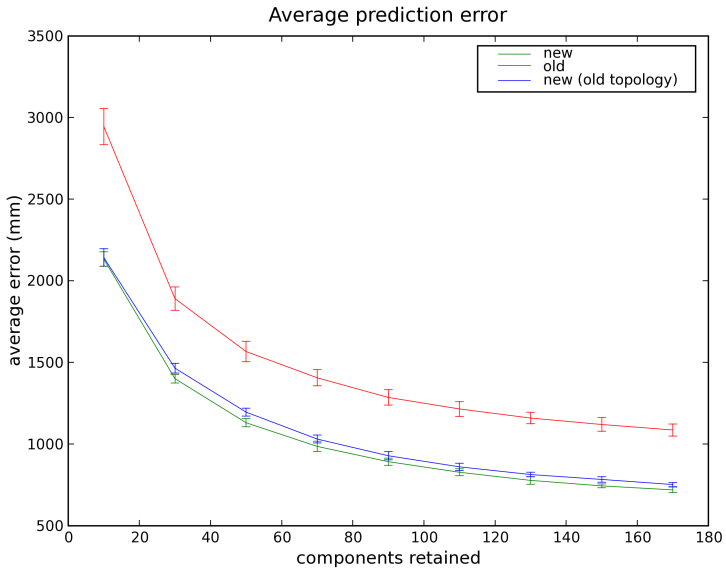


Figure 4.13: Cross-validation results for both models.

small areas, while in the rest of the face the vertices are well within 1.0 mm from the surface. In principle, we could further improve this result including in the energy we minimize a term depending on the distance from the surface, or even a hard constraint forcing the vertices to stay on the surface. However, this would considerably increase the complexity of the optimization.

Evaluation of the Morphable Model.

In the previous section we proved how the new algorithm shows a significant improvement in the quality of the results. This improvement should in turn have a positive effect on the quality of the morphable model built with the registration results. Comparing such two models is a typical problem of statistical learning, known as *model selection* (see [HTF01], chapter 7).

We will consider here the *cross-validation* method, which aims at estimating the *generalization error* of the model. This is the expected error made by the model in reconstructing new data not in the training set. The estimation is obtained by splitting the set of the examples in N subsets (and one then talks about N-fold cross-validation), and then iteratively computing the average reconstruction error over the i-th subset for a model built with the other subsets. In other words, at each iteration the i-th subset is used as test set, and the other subsets as training sets. The mean reconstruction error over all subsets is the estimate of the generalization error. We performed a 10-fold cross-validation on

Gallery\Probe	Frontal	Side	Profile
Frontal	***M1: 99.9*** M2: 99.0	***M1: 98.4*** M2: 96.7	***M1: 75.6*** M2: 72.8
Side	M1: 96.4 ***M2: 96.9***	***M1: 99.3*** M2: 98.9	***M1: 83.7*** M2: 81.4
Profile	M1: 76.3 ***M2: 79.0***	***M1: 86.0*** M2: 84.9	M1: 89.4 ***M2: 91.3***

Table 4.1: Comparison of the image identification results. Gallery and probe images are split into different groups, depending on their pose (see text for a detailed explanation of the experiment). For each pair of gallery and probe groups we list the identification results for the models built with the results of the previous algorithm (***M1***) and of our new algorithm (***M2***). The better results are in bold. ***M1*** is slightly better, with an average performance of ***89.4***%, compared with an average ***89.0***% for ***M2***.

the two models, and the results, summarized in figure 4.13, show that the new model has a better generalization performance than the old one.

Performance of the Morphable Model.

We conclude the results section with an analysis of the performance in a typical application of 3D morphable models. We will consider a face identification experiment run on a database of 3D faces, obtained as reconstructions from a set of images with different poses and lighting conditions; we used the reconstruction method presented in [RV05]. As usual in identification tests, the set of 3D reconstructions is split into a *gallery* and a *probe* set, and the identification task consist in assigning very face in the probe set to a face in the gallery set. A comparison of the results obtained for the two models built with the results from our algorithm and the one of [BV99] (in table 4.1) shows that the performance of the model built with the new results is slightly worse (89.0% vs. 89.4%). The reason for this seems to be however that the topology of the reference mesh we used does not support well the use of edge features by the reconstruction algorithm. In fact, additional experiments run without using the edge informations yielded exactly the same performance for both models, 80.2%.

Chapter 5

Applications

The algorithm described in the previous chapter has been developed in order to be able to register 3D scans of human faces, irrespective of their expression. In the last section of the previous chapter we have already provided an evaluation of the registration results, and in this chapter we are going to show how they can be applied to the face recognition problem. As well as considering this application, we will also discuss an example of application of the combined identity-expression 3DMM to images.

5.1 3D Face Recognition

As already discussed, the input of a 3D recognition problem are two datasets of 3D scans. The gallery set contains one example for each person to be recognized. The probe set contains other examples of the persons in the gallery, acquired in different conditions. In our case, the examples in the gallery have been acquired with neutral expression, while the examples in the probe have been acquired with the six basic emotional expressions. Given the gallery and the probe sets, the recognition task consists of assigning each example of the probe to the correct person in the gallery.

In order to test the performance of our recognition method, we used the database denoted in section 4.7 as D_3. It is a collection of 217 3D scans, taken from 31 different persons. For each person, we acquired seven scans: a neutral expression, plus the six basic expressions (fear, anger, sadness, joy, surprise and disgust). All the scans have been preprocessed and registered as described in the previous chapter, using the combined expression-identity 3DMM built with the datasets D_1 and D_2 of section 4.7. The first is a database of 200 different individuals with neutral expression, while the second a database of 68 expressions from 2 different individuals. We stress again the fact that all the data were processed in the same way, in particular we did not use the registered neutral expressions to register the remaining data of the same person.

The set of registered 3D scans from D_3 was then split into a gallery, made

up of the neutral expressions, and a probe set including all other scans. Note that the gallery could have also been built by choosing randomly an example for each person, but we deem more realistic to assume that the gallery is made up of neutral scans. The recognition could be already performed at this stage, by simply applying a nearest neighbor criterion to the shape vectors (and eventually also the textures) of the registered examples. That is, given a probe shape vector $\boldsymbol{s}$ and the gallery shape vectors $\boldsymbol{s}_i$, we assign $\boldsymbol{s}$ to the gallery example for which the L_2-norm $\|\boldsymbol{s} - \boldsymbol{s}_i\|$ is minimized. With this approach, which is already using the combined expression-identity 3DMM for the registration, we obtain a recognition rate of 85.5%.

The clear problem of such an approach is that the expressions can alter significantly the shape of the face, easily resulting in wrong classifications. Ideally, we should compensate for all the deformations induced to the faces by the expressions. This can be achieved by inferring the optimal identity and expression model coefficients for each of the registered scans, as described in section 2.1.1. In this way we are effectively projecting the shape vectors to two separate spaces, one coding the identity (built with the dataset D_1) and the other the expression information (built with the dataset D_2). Once these two components have been separated, we can perform the recognition using only the identity coefficients.

Recall that with a morphable model combining expression and identity, we assume that the shape vector is generated as in equation (2.6):

$$\boldsymbol{s} = \bar{\boldsymbol{s}} + \begin{pmatrix} \boldsymbol{C}_{id} & \boldsymbol{C}_{xp} \end{pmatrix} \cdot \begin{pmatrix} \boldsymbol{\alpha}_{id} \\ \boldsymbol{\alpha}_{xp} \end{pmatrix} + \boldsymbol{\epsilon},$$

where the identity and expression components are stored into the matrices $\boldsymbol{C}_{id}$ and $\boldsymbol{C}_{xp}$, respectively. As explained in section 2.1.2, given a novel $\boldsymbol{s}$ we can estimate the most likely values for the identity coefficients $\boldsymbol{\alpha}_{id}$ and the expression coefficients $\boldsymbol{\alpha}_{xp}$ from the equation

$$\begin{pmatrix} \boldsymbol{\alpha}_{id} \\ \boldsymbol{\alpha}_{xp} \end{pmatrix} = \boldsymbol{W} \left(\boldsymbol{W}^2 + \sigma^2 \boldsymbol{I}\right)^{-1} \cdot \boldsymbol{U}^T \cdot (\boldsymbol{s} - \bar{\boldsymbol{s}}),$$

where the matrices $\boldsymbol{U}$ and $\boldsymbol{W}$ derive from the SVD of the matrix $\boldsymbol{C} = (\boldsymbol{C}_{id}\ \boldsymbol{C}_{xp})$, and the noise variance σ^2 depends on the number of components retained in the identity model. Using the above equation we assign the probes to the examples in the gallery by comparing their identity coefficients $\boldsymbol{\alpha}_{id}$; discarding the coefficients $\boldsymbol{\alpha}_{xp}$ compensates, at least in part, for the deformations induced by the expressions. Given the coefficients $\boldsymbol{\alpha}_{id}$ of a probe and the coefficients $\boldsymbol{\alpha}_{id}^i$ of the gallery examples, the identification can be done by comparing them with an L_2-norm $\|\boldsymbol{\alpha}_{id} - \boldsymbol{\alpha}_{id}^j\|^2$. However, it has been shown in [BV03] that it is more effective to use the cosine of the angle between the vectors as a distance measure:

$$\cos = \frac{\boldsymbol{\alpha}_{id} \cdot \boldsymbol{\alpha}_{id}^j}{\|\boldsymbol{\alpha}_{id}\| \|\boldsymbol{\alpha}_{id}^j\|}$$

Vectors Used	k_{id}	k_{xp}	σ (mm)	Norm	***Id. Rate***
Shape Vectors	-	-	-	L_2	***85.5%***
ID Coefficients	199	0	0.0	L_2	***90.3%***
				angle	***91.4%***
	175	0	5.24E-2	L_2	***94.1%***
				angle	***94.1%***
ID-XP Coefficients	199	67	0.0	L_2	***97.8%***
				angle	***98.9%***
	150	46	8.87E-2	L_2	***98.9%***
				angle	***99.5%***

Table 5.1: Summary of the identification results. For each result, we specify: the type of feature vectors, the number of identity and expression shape components used by the model and the corresponding noise standard deviation, and finally the norm used to compare the probe vectors with the gallery vectors.

Vectors Used	k_{id}	k_{xp}	σ (mm)	$< L_2 >$	$< angle >$
ID Coefficients	199	0	0.0	0.123	0.899
	175	0	5.24E-2	0.122	0.877
ID-XP Coefficients	199	67	0.0	0.081	0.804
	150	46	8.87E-2	0.087	0.761

Table 5.2: Average distances between the projections of probe and gallery scans on the identity space, on the two rightmost columns. Both distances decreases when using a combined identity-expression model.

In our experiments however we did not observe a significant difference: the identification rate using the L_2-norm is 98.8%, while using the angles is 99.5%.

We already mentioned that the noise variance σ^2 is determined once the number of components retained in the identity model is fixed. Retaining a smaller number of component in the model has the effect of increasing the estimate of the noise variance, and consequently of regularizing the projection. A higher value of the noise variance has the effect of eliminating disturbances in the data which might affect negatively the identification results; on the other hand, the higher its value, the closer the coefficients $\boldsymbol{\alpha}_{id}$ gets to zero, and the more difficult it gets to discriminate correctly between different identities. It is therefore important to choose a number of identity components which trades off between these two effects, but unfortunately there is no absolute criterion we can use. Moreover, the optimal choice will typically depend on the particular problem we want to solve. In order to find the optimal number of identity components to use, we repeated the identification experiments with increasing number of identity components, from 10 to 199 (all the components). As shown in figure 5.1, the best identification rate is obtained with 150 components; however, even a more parsimonious choice of 50 components does not

seem to worsen significantly the identification rate, which only slightly drops to 98.4% (-1.1% compared with the result of using 150 components).

A similar behavior of the identification rate is obtained projecting the shape vectors on the identity space only (figure 5.2). In this case we use only the identity components, that is we are not trying to separate the expression from the identity information. Nevertheless, we obtain a best identification rate of 94.1%, with 175 components. Also in this case, the identification performance obtained with 50 components or more stays in a relatively tight range approximately between 90% and 94%; using the angular distance does not seem to provide a significant advantage. The identification results are summarized in table 5.1. Note that an improvements of the identification rate is coupled with a decrease of the average L_2 and angular distances of a probe vector from its corresponding gallery example (see table 5.2). This is especially evident for the L_2 distance, where the use of the combined model drops the average distance of one third.

Comparison with Bilinear Models As mentioned in the introduction, the combined modeling of identity and expressions could have also been achieved via bilinear models. In theory, a bilinear model has the advantage, compared with the linear model we used, of modeling the dependency of expressions on the identity. It is however not clear if this theoretical advantage can yield an improvement of the identification performance. In order to find this out, we ran additional identification experiments employing a bilinear model for the expressions. That is, we modeled the shape vectors as:

$$\boldsymbol{s} = \bar{\boldsymbol{s}} + \boldsymbol{C}_{id} \cdot \boldsymbol{\alpha}_{id} + \mathcal{C}_{xp} \times_2 \boldsymbol{\alpha}_{id}^T \times_3 \boldsymbol{\alpha}_{xp}^T,$$

where the last term models the expression displacements. In this case the expression model is coded into a 3-mode tensor, $\mathcal{C}_{xp}$, with dimensions $3n \times k_{id} \times k_{xp}$. The second mode of the tensor encodes the dependency of the expression displacements from the identity, and the mode-2 product $\mathcal{C}_{xp} \times_2 \boldsymbol{\alpha}_{id}^T$ results in a matrix with dimensions $3n \times k_{id}$. Conversely, the third mode encodes the dependency of the displacements from the type of expression, and the mode-3 product $\mathcal{C}_{xp} \times_2 \boldsymbol{\alpha}_{xp}^T$ results in a matrix with dimensions $3n \times k_{xp}$. Applying both mode products results in a $3n$-dimensional displacement vector which can be added to the identity $\bar{\boldsymbol{s}} + \boldsymbol{C}_{id} \cdot \boldsymbol{\alpha}_{id}$. For a more extensive treatment of bilinear models, we refer to [TF00, VBPP05].

The use of a bilinear model puts additional constraints on the training set. In order to estimate $\mathcal{C}_{xp}$, we need for each individual in the training set the same expression examples. Although the dataset D_2 does not satisfy this requirement, the dataset D_3 does. Since D_3 has to be used also as probe set, we adopted a leave-one-out approach, where at each iteration the expressions of one individual were removed from D_3 and used as probes, while the expression model was trained with the remaining examples. The identity model, however, has been still built from D_1, and we retained 150 identity components. Performing the identification experiments with this approach both for the bilinear and

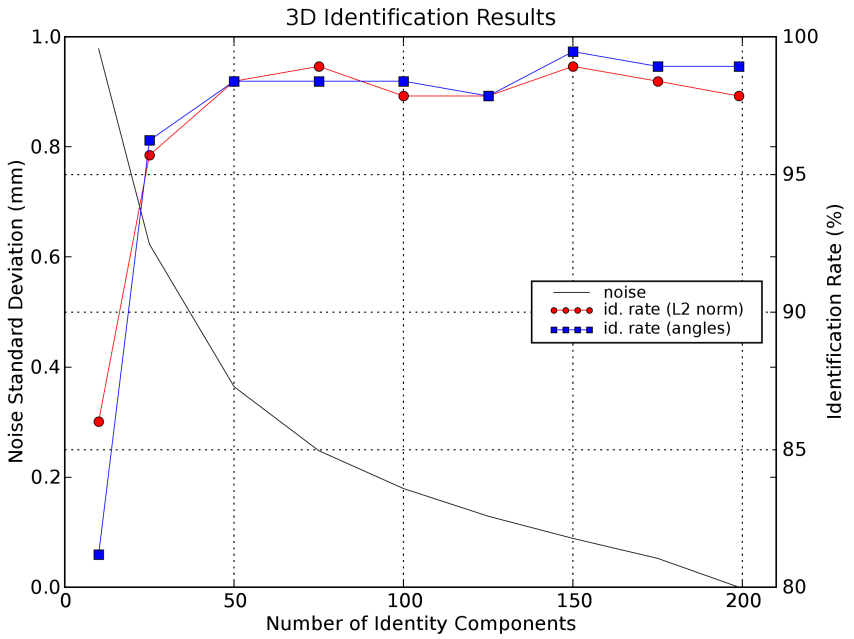


Figure 5.1: Results of the identification experiments, using the combined expression-identity morphable model, with different number of identity components. With increasing number of components, the estimated standard variation of the noise also increases (black solid line, scale on the left). The best identification rate (scale on the right) is obtained using the angular distance between the coefficients vectors, and with 150 identity components. However, when using more than 50 components the identification rate stays essentially constant, varying in a range between 97.8% and 99.5%. Using the L_2-norm rather than the angular distance does not seem to have a significant impact on the results.

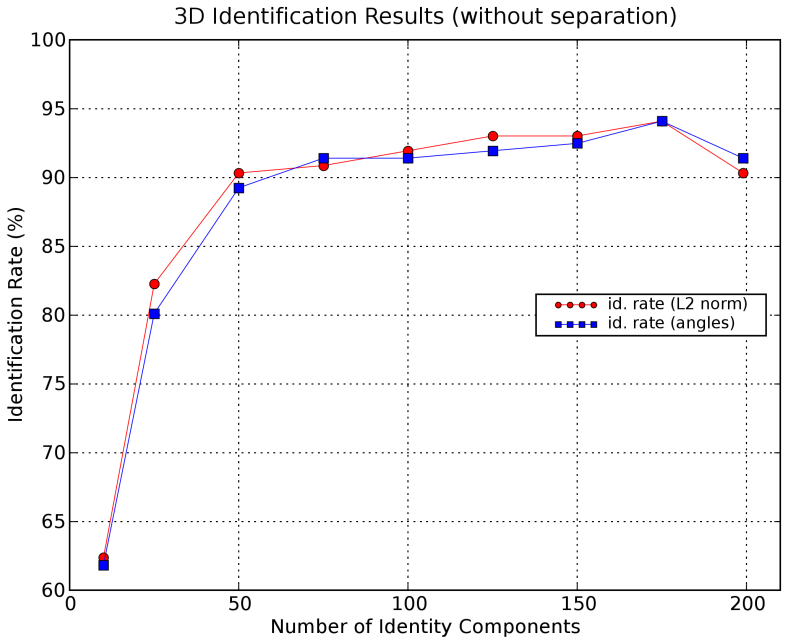


Figure 5.2: Results of the identification experiments, using the identity-only morphable model, with different number of identity components. The best identification rate is obtained in this case with 175 identity components, but as for the results obtained with the combined model, the identification rate stays essentially constant when using more than 50 components.

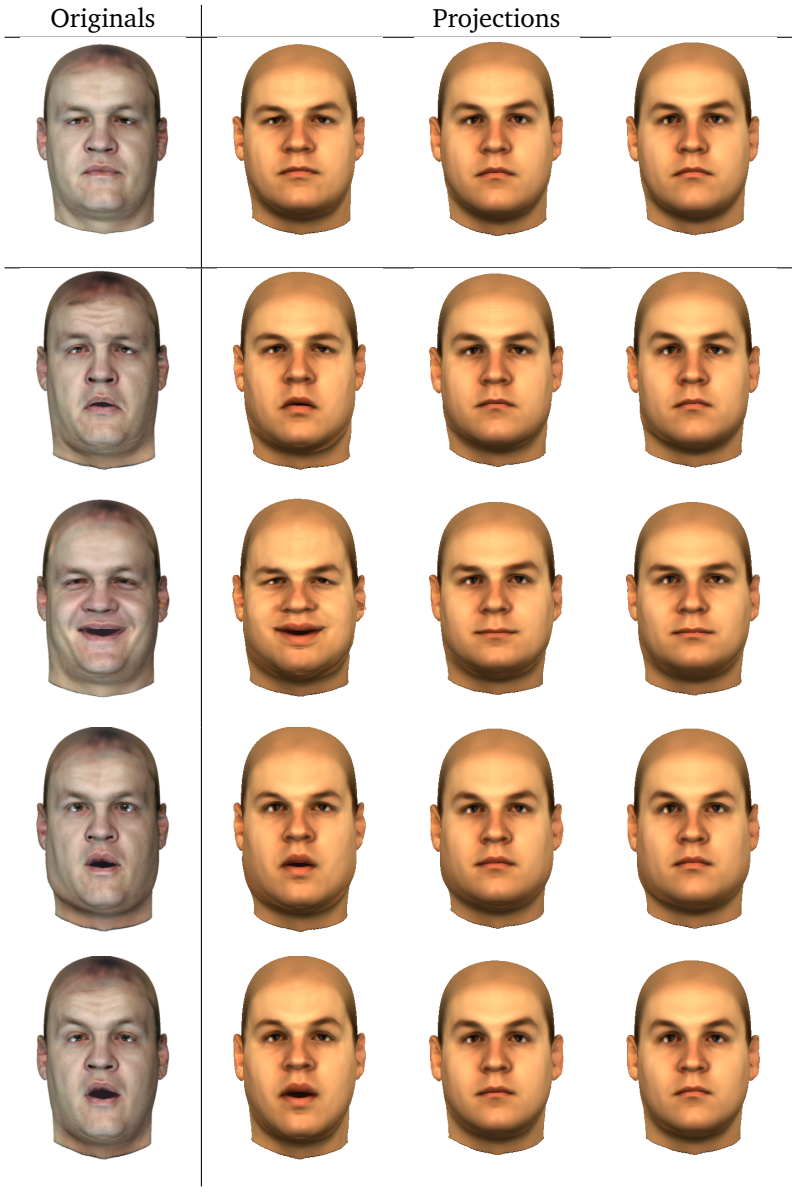


Figure 5.3: Examples of expressions-free reconstructions. On the leftmost column are five registered faces of the same individual. Their shapes are projected to the model space and reconstructed. On the second column are the reconstructions from an identity-only model, while the last two columns are from a combined expression-identity model (they differ in the number of idetity components used, 199 and 100 respectively). Note that we did not apply the original textures to better appreciate the differences in shape.

the linear models, showed that the former is slightly inferior to the latter, with a 97.9% identification rate vs. 99.5% (the same rate obtained using D_2 for the training). The lower identification rate and the additional computation load of the bilinear model, especially for inferring the model coefficients from a shape vector, justify our preference for a linear model.

5.2 Applications to Images

In [BV03] it has been shown how an identity-only 3DMM can be used, with excellent results, for image face recognition; and in principle, the combined expression-identity 3DMM we used in the previous section could also be applied to this problem. However, for image applications an *image fitting* algorithm is required, an algorithm which uses the 3DMM to recover the 3D shape and texture of a face from a single image. Although fitting an expression-only 3DMM to image data has already been tried ([BBPV03]), fitting a combined expression-identity 3DMM has not been tested yet.

Therefore, as a first step towards the application of our model to image face recognition, we experimented with the algorithm of [RV05], fitting images of expressions with our new 3DMM. Although we did not carry on extensive experiments, we were able to test our model on the task of *image normalization*: the process of synthesizing a new image, with a predefined pose and illumination, from the input one. The synthesized image is obtained by fitting a 3DMM to the input image, and then by rendering it back in the same image with the predefined pose and rendering parameters. As shown in [BGPV05], the preliminary normalization of the images can significantly improve the performance of a face recognition algorithm. In figure 5.4 we report some of the results we obtained.

Naturally, these results are no proof that the model can fit any expression, and we do not present them as such. In particular, the method has been tested on a too small number of images in order to assess its reliability. However, we think they might anticipate what can be done in the future. In the conclusions we will return on the issue of image fitting, in particular to try to understand what are the model's limits and how they can be removed.

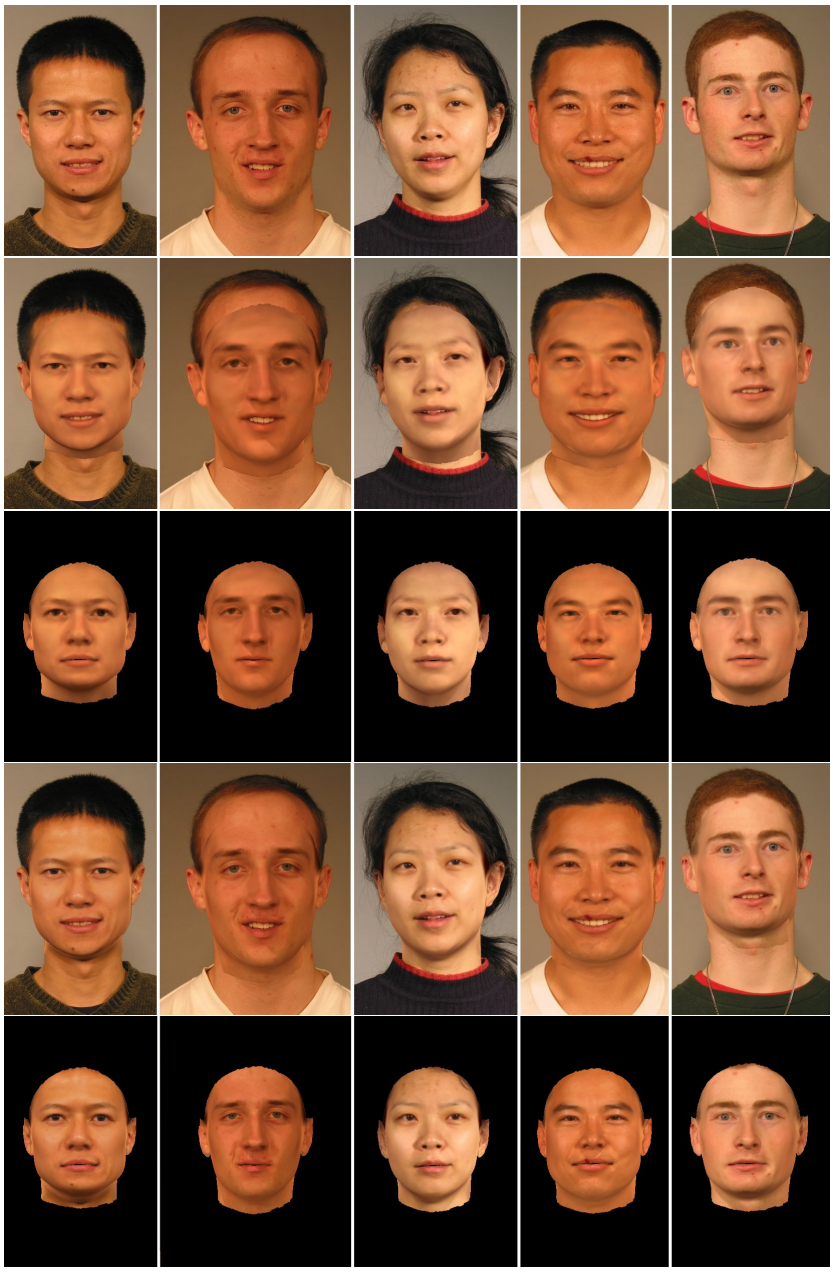


Figure 5.4: Original images on the first row. Fittings on the second row, same fittings normalized on the third. Fittings with extracted texture on the fourth row, normalized on the fifth.

Chapter 6

Conclusion

The construction of a 3DMM for modeling identity and expressions requires a registration algorithm capable to cope with the extreme deformations involved in facial expressions. As already mentioned, the algorithm used in [BV99] yielded excellent results for 3D scans of faces with a neutral expressions, but was not able to register the expressions data. Other algorithms are capable of registering expression data, but they typically work for one individual at a time, since they rely on the knowledge of the individual face with neutral expression. In chapter 4 we described a novel algorithm, aimed at registering expression data with arbitrary identity. The new algorithm uses a slightly modified version of the algorithm from [BV99], based on optical flow, in order to estimate the correspondence between a reference 3D head model and the face examples. The correspondence information allows us to sample the surface of the examples, yielding for each vertex of the model shape and texture values. If the correspondence was perfect and all the model vertices had a correspondence to the examples, we could directly use the sampled values as the ones of the registered results. However, these assumptions are typically not met, and therefore the sampled values are fed to an optimization problem, whose solution is the registered version of the example. Moreover the registration is performed in a bootstrapping scheme, in order to exploit, at each new iteration, the prior knowledge accumulated in the 3DMM built with the examples already registered.

As shown in section 4.7, the novel algorithm allows us to register expressions data and can also cope with missing data in the input. And as well as widening the range of data which can be processed, the new algorithm offers a control on the regularity of the registered results, thanks to the adaptive smoothing performed in the last step of the registration. In fact, the last step is a generalized version of the surface reconstruction method we described in chapter 3. By combining a variational approach with a statistical one, our method ensures continuity at the boundary of the reconstructed areas, while reducing the reconstruction error. From this respect, our registration algorithm differs from others which also allow to process data with missing values, since they typically

employ only a variational criterion for the reconstruction.

The capabilities of the new algorithm offered us the possibility of building a 3DMM combining both identity and expression variations, and in chapter 5 we considered two of its possible applications. The main application we dealt with is 3D face recognition, which we tested on a database of more than 200 3D scans, acquired from 31 individuals. By registering the data, and using a simple nearest-neighbor criterion, we were already able to achieve an identification rate of 85.5%. Projecting the data to the combined identity-expression model, however, we achieved nearly perfect identification results. These identification results might be further improved using the texture information, which we did not use in our experiments. We also showed that the use of a bilinear model for the expression variations does not improve the results; however, this might be due the size of the training set. It is possible that enlarging the training set, especially increasing the number of expressions per individual, would raise the identification performance. Comparisons with the performances of other methods must take into account the differences in the probe and gallery set. In [CBF05], for instance, the authors can claim an identification rate of 91.9%; although their rate is lower than ours, one should consider that it has been obtained on nearly 4000 probes!

In the second application, we showed how the model can be used to the normalize images of faces with expressions, synthesizing new images where the expression had been removed and replaced with a neutral expression. This is an interesting application, since preliminary normalization of the images can greatly improve the performance of face recognition algorithms.

6.1 Outlook

The generalization capabilities of a 3DMM mainly depend on the size of the training set used to build it, and it is therefore desirable to collect and register a large number of examples. As a consequence, it is important that a registration algorithm reduces as much as possible the amount of manual interaction required from the user. As we have seen, the registration algorithm we presented is not completely automatic; the raw 3D scans obtained from an acquisition device require some preprocessing which currently depends on user interaction. In particular, two tasks are left to the user: the selection of few landmarks on the 3D scans, and the selection of the areas not pertaining the model and which should be removed (e.g. hairs, clothes, teeth). An useful improvement of the algorithm would be the automatization of such tasks. Both problems have been already addressed in the literature, in particular the automatic detection of the landmarks. We refer the reader in particular to the method presented in [BBK05], which addresses both problems, and is based on the selection of an area within a certain geodesic distance from the tip of the nose.

Apart from the drawback of the user interaction required in the preprocessing step, the registration algorithm provides very good results, and enabled us to build a 3DMM which performs well in a face recognition task, even if the

probes presents extreme expressions. Although fitting the model to 3D scans usually provides good results without using any landmark, the experience we had fitting the model to images evidenced problems in accurately fitting the lips contour without landmarks. The use of landmarks, however, is not always desirable; in particular, for video tracking it is clearly not possible to manually place landmarks through the whole video. One possible approach might be to place the landmarks in the first frame, and use a dedicated algorithm to track them through the whole video.

We conclude by considering the possible application of the model to the task of *3D face animation*, the automatic generation of dynamic sequences of 3D models simulating human emotions or speech. For such a task additional problems come into play.

- First of all, in order to make the animation method usable, it would be necessary to have a high-level parameterization of the expression space. We mean by this a set of parameters which produce a single and well-identifiable effect on the face, for instance the raising of an eyebrow, or raising of one mouth's corner. This type of parameterization is currently not provided by the model.

- Facial expressions inherently depend on the face identity, and it would increase a model's realism to capture this dependency. Probably, in order to study it, it would be necessary to collect a database of 3D scans acquired from professionals who can activate single facial muscles. Then, the variability of these simple actions across many identities could be modeled (e.g. with bilinear models), and complex expressions might be built in top of them.

- Currently we model expressions only through deformations of the shape. If the shape deformations would perfectly reproduce the real ones, and if our rendering model was perfect, the results would also be perfect. However, since the shape deformations normally approximates the real ones, and since the rendering model is simpler than the real physical phenomenon, some effects are not reproduced. We think in particular to the darkening of areas where the skin folds. This should be modeled in the texture, maybe following the approach of [LSZ01].

Appendix A

Technical Details

A.1 Coarse Alignment

Denoting by $\boldsymbol{L}_S$ and $\boldsymbol{L}_R$ the two $N \times 3$ matrices holding the positions of the N landmarks in the novel and reference meshes, we first compute the mean positions of the two groups of landmarks:

$$\bar{\boldsymbol{l}}_\bullet = \frac{1}{N}\mathbf{1}_N^T \cdot \boldsymbol{L}_\bullet$$

where $\bullet$ is either S or R, and $\mathbf{1}_N$ is an $N \times 1$ vector of ones.

Defining a matrix

$$\boldsymbol{A} = (\boldsymbol{L}_R - \bar{\boldsymbol{l}}_R)^T \cdot (\boldsymbol{L}_S - \bar{\boldsymbol{l}}_S),$$

which can be decomposed by Singular Value Decomposition (SVD) as $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{W}\boldsymbol{V}^T$, the rotation is estimated as

$$\boldsymbol{R} = \boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^T$$

where $\boldsymbol{S}$ is a diagonal 3×3 matrix defined as

$$\boldsymbol{S} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & sign|\boldsymbol{A}| \end{pmatrix}$$

The translation is defined as

$$\boldsymbol{t} = \bar{\boldsymbol{l}}_R - \boldsymbol{R} \cdot \bar{\boldsymbol{l}}_S$$

A.2 Newton Descent Algorithm

In this section we describe the Newton descent algorithm for a cost function of the form

$$E(\boldsymbol{c}) = E_d(\boldsymbol{c}) + \eta_p \sum_i \frac{(c_i - \hat{c}_i)^2}{\sigma_i^2},$$

with $\boldsymbol{c} = (c_1, \ldots, c_n)$. The application of the algorithm to the full form of the cost function used in section 4.3 is straightforward.

In order to find the optimal values h_i for the parameters c_i, one can set to zero the derivatives of the cost function with respect to them:

$$\left.\frac{\partial E}{\partial c_i}\right|_{h_i} = \left.\frac{\partial E_d}{\partial c_i}\right|_{h_i} + \frac{2\eta_p}{\sigma_i^2}(h_i - \hat{c}_i) = 0. \tag{A.1}$$

Of course, since there is no easy analytical representation of E_d, the equation cannot be solved in closed form and the optimal solution is found by iterative update of the coefficients c_i.

In our specific case, the update rule is found by approximating E_d with the quadratic form

$$E_d = \sum a_i (c_i - g_i)^2,$$

with parameters a_i and g_i unknown. From this approximation follows that the first derivative is

$$\frac{\partial E_d}{\partial c_i} = 2a_i(c_i - g_i),$$

and its value at h_i can be expressed in terms of its value for the current estimate of the coefficient as

$$\left.\frac{\partial E_d}{\partial c_i}\right|_{h_i} = \left.\frac{\partial E_d}{\partial c_i}\right|_{c_i} + 2a_i(h_i - c_i).$$

In this latter equation, both the derivative at the point c_i and the parameters a_i (which depends on the diagonal elements of the Hessian matrix of E_d) can be numerically estimated by finite differences.

Given the above approximation, we can rewrite equation A.1 as

$$\left.\frac{\partial E_d}{\partial c_i}\right|_{c_i} + 2a_i(h_i - c_i) + \frac{2\eta_p}{\sigma_i^2}(h_i - \hat{c}_i) = 0$$

so that the optimum h_i is defined by

$$h_i = \left(-\frac{1}{2} \left.\frac{\partial E_d}{\partial c_i}\right|_{c_i} + a_i c_i + \frac{\eta_p}{\sigma_i^2}\hat{c}_i \right) \cdot \left(a_i + \frac{\eta_p}{\sigma_i^2} \right)^{-1}$$

Naturally, since this is only an estimate of the optimum, we only update the coefficient in its direction, that is:

$$c_i \rightarrow c_i + \lambda(h_i - c_i),$$

with the factor $0 < \lambda \leq 1$ equal for all coefficients.

A.3 Mechanical Smoothing of the Flow Field

As detailed in section 4.4, at each iteration of the optical flow algorithm for 3D data a smoothing of the flow field is needed.

The smoothing is performed as in [Bla00] by modeling a mechanical mass-spring system, where each mass corresponds to a pixel of the input data and the springs connect the masses corresponding to neighboring pixel. Denoting by $\boldsymbol{v}_0(u,v)$ the initial flow field and by $\boldsymbol{v}(u,v)$ the result of the smoothing, the mass corresponding to a pixel (u,v) has a position $(u,v)+\boldsymbol{v}(u,v)$ and a certain potential depending on its distance from the initial position $(u,v)+\boldsymbol{v}_0(u,v)$. The total energy of the system is then given by the sum of the mass potential and the spring potential:

$$E = E_s + E_m,$$

and the result $\boldsymbol{v}$ is obtained finding the system configuration at the equilibrium, via a conjugate gradient descent algorithm.

The energy of the springs is easily defined as

$$E_s = \frac{1}{2}\sum \|\boldsymbol{v}(u+1,v)-\boldsymbol{v}(u,v)\|^2 + \frac{1}{2}\sum \|\boldsymbol{v}(u,v+1)-\boldsymbol{v}(u,v)\|^2.$$

The mass potential, however, should be defined such as it strongly penalizes differences between $\boldsymbol{v}$ and $\boldsymbol{v}_0$ only where the contrast in the data is high, while being null in areas of low contrast. More specifically, given the energy minimized to obtain the initial flow field $\boldsymbol{v}_0$ at a given point:

$$E = \sum_{u,v\in R} \left\| \boldsymbol{v}(u,v)\cdot\nabla J|_{(u,v)} + \left.\frac{\partial J}{\partial t}\right|_{(u,v)} \right\|^2,$$

we can compute how much it would increase by modifying the flow field at that point with an additive term (du,dv). One can show that such a change would produce an increase equal to

$$dE = \lambda_1(\boldsymbol{a}_1\cdot(du,dv))^2 + \lambda_2(\boldsymbol{a}_2\cdot(du,dv))^2,$$

where we denote by λ_i and $\boldsymbol{a}_i$ (with $\lambda_1>\lambda_2$) the eigenvalues and eigenvectors of the matrix $\boldsymbol{W}$

$$\boldsymbol{W} = \begin{pmatrix} \sum_R \|\partial_u \boldsymbol{J}\|^2 & \sum_R \langle \partial_u \boldsymbol{J}, \partial_v \boldsymbol{J}\rangle \\ \sum_R \langle \partial_u \boldsymbol{J}, \partial_v \boldsymbol{J}\rangle & \sum_R \|\partial_v \boldsymbol{J}\|^2 \end{pmatrix}.$$

The idea is now to set the potential to zero at the points where dE is low, since these are the low-contrast points where the flow field can be freely smoothed, and to set it to a high value were dE is large, since these are the high contrast points where the flow field should be kept relatively fixed. In practice, given a threshold value s, we identify three cases: (1) if $\lambda_1 \le s$, the effect of an error on the flow would be negligible (low contrast areas); (2) where $\lambda_1 > s > \lambda_2$

only errors along the direction of $\boldsymbol{a}_1$ are significant; (3) if $\lambda_2 \geq s$ all errors are significant (high contrast areas). Accordingly, we define the potential as

$$E_a(u,v) = \begin{cases} 0 & if \quad \lambda_1 \leq s \\ t \cdot (\boldsymbol{a}_1 \cdot (\boldsymbol{v} - \boldsymbol{v}_0))^2 & if \quad \lambda_1 > s > \lambda_2 \\ t \cdot \|\boldsymbol{v} - \boldsymbol{v}_0\|^2 & if \quad \lambda_2 \geq s \end{cases},$$

where t is a constant chosen empirically.

Appendix B

Software

The algorithms described in this thesis have been implemented into a Python (`http://www.python.org`) package, for the Linux operating system. The package includes a set of library modules offering data structures and basic functions for the developer, and a set of tools and applications for the user.

B.1 Installation

The package is stored into the software repository of the Basel University's Computer Science Department, and is available only internally. The following Subversion (`http://subversion.tigris.org`) command extracts – *checks out* in Subversion's terminology – the package to the directory `$PKGROOT`:

```
> svn co https://svn.cs.unibas.ch/repos/gravis/people/basso/\
  python/trunk $PKGROOT
```

After extraction, the directory `$PKGROOT` will contain the library modules, plus the following subdirectories:

- `apps`: tools and applications.
- `doc`: package documentation.
- `src`: C/C++ sources of the extension modules (more on this subject later).
- `test`: test modules.

B.1.1 Dependencies

As well as the Python programming language and Subversion, standard components of an up-to-date Linux distribution, our software requires the following libraries:

- ***numarray***: a Python package used for numerical computations, e.g. linear algebra. If you do not install it manually, you should verify that the automatic installation checks for a BLAS/LAPACK implementation (see next item) and links against it if present. Downloadable from `http://www.stsci.edu/resources/software_hardware/numarray`.
- ***BLAS/LAPACK***: a linear algebra C library available in different implementations. It is not strictly required, but if present numarray can be linked against it resulting in an improved speed for linear algebra operations. I suggest using the ATLAS implementation, downloadable from `http://math-atlas.sourceforge.net/`.
- ***UMFPACK***: a C library for solving unsymmetric sparse linear systems. This is required for surface reconstruction, and therefore for 3D registration. At `http://www.cise.ufl.edu/research/sparse/umfpack/`.
- ***PIL*** (Python Imaging Library): a package used for imaging, but my software uses it only for handling I/O to different file formats. Downloadable from `http://www.pythonware.com/products/pil/`.
- ***wxPython***: bindings to the wxWidgets C++ library, a cross-platform UI framework. It is not strictly necessary, but some useful tools are based on it. A manual installation can be very time-consuming: it is preferable to install a binary package if available. Downloadable from `http://www.wxpython.org/`.
- ***libVB, libNR***: the old C/C++ libraries used by our group, and still required for rendering 3D meshes. They can be extracted from the group's software repository.

B.1.2 Setup

After having installed the required libraries, we can set up our Python package. It is not a *pure* package, with all the modules written in Python: for speed reasons, some of them are so-called *extensions* modules, written in C/C++ and needing compilation. The compilation is performed with the following command (assuming you are in `$PKGROOT`):

```
> python setup.py build_ext --inplace
```

The command first checks if some libraries are present, and depending on this selects the extensions which should be compiled (the `umfpack` module, for instance, is not compiled if the `UMFPACK` library is not found). There are some environment variables which influences the compilation process:

- `CXX,CC`: defines the compiler to be used (by default is `gcc`).
- `CPATH`: specifies a list of directories to be searched for include files by the compiler.

- LIBRARY_PATH: specifies a list of directories to be searched for libraries by the compiler.

The last two variables are used when some of the required libraries has been installed in non-standard locations. After compilation, the setup is completed by adding to the PYTHONPATH environment variable the absolute path of the $PKGROOT directory.

To test that everything is working correctly, you can run the tests present in the test subdirectory:

```
> python $PKGROOT/test/test_all.py
```

preferably from an other directory than $PKGROOT, to be sure that $PYTHONPATH has been correctly set. If the library has been correctly set up, you should obtain an output similar to

```
> python $PKGROOT/test/test_all.py
...
----------------------------------------------------------------
Ran 3 tests in 2.828s

OK
```

Otherwise, you will be reported on which test failed.

B.2 Library Overview

The following is a list of the most important modules in the package, along with a brief description of their main functionalities. We split them in three groups: the first two includes modules related to, respectively, 3D data and 3D morphable models. The last group includes modules with a more general application scope.

Handling and processing of 3D data.

- Obj: parsing of Alias/Wavefront OBJ/MTL files. The main functions are read_mesh(filename) and write_mesh(filename, mesh), which do exactly what their names suggest.

- Mesh: implements a 3D mesh structure with the Mesh class. This relies on the class Topology, storing the list of faces and texture coordinates, and the class BoundingBox.

- cyber: handling/creation of Cyberware data, through the class Cyber. The function frommesh(mesh) converts a Mesh to a Cyber via cylindrical projection.

- `render`: interface to software rendering implemented in `libVB`, via the function `render(scene, params)`. The function's second argument is an object of class `Params`, storing all the necessary rendering parameters. The module also provides a parser for `.rdb` files, in order to load the rendering parameters from the result of an image fitting (obtained via `mouflon`).
- `poisson`: implementation of Poisson surface reconstruction (see 3.4).

Building and usage of 3D Morphable Models.

- `registration`: subroutines for the 3D registration algorithm. This module provides the low-level functions used by the front-end application `apps/linear.py`.
- `Flow`: computation of optical flow between `Cyber` objects, via the function `fromcybers()`. The flow data are read/written with the functions `readFFlow` and `writeFFlow`.
- `sgdfit`: implementation of the 3D fitting via Stochastic Gradient Descent, used by the `registration` module, and by the `apps/fit3d.py` application.
- `Heads`: handling of 3D data in HEAD format, via the `Head` and the `Mask` objects.

Miscellaneous.

- `la`: linear algebra functions not provided by the numarray library. Particularly important are the `Sparse` class, which implements a sparse matrix used for surface reconstruction, and the `load` and `save` functions.
- `trans3d`: estimation of optimal 3D transformations (rigid, affine, similarity) via the `get_trans` function.
- `image`: image processing functions not provided by the numarray or the PIL library. In particular, the functions `laplacian` creates the Laplacian pyramid of a (multi-channel) image. Other functionalities are: conversion between numarray and PIL format, I/O to `Fimage` format, color correction (via gain, offset and contrast parameters).
- `Statistic`: implementation of Principal Component Analysis (PCA), EM-PCA and PPCA, as well as I/O routines.

B.3 Documentation

The code of the library modules is documented. This documentation might be used in two ways.

First, it can be extracted using Doxygen (`http://www.doxygen.org`). A configuration file, called `Doxyfile`, is already available in the software root directory, and issuing (from the `$PKGROOT` directory) the following command

```
> doxygen
```

will create in the `doc/doxy` subdirectory an extensive HTML documentation of the code.

A second way is inherent to Python. In Python every function and class can be documented using so-called *docstrings*, documentation text placed immediately after the object declaration (in Python everything is an object). The docstrings are a member of the object: they can be accessed from the code and, more importantly, from an interactive Python shell. Since the use of an interactive shell is common for Python development, this results in an handy way of accessing documentation.

B.4 Applications

As well as the library modules, our Python package includes a set of applications and small tools, located in the `apps` subdir.

B.4.1 Preprocessing

In order to ease the preprocessing of the input data (described in section 4.2.1), we developed the program `apps/preprocess.py`. With it, the user can preprocess a set of input 3D scans in three steps:

1. ***Landmarks placement.*** After being called, the program displays the textures of the scans, one after the other. For each texture, the user has to place 5 landmarks by left-clicking with the mouse: exterior corners of left and right eyes, tip of the nose, left and right corners of the mouth.
   - if you placed some of the landmarks in the wrong place, press 'u' to remove them and begin again.
   - after having placed the landmarks, press 'ENTER' to store the positions and step to the next texture.
   - press 'l' to load the landmarks from a previously saved file and skip to the next scan.
2. ***Alignment to the reference.*** Once the landmarks have been placed for all textures, the program will proceed to compute for each scan the optimal alignment to the reference. No user interaction is required at this stage.
3. ***Data cleaning.*** Upon completion of the alignment step, the program switches to cleaning mode. The user is presented with the textures of

the scan's cylindrical projections, with the void areas marked in red. By moving the mouse cursor with the left button pressed, the user can remove parts of the data (which will then be shown in red). By moving the mouse cursor with the right button pressed, the user can mark void parts where the texture coordinates should be recovered during registration (shown in green). It is advisable to mark in green only small areas, like holes at the eyes.

- pressing 'a'/'s' increases/decreases the radius of the circle.
- as in step (1), you can press 'ENTER' to accept the modifications and 'u' to undo them.
- pressing 'SPACE' any modification will be discarded.
- pressing 'f' chooses floodfill mode which is useful for marking inpaint regions, the cursor will then change to a cross.
- pressing 'p' chooses pencil mode, the cursor will change back to a circle.
- pressing 'g' sets each red inner pixel to green

B.4.2 Registration

The full registration process of chapter 4 is carried one by a single program, `apps/linear.py`. Its basic usage is simple:

```
> python linear.py [options] <list of input files>
```

where the input files must be in Cyberware format.

There are three groups of options: the initialization, the processing and the output options.

Initialization options. The options in this group are needed to specify some files needed by the registration process.

- `--sgdplan` specifies the plan file for the 3D SGD fitting. No default values is set for this file, and if none is specified, than the fitting is not performed.
- `--mean` specifies the OBJ file where the mean is stored. If no value is given, than the program looks for the a meanxp.obj file in the current directory.
- `--mouth` specifies a text file with the list of vertices belonging to the mouth. The default is set to ./mouth.fp.
- `--laplacian` specifies a binary file where the Laplacian matrix (used in the surface reconstruction) has been stored (with misc.dump). The default is set to ./K.dmp.

Processing options. The options in this group influences how the registration is done. The most important is `--double`, which enables the processing of the two textures images as obtained from the ABW scanner.

Other options are for fine-tuning:

- `--fsmlow` sets the lower threshold for the displacement smoothness, denoted as s in section 4.5.2. Decreasing its value will increase the size of the areas affected by the smoothing. Increasing it will achieve the contrary.
- `--wsmooth` sets the intermediate value of λ, see again section 4.5.2. Decreasing it will increase smoothness of the areas affected.
- `--norm_cos` is used to exclude from the registration the areas of the input cyber where the projection direction is nearly parallel to the surface. Its default value is zero, meaning this functionality is not used. When different from zero, the program exclude the vertices for which the cosine of the angle between their normal and the projection direction is smaller.

The two remaining options are sometimes useful for testing:

- `--no-textr` disables the texture registration.
- `--no-rec` disables the surface reconstruction.

Output options. The options in this group determine what is stored and how.

- `-o` sets the output directory, by default the current one.
- `--no-head` disables saving the output also in HEAD format (only OBJ is used).
- `--test` enables the rendering of caricatures and antifaces at the end of each registration. The reference OBJ must be provided as argument of the option.
- `--tsize` sets the size of the registered texture. By default it is 512, and this is usually enough. It could however occur that one needs a higher resolution for a more realistic result.
- `--debug` enables the storing of some intermediate result, in the `./debug` subdirectory.

B.4.3 3D Fitting

During the registration, the input 3D scan is approximated with a 3DMM. This process of 3D fitting can be also performed independently from the registration, using the `apps/fit3d.py` program:

```
> python fit3d.py -p <planfile> [other options] <input cybers>
```

The `planfile` is a configuration file, used also by the `linear.py` program (where is specified by the `--sgdplan` option). It is divided in three sections: Global, Fullface and Segment.

```
[Global]
# section with global options
# ...

[Fullface]
# section with options for fitting the whole face
# ...

[Segment]
# section with options for fitting subsets of the face
# ...
```

In the global section the options are specified with a

```
keyword = value
```

format, where `value` can also include an environment variable. If `value` specifies a relative path, it is assumed to be relative to the position of the plan file. The global section can be as simple as

```
[Global]
mesh = mean.obj # OBJ file with mean head
model_id = ID   # prefix of the identity PCA model
...
```

In the other two sections, only one keyword is used, `step`, and its value is made up of different lines, each line specifying an optimization step. The line has the format

```
iter sample shape textr sigma shape_l textr_l rigid_l [color_l]
```

- the `iter` parameter specifies how many iterations should be done at that step, while the `sample` parameter specifies how many vertices should be sampled at each iteration; typical values are 6000 and 40, respectively.
- the `shape` and `textr` parameters specify the number of shape and texture components should be used at that step.
- the `sigma` parameter can be used to set the relative weight of the prior probability in the cost function; however, if set to 0, the weight is defined based on the number of shape and texture component used at that step.
- the last four parameters, `shape_l` `textr_l` `rigid_l` and `color_l`, specifies the update length; if set to -1, their value is defined by `shape_l`.

As an example, the following block defines two steps, in which the rigid coefficients are not fit anymore.

```
steps = 6000 40  99 99  0.0  0.001   -1  0.
        6000 40  99 99  0.0  0.0005  -1  0.
```

B.4.4 Model Building

After registration, the construction of a 3DMM requires to perform a PCA over the registered examples. To this aim, two programs are used, `pca-heads.py` and `pca-xp.py`, for identity and expression data respectively.

The PCA over the identity data is performed issuing the command

```
> python pca-head.py -o <PCA file prefix> -m <output mean> \
  --topo <reference OBJ> <HEAD files>
```

The program takes as input a set of registered examples in HEAD format, and the reference OBJ (`--topo` option). After storing the data matrix in `<PCA file prefix>.data`, the PCA is computed, and the result is stored in the file `<PCA file prefix>.pca`. The mean is stored in an OBJ file (option `-m`).

The PCA over the expression is performed in a similar way, with the command

```
> python pca-xp.py -o <PCA file prefix> -m <output mean> \
  --xp <config file>
```

In this case the input data are specified in a configuration file, passed with the option `--xp`. The configuration file is needed because the expressions data are grouped according to the identity of the subject, and a different neutral expression has to be subtracted from them. For example, the configuration

```
[A]
neutral = output_id/A_neutral
xps = output/A

[B]
neutral = output_id/B_neutral
xps = output/B
```

specifies two expression data sets, for persons A and B. The neutral expressions are loaded from the directory `output_id`; all the HEAD files in the directories `output/A` and `output/B` are assumed to be expression examples of person A and B, respectively.

List of Figures

List of Tables

Curriculum Vitae

Address	Reg. Oliveto 4 17031 Albenga (SV) Italy
Tel.	+39-0182-20848
Email	curzio.basso@gmail.com
Date of Birth	10th of June 1975
Citizenship	Italian

Education and Employment

2003-2005	University of Basel, Switzerland – Philosophical Doctorate in Computer Science, with the thesis *Separation of Identity and Expression Information in 3D Scans of Human Faces*.
2000-2003	Researcher at the University of Freiburg, Germany.
1994-1999	University of Genova, Italy – Master degree in Physics, with the thesis *Support Vector Machines applied to Regression Problems*.

Publications

C. Basso, P. Paysan, and T. Vetter. Registration of expressions data using a 3d morphable model. To appear in *Proceedings of Automatic Face and Gesture Recognition*, Southampton, UK, 10–12 April 2006.

C. Basso and T. Vetter. Surface reconstruction via statistical models. In *Proceedings of 2nd Inter national Conference on Reconstruction of Soft Facial Parts* (RSFP 2005), Remagen, Germany, 17–18 March 2005.

S. Romdhani, V. Blanz, C. Basso, and T. Vetter. Morphable Models of Faces. Chapter 10 of *Handbook of Face Recognition*. Springer-Verlag, 2004.

C. Basso, T. Vetter, and V. Blanz. Regularized 3d morphable models. In *Proceedings of the 1st IEEE International Workshop on Higher-Level Knowledge in 3D Modeling and Motion Analysis* (HLK 2003), pages 3–11, Nice, France, 17 October 2003. IEEE Computer Society Press.

V. Blanz, C. Basso, T. Poggio, and T. Vetter. Reanimating faces in images and video. *Computer Graphics Forum*, 22(3):641–641, 2003. Best Paper Award.

M. Pittore, C. Basso, and A. Verri. Representing and recognizing visual dynamic events with suppor t vector machines. In *Proceedings of the 10th International Conference on Image Analysis and Processing* (ICIAP 1999), pages 18–25, Venice, Italy, 27–29 September 1999. IEEE Computer Society Press.

Bibliography

[ACP03] B. Allen, B. Curless, and Z. Popović. The space of human body shapes: reconstruction and parameterization from range scans. In Rockwood [Roc03].

[ASP+05] D. Anguelov, P. Srinivasan, H.-C. Pang, D. Koller, S. Thrun, and J. Davis. The correlated correspondence algorithm for unsupervised registration of nonrigid surfaces. In L.K. Saul, Y. Weiss, and L. Bottou, editors, *Advances in Neural Information Processing Systems*, volume 17, pages 33–40. MIT Press, 2005.

[BA83] P. Burt and E. Adelson. The laplacian pyramid as a compact image code. *IEEE Transactions on Communications*, pages 532–540, 1983.

[BAHH92] J. Bergen, P. Anandan, K. Hanna, and R. Hingorani. Hierarchical model-based motion estimation. In *Proceedings of the European Conference on Computer Vision*, pages 237–252, 1992.

[BBK05] A.M. Bronstein, M.M. Bronstein, and R. Kimmel. Three-dimensional face recognition. *International Journal of Computer Vision*, 64(1):5–30, 2005.

[BBPV03] V. Blanz, C. Basso, T. Poggio, and T. Vetter. Reanimating faces in images and video. *Computer Graphics Forum*, 22(3):641–641, 2003. Best Paper Award.

[BGPV05] V. Blanz, P. Grother, J. Phillips, and T. Vetter. Face recognition based on frontal views generated from non-frontal images. In *Proceedings IEEE Conference on Computer Vision and Pattern Recognition* [cvp05].

[Bla00] Volker Blanz. *Automatische Rekonstruktion der dreidimensionale Form von Gesichtern aus einem Einzelbild*. PhD thesis, Fakultät für Physik, Eberhard-Karls-Universität, Tübingen, 2000.

[BMVS04] V. Blanz, A. Mehl, T. Vetter, and H. P. Seidel. A statistical method for robust 3d surface reconstruction from sparse data. In *Int. Symp. on 3D Data Processing, Visualization and Transmission, Thessaloniki, Greece*, 2004.

[BN98] H. Burkhardt and B. Neumann, editors. *Proceedings of the 5th European Conference on Computer Vision (ECCV 98), Volume II*, volume 1407 of *Lecture Notes in Computer Science*. Springer, June 2–6 1998.

[BOP98] S. Basu, N. Oliver, and A. Pentland. 3d lip shapes from video: A combined physical-statistical model. *Speech Communication*, 26(1):131–148, 1998.

[BSCB00] M. Bertalmio, G. Sapiro, V. Caselles, and C. Ballester. Image inpainting. In *Proc. of SIGGRAPH'00*, 2000.

[BV99] Volker Blanz and Thomas Vetter. A morphable model for the synthesis of 3d faces. In *Proceedings of the 26st International Conference on Computer Graphics and Interactive Techniques (SIGGRAPH'99)*, pages 187–194. ACM Press, 1999.

[BV03] V. Blanz and T. Vetter. Face recognition based on fitting a 3d morphable model. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(9), 2003.

[BVB03] C. Basso, T. Vetter, and V. Blanz. Regularized 3d morphable models. In *Proceedings of the 1st IEEE International Workshop on Higher-Level Knowledge in 3D Modeling and Motion Analysis (HLK 2003)*, pages 3–11, Nice, France, 17 October 2003. IEEE Computer Society Press.

[CBF05] K.I. Chang, K.W. Bowyer, and P.J. Flynn. Adaptive rigid multi-region selection for handling expression variation in 3d face recognition. In *Proceedings of the IEEE Workshop on Face Recognition Grand Challenge Experiments*, San Diego, CA, 21 June 2005. IEEE Computer Society Press.

[CCET99] N. Costen, T. Cootes, G. Edwards, and C. Taylor. Automatic extraction of the face identity-subspace. In *Proceedings of the British Machine Vision Conference (BMVC 99)*, pages 513–522, 1999.

[CDB02] E.S. Chuang, H. Deshpande, and C. Bregler. Facial expression space learning. In *Proceedings of Pacific Graphics*, 2002.

[CDD+04] U. Clarenz, U. Diewald, G. Dziuk, M. Rumpf, and R. Rusu. A finite element method for surface restoration with smooth boundary conditions. *Computer Aided Geometric Design*, 21(5):427–445, 2004.

[CET98] T.F. Cootes, G.J. Edwards, and C.J. Taylor. Active appearance models. In Burkhardt and Neumann [BN98], pages 484–498.

[cvp05] *Proceedings IEEE Conference on Computer Vision and Pattern Recognition*. IEEE Computer Society Press, 2005.

[Dav04] Timothy A. Davis. Umfpack version 4.3, 2004. http://www.cise.ufl.edu/research/sparse/umfpack/.

[DCOY03] I. Drori, D. Cohen-Or, and H. Yeshurum. Fragment-based image completion. In Rockwood [Roc03], pages 303–312.

[DLR77] A.P. Dempster, N.M Laird, and D.B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society series B*, 39:1–38, 1977.

[DMGL02] J. Davis, S.R. Marschner, M. Garr, and M. Levoy. Filling holes in complex surfaces using volumetric diffusion. In G.M. Cortelazzo and C. Guerra, editors, *Proc. 1st International Symposium on 3D Data Processing Visualization and Transmission (3DPVT-02)*, pages 428–438. IEEE Computer Society, 2002.

[ECT98] G.J. Edwards, T.F. Cootes, and C.J Taylor. Face recognition using active appearance models. In Burkhardt and Neumann [BN98], pages 581–595.

[GGSC96] S.J. Gortler, R. Grzeszczuk, R. Szeliski, and M.F. Cohen. The lumigraph. In *Proceedings of the 23rd International Conference on Computer Graphics and Interactive Techniques (SIGGRAPH'96)*, pages 43–54. ACM Press, 1996.

[HS81] B.K.P. Horn and B.G. Schunck. Determining optical flow. *Artificial Intelligence*, 17:185–203, 1981.

[HTF01] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer-Verlag, 2001.

[KCVS98] L. Kobbelt, S. Campagna, J. Vorsatz, and H.-P. Seidel. Interactive multi-resolution modeling on arbitrary meshes. In *Proc. of SIGGRAPH '98*, 1998.

[KHYS02] K. Kähler, J. Haber, H. Yamauchi, and H.-P. Seidel. Head shop: Generating animated head models with anatomical structure. In S.N. Spencer, editor, *Proceedings of the 2002 ACM SIGGRAPH Symposium on Computer Animation*, pages 55–64, 2002.

[Lie03] P. Liepa. Filling holes in meshes. In *Symposium on Geometry Processing*, pages 200–205, 2003.

[LK81] B.D. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In *Proc. IJCAI*, pages 674–679, 1981.

[LSZ01] Z. Liu, Y. Shan, and Z. Zhang. Expressive expression mapping with ratio images. In Pocock [Poc01], pages 271–276.

[MGR00] S.R. Marschner, B. Guenter, and S. Raghupathy. Modeling and rendering for realistic facial animation. In *Proceedings of the 11th Eurographics Workshop on Rendering*, pages 231–242, 2000.

[MN02] Jan R. Magnus and Heinz Neudecker. *Matrix Differential Calculus*. Wiley Series in Probability and Statistics. John Wiley & Sons, 2002.

[PGB03] P. Perez, M. Gangnet, and A. Blake. Poisson image editing. In *Proc. of SIGGRAPH'03*, 2003.

[Poc01] Lynn Pocock, editor. *Proceedings of the 28st International Conference on Computer Graphics and Interactive Techniques (SIGGRAPH2001)*. ACM Press, 2001.

[PSS01] E. Praun, W. Sweldens, and P. Schröder. Consistent mesh parameterization. In Pocock [Poc01], pages 179–184.

[Roc03] Alyn P. Rockwood, editor. *Proceedings of the 30st International Conference on Computer Graphics and Interactive Techniques (SIGGRAPH2003)*. ACM Press, 2003.

[Row97] S. Roweis. Em algorithms for pca and spca. In *Advances in Neural Information Processing Systems*, volume 10, 1997.

[RV05] S. Romdhani and T. Vetter. Estimating 3d shape and texture using pixel intensity, edges, specular highlights, texture constraints and a prior. In *Proceedings IEEE Conference on Computer Vision and Pattern Recognition* [cvp05].

[SACO04] A. Sharf, M. Alexa, and D. Cohen-Or. Context-based surface completion. In *Proc. of SIGGRAPH'04*, 2004.

[SK02] V. Savchenko and N. Kojekine. An approach to blend surfaces. In J. Vince and R. Earnshaw, editors, *Proceedings of CGI'2002 conference, Advances in Modeling, Animation and Rendering*, pages 139–150, June 2002.

[SL94] R. Szeliski and S. Lavallée. Matching 3-d anatomical surfaces with non-rigid deformations using octree-splines. In *IEEE Workshop on Biomedical Image Analysis*, pages 144–153. IEEE Computer Society, 1994.

[Slo04] Dena Slothower, editor. *Proceedings of the 31st International Conference on Computer Graphics and Interactive Techniques (SIGGRAPH2004)*. ACM Press, Agust 8–12 2004.

[SP04] R.W. Sumner and J. Popović. Deformation transfer for triangle meshes. In Slothower [Slo04], pages 399–405.

[Tau95] G. Taubin. A signal processing approach to fair surface design. In *Proc. of SIGGRAPH'95*, 1995.

[TB99] M. E. Tipping and C. M. Bishop. Probabilistic principal component analysis. *Journal of the Royal Statistical Society, Series B*, 21(3):611–622, 1999.

[TF00] J.B. Tenenbaum and W.T. Freeman. Separating style and content with bilinear models. *Neural Computation*, 12(6):1247–1283, 2000.

[Ume91] S. Umeyama. Least-squares estimation of transformation parameters between two point patterns. *PAMI*, 13(4):376–380, 1991.

[VBPP05] D. Vlasic, M. Brand, H. Pfister, and J. Popović. Face transfer with multilinear models. In James L. Mohler, editor, *Proceedings of the 32nd International Conference on Computer Graphics and Interactive Techniques (SIGGRAPH2005)*. ACM Press, July 31–August 4 2005.

[VCBS03] J. Verdera, V. Caselles, M. Bertalmo, and G. Sapiro. Inpainting surface holes. In *Proc. of the IEEE International Conference on Image Processing (ICIP 2003)*, September 2003.

[VJP97] T. Vetter, M.J. Jones, and T. Poggio. A bootstrapping algorithm for learning linear models of object classes. In *Proceedings IEEE Conference on Computer Vision and Pattern Recognition*, pages 40–46. IEEE Computer Society Press, 1997.

[VT02] M.A.O. Vasilescu and D. Terzopoulos. Multilinear analysis of image ensembles: Tensorfaces. In A. Heyden et al., editor, *ECCV 2002*, volume 2350 of *LNCS*, pages 447–460. Springer-Verlag Berlin Heidelberg, 2002.

[WHL+04] Y. Wang, X. Huang, C.-S. Lee, S. Zhang, Z. Li, D. Samaras, D. Metaxas, A. Elgammal, and P. Huang. High resolution acquisition, learning and transfer of dynamic 3-d facial expressions. In M.-P. Cani and M. Slater, editors, *Proceedings of Eurographics 2004*, volume 23(3). The Eurographics Association, Blackwell Publishing, 2004.

[ZOF01] H.-K. Zhao, S. Osher, and R. Fedkiw. Fast surface reconstruction using the level set method. In *Proc. of the IEEE Workshop on Variational and Level Set Methods (VLSM'01)*, 2001.

[ZSCS04] L. Zhang, N. Snavely, B. Curless, and S.M. Seitz. Spacetime faces: High resolution capture for modeling and animation. In Slothower [Slo04], pages 548–558.