Imperial College London

Expression Transfer

Author:
Martin Papanek

Supervisor:
Professor Duncan Gillies

August 30, 2010

Abstract

Acknowledgments

Contents

Chapter 1

Introduction

1.1 Motivation

Allowing computers to understand the world around them is one of the most intriguing goals of computer science. In order to aid humans in day-to-day tasks, the ideal computer should be able to perceive its surroundings, correctly identify the objects and beings around it and act based on this information. Achieving this level of sophisticated, environment-aware behavior is the focus of popular computer science fields such as machine learning, computer vision and logic.

The problem of understanding the surrounding world can be broken down into a number of sub-problems. First the machine must obtain and process the information on its sensors. Then it has to process this data in order to find objects in the sensory input. Finally, the machine has to assign meaning to the scene it perceived based on the the configuration and the properties of the objects it found. This allows the machine to understand what state the environment is in and it may then utilize this information using simple if-then rules.

For humans, all of the aforementioned sub-problems seem simple. However, programming machines to do the same is quite difficult. Computers often do possess better sensors than most humans and thus are readily able to obtain data from sensors. Yet, they are sorely lacking when it comes to locating objects in this sensory input and correctly assessing the properties and configuration of these objects. While it is possible to locate circles and lines, joining these to locate a face or a tree can only be done if the machine knows what a face or a tree should look like. Thus, the machine needs to have prior information about the objects it can expect. This prior information can be encoded in a *model.* The model describes the structure of the object. This, in turn, allows the machine to explain aspects of its sensory input as the occurrence of that object. Finding objects by means of locating an instance of a model in the input is called *model-based recognition.*

Certain objects may also change their shape or appearance. For example a face may transition from a closed eyed state to an open eyed state. An even better example is the body of a human, which is also highly dynamic. These deformable objects are often of special interest to us in our everyday life. A machine should therefore be able to recognize an arbitrarily deformed object and also correctly identify the the degree of deformation, since the amount of deformation may be crucial for the understanding of the scene. The challenge thus lies in constructing an appropriate *deformable model.*

Given a deformable model, the machine then has to process the sensor input and locate instances of the model by adjusting the model's parameters. This task is known as *model-based object recognition*. The parameters of the model then allow the machine to interpret the scene. In addition to locating an instance of the model in one sensory input the machine should also be able to track its movement given a sequence of snapshots of the environment. In *model-based tracking* an instance of the model is identified and tracked from snapshot to snapshot, in spite of deformations of shape and changes in position.

We will describe convenient deformable shape and appearance models and effective algorithms for locating these models. For our sensory snapshots, we will focus exclusively on images. As we will be dealing with images we will investigate motion tracking and feature detection algorithms. These are necessary to obtain cues from our input image which allow us to first locate the model in the image and then track its movements through successive images.

1.2 Contributions

The area where deformable models are applicable is quite large. We will focus on one very interesting application of dynamical models and model extraction which is *expression transfer*. The purpose of expression transfer is to capture the expressions and visemes - speech related mouth articulations - from a video recording of one individual and generate a video of another individual mimicking these expressions and visemes. The alternative to transferring these expressions would be to construct a physical model of the face and simulate the observed expressions and visemes using the model. However, transferring the dynamics of a subjects face to that of another enables us to create very realistic animations without much intervention. On the other hand, it is quite difficult to generate realistic expressions by setting the appropriate parameters of a physical model simply because of the complexity of the physics behind the movement that is responsible for expressions.

Chapter 2

Background

Considerable amount of research has been done on using models to characterize a deformable object as well as on model-based object recognition and tracking. In this chapter we will present an overview of various modeling techniques. Likewise, we will discuss computer vision algorithms which will allow us to locate the features of interest in the image. These features then make it possible to fit an instance of the model to the image.

2.1 Deformable Models

Models give the computer prior information about the structure of a class of objects. Most real world objects have a dynamic structure. To allow the computer to locate such dynamic objects it is necessary to account for this variability by giving the computer prior information about the possible variations. A deformable model describes the expected structure of a class of objects while at the same time allowing for variations from the expected structure. Thus, if a somewhat deformed instance of this class of objects is present in the image, the deformable model will be able to explain this is a deformation from the expected shape.

The quality of a deformable model can be assessed based on two important characteristics. First, the model should be *general* enough so that it is capable of representing any realistic deformation of the object. On the other hand, the computer should only to be able to find an instance of this model in its surroundings if and only if this object is present. The deformable model therefore has to be *specific* so that it does not locate non-existent instances of a model in the input. Clearly, a model that is too general will inadvertently fit objects from a different class and vice versa a model that is too specific will not be able to explain all the variations of the object structure and thus fail to locate instances of the correct class. The optimal deformable model should balance these two attributes.

The variance in shape or appearance of a model may be due to combination of the following factors:

- Variations of shape due to deformations of the object.
- Arbitrary scaling of the object, possibly due to distance from the observer.
- Arbitrary rotations of the object which may cause occlusions.

- Some measure of Gaussian input noise.
- Differences in color or intensity caused by a change in lighting conditions.

Some of these sources of variation may be explained away using standard machine vision techniques. Noisy input can be explained by means of Gaussian filtering. Other sources of variation such as those due to deformations of the object need to be represented by appropriate parameters of the model. The deformable model simulates the deformations based on the parameter values. The model thus needs to learn the mapping from parameters to deformations. The two most prevalent approaches to discovering this mapping are using either *physical models* or *statistical models.*

Physical models construct deformable models which mimic the elastic deformations of the class of object they model. The parameters of a physical model control the amount of actual physical deformation.

Statistical deformable models are trained on a set of examples of a class of objects. Through statistical analysis a basis for the deformations observed in this set of examples is constructed. This basis then allows the model to predict probable instances of the objects of this class.

An important subgroup of statistical models are the *tensor models.* The tensor models are obtained as a generalization of statistical analysis techniques in higher dimensions.

Finally, given that a deformable model is uniquely defined by its parameters, the goal of model-based object recognition is then to fit an instance of a model to an appropriate object in the image. The problem of fitting the model to the image is in essence an optimization problem. In order to fit the model we need to correctly adjust parameters that control the model. In addition to determining the intrinsic parameters of the model it is necessary to also find how the object is rotated, moved and scaled to explain the variation in structure which is not caused by the deformations. The recognition algorithm also needs to be robust in order to deal with variations caused by noise.

2.1.1 Physics Based Models

Natural objects all obey physical laws. Human and animal bodies change shape when their muscles contract and loosen, which in turn alters the shape of the elastic soft tissues which surround the muscles. Movement is further constrained by the skeletons and gravity. Due to the dynamic nature of such objects it is quite impractical to model the structure of these objects as consisting of only rigid components. To cope with these highly dynamic bodies researchers have turned to physics to describe the rules governing these dynamics in form of a set of equations. Pentland and Sclaroff in [22] give a closed form solution for extracting a physical model from images. Terzopoulos and Metaxas combine physics-based deformable models with Kalman filtering theory in [19].

As the name suggests, physical models emulate the physical laws that govern deformations. Physical models are predominantly formulated using the finite element analysis. An in depth discussion of finite element analysis can be found in [1].

2.1.1.1 Finite Element Method (FEM)

The FEM method is a numerical engineering technique for the simulation of dynamic behavior of solids and structures. With the finite element method the assumption is made that the object alters shape as if it were made of a clay-like material. In physics, *strain* is the measure of deformation or displacement from a rigid body state. The stiffness of the material determines responses to strain and stress and thus describes the degrees of freedom of the material.

The underlying idea behind FEM is that the object is approximated as an assemblage of elements of finite size. These elements are interconnected with each other by nodal points throughout the object. The structure of an object is thus discretized into a mesh of N finite elements. The elements are placed next to each other so that no gaps remain in between. When the object undergoes a deformation, this in turn propagates through the mesh of finite elements which themselves deform accordingly. The displacements within these elements are assumed to be a function of the displacements measured at the nodal points. This assumption is fundamental for the FEM and can be formalized as

$$u^{(m)}(x,y,z) = \mathbf{H}^{(m)}(x,y,z) * \mathbf{U} \tag{2.1}$$

where $u^{(m)}$ is the displacement at x,y, and z coordinate within the element m, $\mathbf{H}^{(m)}$ the displacement interpolation matrix and $\mathbf{U}$ the global displacement measured at every nodal point. With equation 2.1 the displacements at any point in the object can be calculated. The values of the displacement interpolation matrix $\mathbf{H}^{(m)}$ depend on the choice of the finite elements that make up the mesh.

From the displacement $u^{(m)}(x,y,z)$, the strain can be calculated as the derivative of the displacement with respect to x,y, and z. Thus, the strain $\epsilon^{(m)}(x,y,z)$ at the element m is given by

$$\epsilon^{(m)}(x,y,z) = \mathbf{B}^{(m)}(x,y,z) * \mathbf{U} \tag{2.2}$$

where $\mathbf{B}^{(m)}(x,y,z)$ is the strain-displacement matrix. This matrix can be calculated by differentiating the displacement interpolation matrix $\mathbf{H}^{(m)}$. With equations 2.1 and 2.2 the behavior of the object structure given a global nodal point displacement $\mathbf{U}$ is defined.

The goal of displacement-based finite element analysis is to calculate unknown nodal point displacements from a known force or load acting on the object. When a load is applied to the structure of the object it will cause a deformation of the mesh. The nodal points in the mesh will bounce and move until they reach a state of equilibrium. It is important to stress that in this equilibrium the shape of the object is still deformed due the applied force. However, the nodal points may have assumed new stable positions hence we refer to it as an equilibrium. The equation governing the equilibrium is derived using equations 2.1, 2.2 and the Principle of Virtual Work [1]. It relates the stiffness matrix $\mathbf{K}$ and the unknown nodal displacements $\mathbf{U}$ to the loads $\mathbf{R}$ as follows

$$\mathbf{KU} = \mathbf{R} \tag{2.3}$$

The stiffness matrix $\mathbf{K}$ is calculated as the sum of the stiffness matrices $\mathbf{K}^{(m)}$ of the individual finite elements. The $\mathbf{K}^{(m)}$ are computed from the strain-displacement matrices $\mathbf{B}^{(m)}$ and the elasticity matrix $\mathbf{E}^{(m)}$ as

$$\mathbf{K} = \sum_m \mathbf{K}^{(m)} = \sum_m \int_{V^{(m)}} \mathbf{B}^{(m)T}\mathbf{E}^{(m)}\mathbf{B}^{(m)} dV^{(m)} \tag{2.4}$$

where the integral goes over the volume $V^{(m)}$ of the element m. This approach to computing the stiffness matrix is known as the *direct stiffness method.* The displacement interpolation matrix and strain-displacement interpolation matrix are constructed for each finite element. Their calculation depends on the displacement interpolation function seen in equation 2.1. Fortunately, these functions have a well defined formulation which depends on the degrees of freedom (nodal point connections) of the element. This formulation, along with numerical integration techniques for calculating the stiffness matrix can be found in chapter 5 of [1]. The elasticity matrix relates stress and strain in the material. Its form depends on the dimensionality of the element. In a one dimensional element the elasticity matrix is a scalar known as *Young's Modulus.* The value of Young's Modulus describes the elastic properties of materials and is computed as the constant ratio between stress and strain in the material.

Equation 2.3 describes a static equilibrium at a specific point in time. If the loads are applied rapidly then element inertial forces and energy damping forces must be considered as well. Equation 2.5 gives the dynamic form of the FEM equilibrium equation where $\dot{\mathbf{U}}$, $\ddot{\mathbf{U}}$ are the first and second time derivative of $\mathbf{U}$ respectively.

$$\mathbf{M}\ddot{\mathbf{U}} + \mathbf{C}\dot{\mathbf{U}} + \mathbf{K}\mathbf{U} = \mathbf{R} \tag{2.5}$$

Here $\mathbf{M}$ is the mass matrix and $\mathbf{C}$ the damping matrix. This differential equation is often referred to as the FEM governing equation. To solve for the unknown nodal displacements $\mathbf{U}$ standard techniques for solving differential equations can be used.

2.1.1.2 Modal Analysis

Solving the dynamic equilibrium equation 2.5 by direct integration is very costly [1]. However, it is possible to diagonalize the system of equations by changing the finite element displacement basis to a generalized displacement basis Φ as

$$\mathbf{U} = \Phi\tilde{\mathbf{U}} \tag{2.6}$$

The advantage of the new generalized basis can be seen when equation 2.5 is premultiplied from the left by Φ^T to give

$$\Phi^T\mathbf{M}\Phi\ddot{\tilde{\mathbf{U}}} + \Phi^T\mathbf{C}\Phi\dot{\tilde{\mathbf{U}}} + \Phi^T\mathbf{K}\Phi^T\tilde{\mathbf{U}} = \Phi^T\mathbf{R} \tag{2.7}$$

The above equation can be diagonalized if Φ is chosen as consisting of the n eigenvectors of the eigensolutions (ω_1^2, ϕ_1) , ... , (ω_n^2, ϕ_n) which solve the eigenproblem

$$\mathbf{K}\Phi = \Omega^2\mathbf{M}\Phi \tag{2.8}$$

so that

$$\Phi^T\mathbf{K}\Phi = \Omega^2 \tag{2.9}$$

$$\Phi^T\mathbf{M}\Phi = \mathbf{I} \tag{2.10}$$

These eigenvectors are the free vibrational modes of the equilibrium equation. Choosing Φ in this manner transforms equation 2.7 into

$$\ddot{\tilde{\mathbf{U}}} + \tilde{\mathbf{C}}\dot{\tilde{\mathbf{U}}} + \Omega^2\tilde{\mathbf{U}} = \tilde{\mathbf{R}} \tag{2.11}$$

Under the assumption that the transformed damping matrix $\tilde{\mathbf{C}}$ is diagonal as well the above equation 2.11 is diagonalized since Ω^2 is a diagonal matrix. Diagonalizing decouples the individual components of $\mathbf{U}$. This means that we obtain a separate differential equation for each component of $\mathbf{U}$ and as such it is not necessary to compute the inverse of the $\mathbf{K}$. Computing this inverse is costly and may not always be possible if $\mathbf{U}$ is singular. Thus, the diagonalized equation 2.11 can be solved for the displacements $\tilde{\mathbf{U}}$ either in closed form or integrated numerically in fewer steps.

2.1.1.3 Recovering shape with FEM

Through Modal analysis it is possible to easily generate displacements of an arbitrary object given a stiffness matrix $\mathbf{K}$. These displacements deform the object. Since the matrix Φ encodes the free vibration modes, it is possible to generate a new face by taking a vector of weights $\mathbf{u}$ and calculating a deformed instance of the object class by adding the displacements to the mean shape $\bar{\mathbf{x}}$ using

$$\mathbf{x} = \bar{\mathbf{x}} + \Phi\mathbf{u} \tag{2.12}$$

Equation 2.12 synthesises a new object from a vector of parameters $\mathbf{u}$. It is therefore possible to fit a model of a class of objects with a known stiffness function to a candidate object in an image by solving an optimization problem which locates the parameters $\mathbf{u}$.

Alternatively, it is also possible to use the equilibrium equation 2.3 to recover the shape of the candidate object. Assuming that the positions of the nodal points are found in the image and that their 3D position is recovered then the load vector $\mathbf{R}$ can be calculated as

$$[r_{3k}, r_{3k+1}, r_{3k+2}] = [x_k^w, y_k^w, z_k^w] - [x_k, y_k, z_k] \tag{2.13}$$

where x_k^w, y_k^w and z_k^w is the position obtained from the image of the nodal point with index k. The rest position of this nodal point in the model is given by x_k, y_k and z_k. This approach to constructing the load vector $\mathbf{R}$ is analogous to attaching springs between the nodal point measurements and their corresponding points in the model. Given the load vector the goal is now to solve the previously seen equilibrium equation

$$\mathbf{KU} = \mathbf{R} \tag{2.14}$$

To solve for the displacements $\mathbf{U}$ the stiffness matrix $\mathbf{K}$ needs to be inverted. This is often a very costly operation which can be avoided by diagonalizing the equilibrium equation. This is done using modal analysis with the generalized basis from equation 2.8.

2.1.1.4 Combining Statistical and FEM modes

2.1.2 Statistical Models !ALTERED

Statiscal models attempt to understand and model the variation inherent in the deformable object through stastical analysis. In statistical analysis a data set is examined and characterised so that predictions and judgements based on the underlying data distributions can be made. To use the techniques of statistical analysis to build a model it is necessary to have a dataset to analyze. This dataset is made up of many instances of the class of objects we want to model. These instances are usually given as a set of vertices (either 2D or 3D) which define the object.

The less sophisticated models attempt to model the variation of the data that defines the shape of an object. A *shape model* describes the boundaries of an object. For instance, a shape model of a face will denote the location and measure of the defining contours of a face.

The more sophisticated models perform statistical analysis on the entire appearance of the object – which includes variance coming from the texture, lighting etc. This *appearance model* describes the location and color of the pixels that define the object.

2.1.2.1 PCA and Eigenfaces !ALTERED

In 1991, Turk and Pentland [30] pioneered a 2D model-based face recognition approach based on the statistical technique called *principal component analysis* (PCA). The PCA is an orthogonal projection of the data onto a lower dimensional linear space such that the variance of the projected data is maximized. The PCA therefore identifies the primary directions in which data varies, since these are the vector directions which maximize variance. These vectors span the *principal subspace*. The formula to compute the principal directions is derived from maximizing the variance of a projection of the data [3]. By solving this maximization problem we find that variance in the M-dimensional principal subspace, spanned by the vectors $\mathbf{u}_i$, is given by

$$\mathbf{S}\mathbf{u}_i = \lambda_i \mathbf{u}_i \tag{2.15}$$

where $\mathbf{S}$ is the $D \times D$ covariance matrix of the D-dimensional data set. From 2.15 we see that the vectors $\mathbf{u}_i$ are the eigenvectors, and that the λ_i are the eigenvalues of the covariance matrix of data set. In order to maximize the variance of the M-dimensional principal subspace, given by this equation, we pick the M eigenvectors which correspond to the largest eigenvalues. These M eigenvectors are the principal directions which span the M-dimensional principal subspace. If $D = M$, then there is no dimensionality reduction and the PCA reduces to just a rotation of the coordinate axes to align with the axes of the principal subspace. To express PCA as a linear transformation we combine the M eigenvectors into a matrix $\mathbf{\Phi}_{pca} = [\mathbf{u}_1 \ldots \mathbf{u}_m]$ to get

$$\mathbf{S}\mathbf{\Phi}_{pca} = \mathbf{\Phi}_{pca}\mathbf{\Lambda} \tag{2.16}$$

Given the D-dimensional data set represented as a matrix $\mathbf{D}$, then the PCA transformation matrix $\mathbf{\Phi}_{pca}$ can be calculated in two equivalent ways. Either we compute the covariance matrix $\mathbf{S} = \mathbf{U}^T\mathbf{U}/(D-1)$, where $\mathbf{U}$ is the mean centered data set matrix. Or the PCA transformation matrix is obtained from the singular value decomposition of the the mean centered data set as $\mathbf{U} = \mathbf{\Phi}_{pca}\Sigma\mathbf{V}$.

Turk and Pentland's face recognition scheme uses a data set of images to learn what they call *eigenfaces*. Expressed in mathematical terms – the eigenfaces are eigenvectors of the 2D image space. As seen from the PCA analysis, these eigenvectors represent the axes of a space spanning the 2D face data set, which are responsible for any significant variation. To obtain the eigenfaces they compute Turk and Pentland SVD decomposition of this covariance matrix. Any individual face from the training data set can then be exactly represented as a linear combination all the eigenfaces.

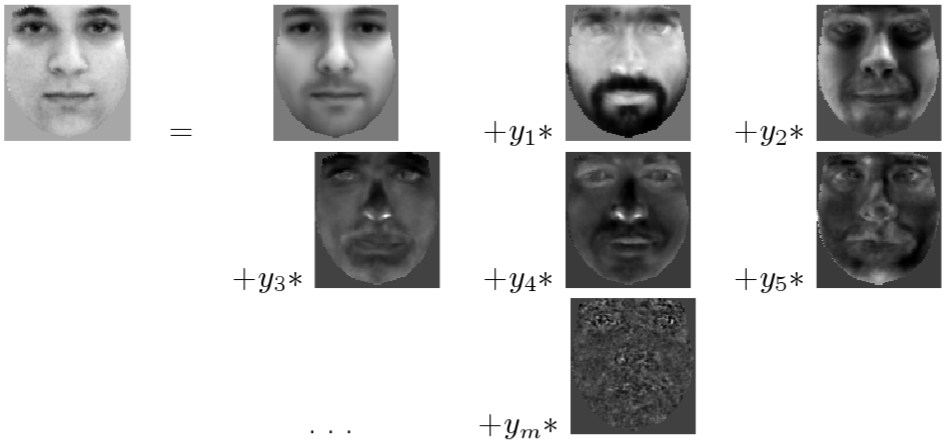


Figure 2.1: Using the eigenfaces we can represent an image as a linear combination of the eigenfaces. Taken from [24]

The weight of each eigenface in the linear combination is computed from the projection of the face image onto this eigenface. To recognize a face we first calculate these weights. Then we calculate the Euclidean distance of the vector of weights to the weights of faces from our training data set. The face from the data set with weights which are closest to the new image's weights is chosen as a match.

In an image of 256 by 256 we have a 65,536 dimensional vector that represents the face image. This means we would require 65,536 eigenfaces to be able to exactly represent every face. However, images of faces are very similar in their configuration which means that the underlying principal subspace of faces has a lower dimension than 65,536. To reduce the dimensionality, PCA is performed to identify the subset of eigenfaces which span a lower dimensional principal subspace. Using this subset of eigenfaces, which are called the principal eigenfaces, we can effectively encode a 65,536 dimensional face image vector using a vector of much smaller dimensions.

The drawback of a PCA based model is that the recognition rate drops significantly once independent sources of variation are introduced. Turk and Pentland noted that the eigenfaces approach has issues with variations in lighting, head size, head orientation or faces exhibiting expressions [30]. Likewise, when faces are partially occluded in images it causes difficulties to the technique.

The eigenfaces approach is based heavily information and coding theory. As explained, the PCA makes it possible to encapsulate a face image using low dimensional vector. As such PCA is often used to reduce dimensionality in more sophisticated modeling approaches.

2.1.2.2 Morphable Model

Blanz et al use PCA to construct a statistical model of 3D face shape and texture [4], [5]. The orthogonal matrix of basis vectors is extracted from a database of 3D faces with texture. This basis spans a vector space which Blanz refers to as the *Morphable Model.*

The faces in the database are encode as shape vectors with S_i to being the vector representing the 3D shape of a human face, stored as the x, y, z-coordinates of all the vertices of that face. So

$$\mathbf{S_i} = (x_1, y_1, z_1, x_2, ..., x_n, y_n, z_n)^T \tag{2.17}$$

Similarly, the texture vectors are defined as

$$\mathbf{T_i} = (R_1, G_1, B_1, R_2, ..., R_n, G_n, B_n)^T \tag{2.18}$$

The idea behind the morphable model is that linear combination of the $\mathbf{S_i}$ and $\mathbf{T_i}$, which is within ±3 standard deviations from the mean, constitutes a realistic face. The full morphable model is the linear combination of the examples given by the formula

$$\mathbf{S} = \sum_{i=1}^{N} a_i \mathbf{S_i} \qquad \mathbf{T} = \sum_{i=1}^{N} b_i \mathbf{T_i} \tag{2.19}$$

Blanz et al also performed a Principal Component Analysis (PCA) on the dataset of examples to produce orthogonal basis vectors that can be used in the morphable model instead of the examples directly. This reduces the dimensionality of the model. The eigenvectors given by $\mathbf{s_i}$ and $\mathbf{t_i}$ can be used to generate new faces as follows

$$\mathbf{S} = \bar{\mathbf{s}} + \sum_{i=1}^{m} \alpha_i . \mathbf{s_i} \qquad \mathbf{T} = \bar{\mathbf{t}} + \sum_{i=1}^{m} \beta_i . \mathbf{t_i} \tag{2.20}$$

where $\bar{\mathbf{s}}$, $\bar{\mathbf{t}}$ are the means of the shape and texture data and $m < N$. The parameters α_i and β_i are the shape and texture parameters controlling the deformable model.

The PCA process identifies vectors which represent the most variance in the examples and as such contain the most information about the images. This means any image can now be represented by the linear combination of a smaller number of these components than would be required of the original examples. The linear combination is controlled by the parameters of the deformable model. This is in essence the eigenfaces modelling approach only now applied to 3D shapes and textures.

The shape vector and texture vector encode the location and texture values at specific feature (landmark) points. Therefore, to allow the morphable model to fit to the source image, a point-to-point correspondence must be ensured on the example images, so that important features such as the tip of the nose are correctly matched up in the examples. During the recognition process the features in the source image are matched to those in the examples, in order to calculate the correct combinations of the basis vectors. According to Blanz, to fit the Morphable Model of 3D faces to an image, the user has to manually select about 7 feature points.

2.1.2.3 Combined Models of Shape and Appearance !ALTERED

Texture variations and shape variations are correlated. Clearly, with any source of lighting it is the shape of an object which influences what parts of the object will appear darker or lighter. Cootes and Taylor [10] describe modeling approaches which account for the correlations between the two sources of variation can be modelled. The models they introduce are well suited for highly variable structures such as faces or internal organs. Using PCA based shape and texture models, Cootes and Taylor construct Active Shape and Active Appearance Models which, despite the name, are model fitting algorithms that find the parameters statistical models to fit instances of objects in images.

The shape model are constructed by annotating feature points in images, collecting these into a data matrix and then performing PCA on the covariance matrix of this data to generate a lower dimensional basis $\mathbf{P}_s$. The shape model then makes it possible to synthesise new shapes as follows

$$\mathbf{S} = \bar{\mathbf{s}} + \mathbf{P}_s\mathbf{b_s} \tag{2.21}$$

To construct the texture model, the image is first warped into the mean shape $\bar{s}$ to obtain a “shape-free” image. The gray-scale values at points in the shape-free which correspond to feature points in the mean shape are then measured and combined into a texture data matrix. Then PCA is used to find eingenvector matrix $\mathbf{P}_t$. This matrix of eigenvectors gives the linear model for texture as

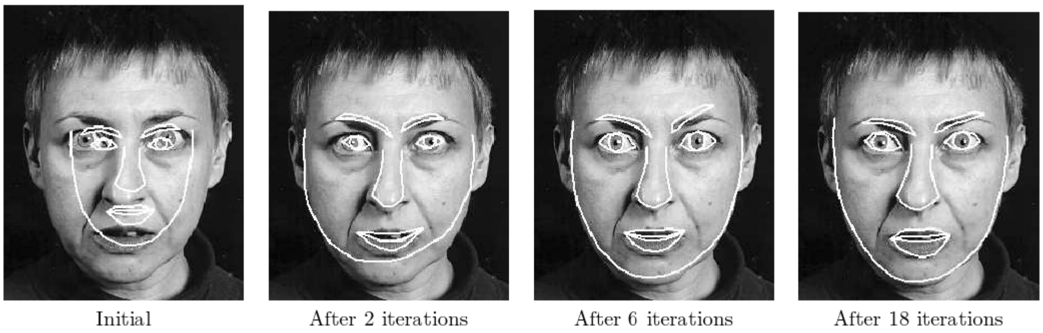
$$\mathbf{T} = \bar{\mathbf{t}} + \mathbf{P}_t\mathbf{b_t} \tag{2.22}$$

So similarly to the approach of Blanz as seen in formula 2.20, Cootes and Taylor separate the shape and the information of a target image into two distinct vectors – the shape parameter vector $\mathbf{b_s}$ and the texture represented by the grey-level vector $\mathbf{b_t}$ (here the image is gray-scaled). To obtain a model which addresses both shape and texture, these parameter vectors are combined into one vector $\mathbf{b} = [\mathbf{b_s}\,\mathbf{b_t}]^T$. Then PCA is applied to this combined vector to get an eigenvector basis $\mathbf{P_c} = \left(\begin{smallmatrix} P_{cs} \\ P_{ct} \end{smallmatrix}\right)$ for the parameters $\mathbf{b_s}$ and $\mathbf{b_t}$. This new basis encapsulates the correlation between shape and texture, thus making it possible to express the other parameters as functions of a parameter vector $\mathbf{c}$ as follows

$$\mathbf{S} = \bar{\mathbf{s}} + \mathbf{P}_s\mathbf{W}_s^{-1}\mathbf{P}_{cs}c \tag{2.23}$$
$$\mathbf{T} = \bar{\mathbf{t}} + \mathbf{P}_t\mathbf{P}_{ct}\mathbf{c} \tag{2.24}$$

where $\mathbf{W}_s$ is a diagonal matrix of weights which allows the shape and texture models to have different units. To reconstruct a new face using the model we first determine the $\mathbf{b_s}$ and $\mathbf{b_g}$ parameters from the new face image. These parameters are then used to calculate the c using by projecting them using the eigenvector basis as $\mathbf{c} = \mathbf{P_c}^T\mathbf{b}$.

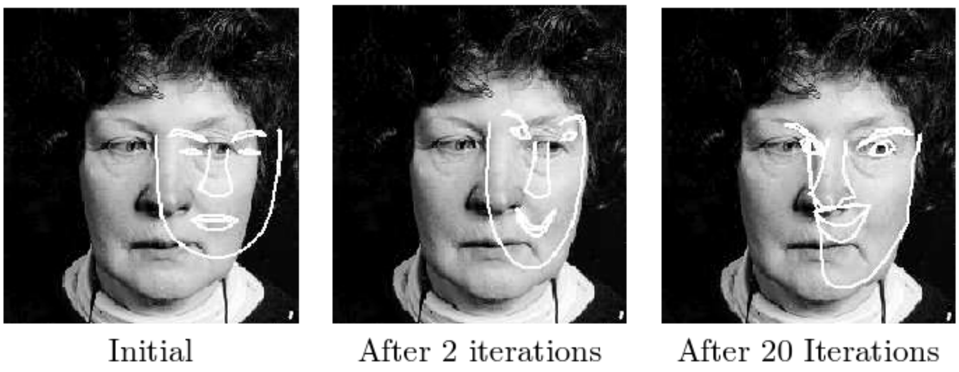
Active Shape Model. The Active Shape model (ASM) proposed by Cootes and Taylor is model fitting algorithm based on the statistical models of shape and texture. The ASM enables us to locate a shape in an image using an iterative procedure as shown in figure 2.2. To search for objects in an image using the ASM, we first place the shape model, which we obtained from the training data, into the centre of the image. In the next iterations a texture profile of k neighboring points around each point of the shape model is taken and compared with the texture profile of this point obtained when training the model. Then the point of the shape model is moved to make the difference between the two profiles smaller. This process is repeated until convergence is reached.



Initial After 2 iterations After 6 iterations After 18 iterations

Figure 2.2: Active Shape Model iterations shown on finding the shape of a previously unseen face. Taken from [9]

The ASM moves the point only based on the texture profile of a local neighborhood of this point. The obvious problem with this approach is that it is a local search technique and thus it depends too much on the choice of the starting point. If the target image is not centered on the face and the algorithm begins in the center of the image, then it can possibly converge on incorrect shapes such as the nose as can be seen in figure 2.3. This is due to the fact that as a local search method, the ASM cannot escape a local minimum.



Initial After 2 iterations After 20 Iterations

Figure 2.3: Incorrect convergence of the ASM when the search is initialized incorrectly. This can happen when the face is not centered in the image or when the search does not begin in the center of the image. Taken from [9]

To improve the efficiency and the robustness of ASM, Cootes suggests performing several searches at different resolution. First a coarse search is performed on the image with a low resolution until the search converges. The resulting configuration of the shape model points from this coarse search then serves as the starting point for a search in the same image with a better resolution. The advantage of adopting this coarse to fine search approach is that it makes the search less susceptible to converging on incorrect shapes. When searching at a lower resolutions it will be easier to find the outlines of the face. Once the search is restarted with a higher resolution, shapes like the mouth and the nose can be identified.

Active Appearance Model. The Active Appearance Model (AAM) is a model fitting technique based on a combined shape and texture model defined by equations 2.23 and 2.24. The AAM enhances the Active Shape model search by also considering the texture information in the entire object.

The AAM search attempts to minimize the difference between the texture (the grey-levels) of the image we are searching and the grey-levels of the image generated with the current model parameters. The difference vector which is minimized every iteration is defined as:

$$\delta \mathbf{I} = \mathbf{I}_i - \mathbf{I}_m \tag{2.25}$$

where $\mathbf{I}_m$ is the vector of the grey-levels generated with the current model parameters and $\mathbf{I}_i$ is a vector containing the grey-levels of the image. The goal of AAM is to minimize the difference $\delta \mathbf{I}$ between the synthesized and observed grey-levels.

To simplify this optimization problem, Cootes and Taylor propose to allow the computer to learn the relationship between the difference vector $\delta \mathbf{I}$ and the error in the model parameters. Learning this relationship makes it possible to correctly improve the model for a given measured error in an iterative procedure.

The AAM iterations are depicted in figure 2.4. The process attempts change model parameters to fit it to an unseen face. This fitting is done by minimizing the differences between pixels at the landmark points.

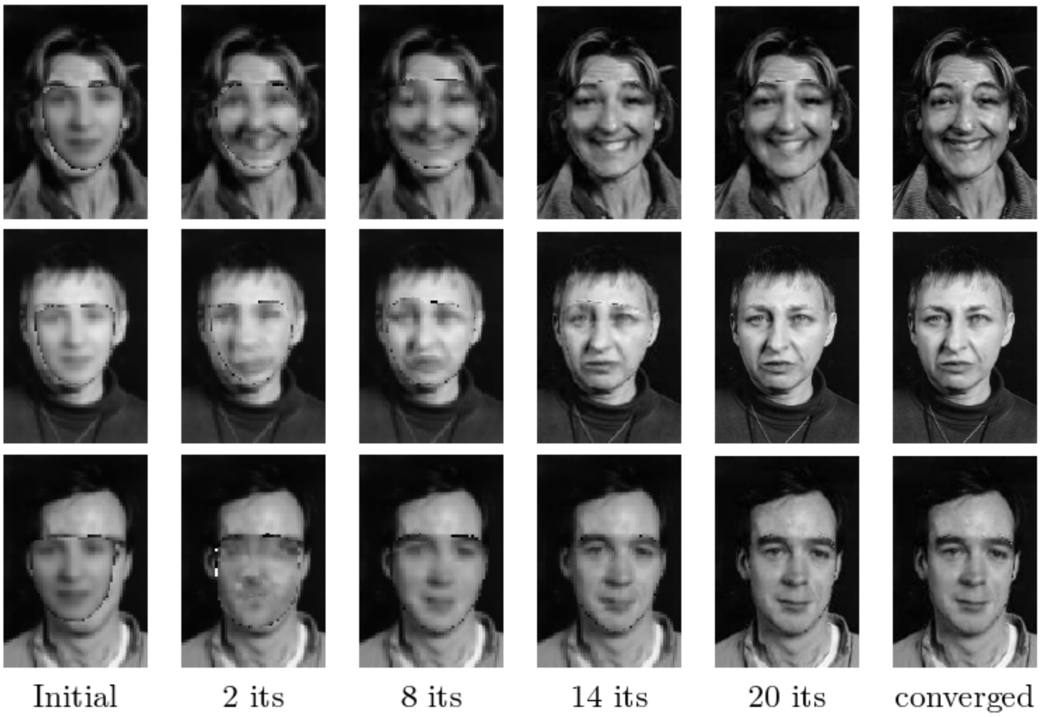


Figure 2.4: Active Appearance Model iterations shown when searching for previously unseen faces. Taken from [10]

The drawback of both the ASM and AAM search is that it is required of the user to specify landmark points on the target image. This means the technique is not well suited for applications where the model would have to be matched to a large number of faces. To combat this problem, a number of automatic 2D and 3D landmark location solutions can be used [10].

Another problem of both the ASM and the AAM lies in the fact that they do not distinguish or model the individual factors responsible for shape or texture variation. They only acknowledge two sources of variation in the image. In the case of face models, this means that variation caused by identity and the variation caused by expression are not perceived as two fundamentally different sources. Thus, it is impossible to only alter one variational source while leaving the other source constant. This makes the models unsuitable for expression transfer.

2.1.3 Tensor Models

Vasilescu and Terzopoulos [31], [33], [32] defined a generalization of statistical methods using multilinear algebra. Statistical analysis of images using PCA determine the variation of the training dataset. The drawback of the PCA is that it only addresses one factor – one source of variance in the images. All variation is explained using the orthogonal eigenvectors and the eigenvalues. With images of faces, the PCA approach works well when only the identity or only the expression varies. Using methods of multilinear analysis it is possible to model the interaction of several sources of variation.

The multilinear approach has mostly replaced the PCA method and will be also used in the expression transfer application we develop. Here we will go over an overview of multilinear algebra to elucidate the advantages of multilinear models. Details concerning design and implementation specifics of the tensor based model will be covered in chapter 3. A comprehensive discussion of multilinear algebra and its application was compiled by De Lathauwer in [17]. Detailed discussion of matrix calculations and matrix calculus can be found in [12], [16], [27].

2.1.3.1 Multilinear Algebra

Multilinear algebra is based on the generalization of matrix operations. Matrices are linear operators and can be seen as linear functions defined over vector spaces. In multilinear algebra the operators are *tensors.*

Tensors. In multilinear algebra tensors define multilinear operators over a set of vector spaces. Tensors are organized into mode spaces. The N-mode tensor contains N mode spaces and each item of data contained in the tensor is indexed by N values. The number of modes determines the order of a tensor, so an N-mode tensor has order N. If $\mathcal{A}$ is a tensor of order N, then $\mathcal{A} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_N}$ where I_i are the mode spaces. Vectors are 1^{st} order tensors, matrices are 2^{nd} order tensors. In a matrix the first mode space is the row space, the 2-mode space is the column space.

Tensor Flattening. Any tensor can be flattened into a matrix. Tensor flattening (or matrix unfolding) is defined as follows; given an Nth-order tensor $\mathcal{A} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_N}$, this tensor can be flattened along mode $i \in \{1, 2, \ldots, N\}$ to give $\mathbf{A_{(i)}} \in \mathbb{R}^{I_i \times I_1 I_2 \ldots I_{i-1} I_{i+1} \ldots I_N}$.

The flattening corresponds to aligning the vectors from the mode space i into a matrix. To illustrate tensor flattening lets define the 3^{rd} order tensor $\mathcal{B} \in \mathbb{R}^{3 \times 2 \times 3}$ by $b_{111} = b_{112} = b_{113} = b_{211} = b_{123} = -b_{212} = -b_{313} = 1$, $b_{311} = b_{122} = b_{322} = b_{323} =$

$-b_{213} = -b_{222} = 2$, $b_{312} = 0$, $b_{121} = b_{221} = 4$ and $b_{223} = -5$. Then tensor flattening along mode 1 is given by

$$\mathbf{B_{(1)}} = \left(\begin{array}{ccc|ccc} 1 & 1 & 1 & 4 & 2 & 1 \\ 1 & -1 & -2 & 4 & -2 & -5 \\ 2 & 0 & -1 & 2 & 2 & 2 \end{array}\right)$$

The flattening along mode 2 are the vectors along the second mode space

$$\mathbf{B_{(2)}} = \left(\begin{array}{ccc|ccc|ccc} 1 & 1 & 2 & 1 & -1 & 0 & 1 & -2 & -1 \\ 4 & 4 & 2 & 2 & -2 & 2 & 1 & -5 & 2 \end{array}\right)$$

Tensor flattening is useful since it makes it possible to express tensor operations using matrices. The schema for flattening a 3^{rd} order tensor is shown in figure 2.5

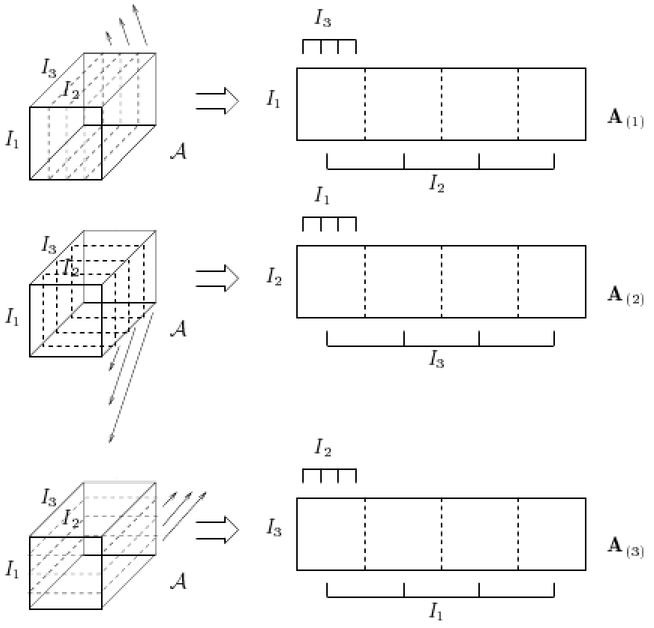


Figure 2.5: Flattening of the $(I_1 \times I_2 \times I_3)$-tensor $\mathcal{A}$ into three matrices; the $(I_1 \times I_2 I_3)$-matrix $\mathbf{A_{(1)}}$, the $(I_2 \times I_3 I_1)$-matrix $\mathbf{A_{(2)}}$ and the $(I_3 \times I_1 I_2)$-matrix $\mathbf{A_{(3)}}$. Taken from [17].

Kronecker product. The Kronecker Product between a $m \times n$ matrix A and a $p \times q$ matrix B is a $(mp) \times (nq)$ matrix defined as

$$A \otimes B = \begin{pmatrix} a_{11}B & a_{12}B & \dots & a_{1n}B \\ a_{21}B & a_{22}B & \dots & a_{2n}B \\ \vdots & & & \\ a_{m1}B & a_{m2}B & \dots & a_{mn}B \end{pmatrix} \tag{2.26}$$

Properties. The Kronecker product possesses a number of useful properties. First the connection between matrix product and kronecker product is given by

$$(A \otimes B)(C \otimes D) = (AC \otimes BD) \tag{2.27}$$

where A is a $m \times n$ matrix, B a $p \times q$ matrix, C a $n \times r$ matrix and D a $q \times s$ matrix.

Proof. The proof comes from block multiplication of the two Kronecker products as follows

$$\begin{aligned}(A \otimes B)(C \otimes D) &= \begin{pmatrix} a_{11}B & a_{12}B & \dots & a_{1n}B \\ a_{21}B & a_{22}B & \dots & a_{2n}B \\ \vdots & & & \\ a_{m1}B & a_{m2}B & \dots & a_{mn}B \end{pmatrix} \begin{pmatrix} c_{11}D & c_{12}D & \dots & c_{1r}D \\ c_{21}D & c_{22}D & \dots & c_{2r}D \\ \vdots & & & \\ c_{n1}D & c_{n2}D & \dots & c_{nr}D \end{pmatrix} \\ &= \begin{pmatrix} (\sum_{j=1}^n a_{1j}c_{j1})DB & (\sum_{j=1}^n a_{1j}c_{j2})DB & \dots & (\sum_{j=1}^n a_{1j}c_{jr})DB \\ (\sum_{j=1}^n a_{2j}c_{j1})DB & (\sum_{j=1}^n a_{2j}c_{j2})DB & \dots & (\sum_{j=1}^n a_{2j}c_{jr})DB \\ \vdots & & & \\ (\sum_{j=1}^n a_{mj}c_{j1})DB & (\sum_{j=1}^n a_{mj}c_{j2})DB & \dots & (\sum_{j=1}^n a_{mj}c_{jr})DB \end{pmatrix} \\ &= (AC \otimes BD)\end{aligned}$$

The transpose of the Kronecker product is the transpose of the component matrices.

$$(A \otimes B)^T = A^T \otimes B^T \tag{2.28}$$

Proof. The proof can be seen from block transposing the Kronecker product.

$$(A \otimes B)^T = \begin{pmatrix} a_{11}B & \dots & a_{1n}B \\ a_{21}B & \dots & a_{2n}B \\ \vdots & & \\ a_{m1}B & \dots & a_{mn}B \end{pmatrix}^T = \begin{pmatrix} a_{11}B^T \dots & a_{m1}B^T \\ a_{12}B^T \dots & a_{m2}B^T \\ \vdots & \\ a_{1n}B^T \dots & a_{mn}B^T \end{pmatrix} = A^T \otimes B^T$$

If the matrices A and B are invertible then the Kronecker product is invertible as well. The inverse of the Kronecker product is given by

$$(A \otimes B)^{-1} = A^{-1} \otimes B^{-1} \tag{2.29}$$

Proof. The inverse formula is derived from property 2.27.

$$(AA^{-1} \otimes BB^{-1}) = (A \otimes B)(A^{-1} \otimes B^{-1}) = I$$

thus $(A^{-1} \otimes B^{-1})$ is the inverse of the Kronecker product.

From 2.27 and 2.28 we can see that the Kronecker product of two orthogonal matrices is again orthogonal.

$$(A \otimes B)^T(A \otimes B) = (A^T \otimes B^T)(A \otimes B) = (A^TA \otimes B^TB) = (I \otimes I) = I$$

and similarly to show for $(A \otimes B)(A \otimes B)^T = I$.

Frobenius Norm. The Frobenius-norm of a tensor is a measure of the size of the tensor. It is defined as

$$\|\mathcal{A}\| = \sqrt{\sum_{i_1}\sum_{i_2}\dots\sum_{i_N} a^2_{i_1 i_2 \dots i_N}} \tag{2.30}$$

Thus the Frobenius norm is merely the sum over the square of all elements. For matrices the Frobenius norm simplifies to a square root of the trace of the squared matrix.

Mode-n product. The mode spaces of a tensor can be altered independently of the other mode spaces by a linear transformations called *mode-n product*. The mode-n product is defined between a matrix $\mathbf{M}$ and a tensor $\mathcal{A}$ with the result being another tensor $\mathcal{B}$. The product is written as $\mathcal{B} = \mathcal{A} \times_n \mathbf{M}$. The mode-$n$ product by the matrix $\mathbf{M}$ transforms all the vectors $\mathbf{v}$ in the n-th mode of the tensor into vectors $\mathbf{Mv}$. It is therefore possible to separately transform the vectors in mode-3 and the vectors in mode-2. The mode-n product can also be expressed in terms of flattened matrices as $\mathbf{B}_{(n)} = \mathbf{MA}_{(n)}$. We can therefore rewrite the singular value decomposition (SVD) of a matrix $\mathbf{D} = \mathbf{U\Sigma V^T}$ using the mode-n product as $\mathbf{D} = \mathbf{\Sigma} \times_1 \mathbf{U} \times_2 \mathbf{V}$.

The successive application of N mode-n products to a tensor can be represented using flattened tensors, matrix products and Kronecker products as follows

$$\mathcal{B} = \mathcal{A} \times_1 \mathbf{M}_1 \times_2 \mathbf{M}_2 \dots \times_n \mathbf{M}_1 n \dots \times_N \mathbf{M}_N \tag{2.31}$$

$$\mathbf{B}_{(n)} = \mathbf{M}_n \mathbf{A}_{(n)} (\mathbf{M}_{n-1} \otimes \mathbf{M}_{n-2} \otimes \dots \mathbf{M}_1 \otimes \mathbf{M}_N \otimes \dots \mathbf{M}_n + 1)^T \tag{2.32}$$

Equation 2.31 shows the application of N mode-n products. In equation 2.32 the equivalent formulation is given using matrices instead of tensors. Note that the transpose can be brought into the parantheses in the second term of 2.32, due to the transpose property of the Kronecker product 2.28. The order of the matrices inside of the parantheses is important. If this order is changed than the result has the same columns but in a different order.

High Order Singular Value Decomposition (HOSVD). The *N-Mode singular value decomposition* (or high order SVD) is a linear transformation of a tensor. An N-order tensor has N mode spaces. The high order SVD orthogonalizes the mode spaces by finding the N sets of orthonormal basis vectors which span the column space along the respective mode space. The HOSVD decomposition is as follows [17]

$$\mathcal{A} = \mathcal{S} \times_1 \mathbf{U}_1 \times_2 \mathbf{U}_2 \times_3 \mathbf{U}_3 \cdots \times_N \mathbf{U}_N \tag{2.33}$$

Equation 2.33 decomposes the tensor into a *core tensor* $\mathcal{S}$ and orthonormal mode matrices $\mathbf{U}_1 \dots \mathbf{U}_N$. The mode matrix $\mathbf{U}_i$ contains the orthonormal vectors $u_i j$ which form the orthogonal basis of the column space of the flattened tensor $\mathbf{A}_{(n)}$. The core tensor governs the interaction between these mode matrices and is as such analogous to a diagonal matrix of eigenvalues used in PCA. Eigenvalues can be seen as measures of the variance along the corresponding eigenvector directions. However, the core tensor does not have a diagonal structure. Rather it has the property of all-orthogonality whereby all subtensors of $\mathcal{S}$, obtained by fixing one index, are orthogonal. The Forbenius norms of these subtensors decrease, which makes it possible to reduce the dimensionality of the data set by truncating the core tensor. It is therefore possible to approximate the original tensor using a reduced core tensor. The approximation of a tensor $\mathcal{A}$ using a reduced core tensor $\mathcal{S}_{reduced}$ is derived from the mode-n SVD and is defined as:

$$\mathcal{A} \approx \mathcal{S}_{reduced} \times_1 \hat{\mathbf{U}}_1 \times_2 \hat{\mathbf{U}}_2 \times_3 \hat{\mathbf{U}}_3 \cdots \times_N \hat{\mathbf{U}}_N \tag{2.34}$$

The matrices $\hat{\mathbf{U}}_i$ are truncated versions of the eigenvector matrices for the corresponding mode space. Figure 2.6 depicts the approximate HOSVD using the truncated core tensor and mode matrices.

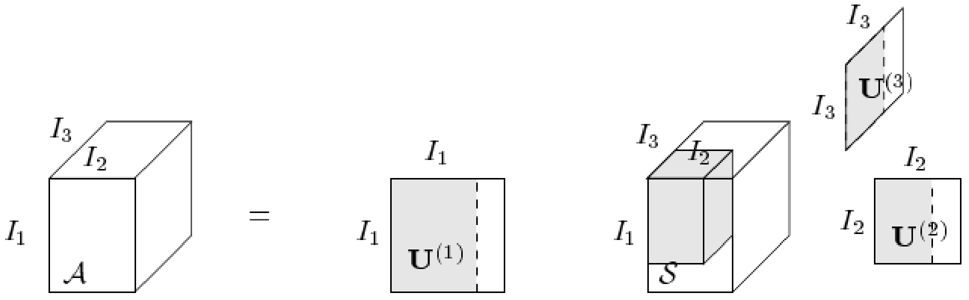


Figure 2.6: Multilinear face model showing how different attributes change along the modes. Taken from [17]

HOSVD Algorithm. The HOSVD decomposition of a tensor $\mathcal{A}$ of order N is computed as follows:

- **Step 1**. For $i = 1, \ldots, N$ compute the mode matrix $\mathbf{U}_i$ by calculating the SVD of the flattened matrix $\mathbf{D}_{(i)} = \mathbf{U_i \Sigma_i V_i^T}$. The left singular matrix of $\mathbf{D}_{(i)}$ is the mode matrix for mode i.

- **Step 2**. Compute the core tensor by reversing equation 2.33 to get the core tensor as
$$\mathcal{S} = \mathcal{A} \times_1 \mathbf{U}_1^T \times_2 \mathbf{U}_2^T \times_3 \mathbf{U}_3^T \cdots \times_N \mathbf{U}_N^T \tag{2.35}$$

2.1.3.2 Tensor-Based Model

The possibility of using a multilinear approach to construct face models was explored by numerous researchers. Vasilescu and Terzopoulos [31] build a tensor from a database of face images and construct *tensorfaces* by performing HOSVD on the data tensor. Tensorfaces are contained as vector the mode matrices $\mathbf{U}_i$. Vlasic, Brand, Pfister and Popovic [35] build a tensor model of 3D faces an utilize it to recognize faces in video sequences for the purpose of expression transfer. The group successfully managed to implemented a face transfer application based on a multilinear face model which allowed for expressions and even for speech related movements to be transferred between video-recordings of two different subjects. The multilinear model was estimated from geometric variations in 3D face scans that Vlasic and his group collected. Two tensor models with different dimensionality were utilized. Their first model was a lower dimensional bilinear model that organized the data into three groups of vertices, identity and expression. Their higher dimensional model organized faces into groups of expressions, vertices, identities and visemes. The parameters for the multilinear face model, which were used to synthesize new expressions, were extracted from the video input using an optical flow algorithm.

To construct a bilinear face model it is necessary to separate the faces from the database into groups of expressions and identities in order to organize them into the tensor. The tensor based approach allows us to organize the data in a way that makes for easier manipulation with a transformation matrix. The groups are aligned in the tensor along the modes as shown in figure 2.7. In this example the vertices change along the mode-1 space, the identity changes along the mode-2 space and the expressions change along the mode-3 space.

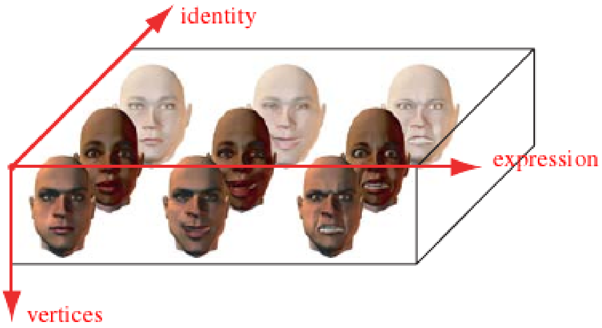


Figure 2.7: Multilinear face model showing how different attributes change along the modes. Taken from [35]

Vlasic et al constructed a multilinear face model using mode-n SVD and decomposing the organized tensor into mode matrices. A generative model can be constructed when the core tensor and the the mode-1 matrix are combined together using the mode-n product. The first mode space which holds the vertices, which the model should be capable of generating. Using equation 2.34 the data tensor $\mathcal{D}$ is decomposed as

$$\mathcal{D} \approx \mathcal{M} \times_2 \hat{\mathbf{U}}_2 \times_3 \hat{\mathbf{U}}_3 \cdots \times_N \hat{\mathbf{U}}_N \tag{2.36}$$

where tensor $\mathcal{M}$ is called the *multilinear model.* The multilinear model can then be used to synthesize new faces. Mode multiplying the multilinear model with one row from each mode matrix we recover exactly one of the original faces. Mode multiplying the model $\mathcal{M}$ with a linear combination of rows from the truncated mode matrices synthesizes a new face $\mathbf{f}$ giving us the generative tensor model as

$$\mathbf{f} = \mathcal{M} \times_2 \mathbf{w}_2^T \times_3 \mathbf{w}_3^T \cdots \times_N \mathbf{w}_N^T \tag{2.37}$$

The weight vectors $\mathbf{w}_2, \ldots, \mathbf{w}_N$ are column vectors which encode the attribute of the corresponding mode space.

The tensor-based model is similar to what we saw in the eigenfaces PCA approach in figure 2.1. However, the tensor-based approach has the advantage that we can manipulate the different sources of variance (i.e. in the case of a face model the identity, the expressions) separately. This advantage outweighs the additional computational complexity carried by the high order SVD and makes this model suitable for the task of expression transfer.

The drawback of the method described by Vlasic et al is that it requires the tensor to be fully populated. This means that we require all expressions to be performed by all subjects to have a full tensor. This raises issues when some data is corrupted or lost during data gathering. There are two ways of coping with this problem. The first approach is to use a statistical method to model the data. Alternatively the missing data can be estimated from the current data.

2.1.3.3 Tensor-based Statistical Discriminant Method

The tensor-based statistical discriminant methods (SDM) was successfully used by Minoi and Gillies [21, 20] to synthesize expressions and to neutralize faces displaying an expression. The SDM is based on Fischer's linear discriminant analysis (LDA) [14]. The LDA differs from the standard PCA eigenfaces approach in that it seeks to separate data into distinct classes. The way this is done is by projecting the data into a lower dimensional subspace that maximizes the between class separability and minimizes the within class variance.

In LDA we first separate the training data set into a number of groups. Then we look for the between class scatter matrix S_b and the within class scatter matrix S_w. The matrix Φ_{lda} that defines the projection onto the desired low dimensional space is defined as:

$$\Phi_{lda} = \arg\max_{\Phi} \frac{|\Phi^T S_b \Phi|}{|\Phi^T S_w \Phi|} \tag{2.38}$$

In equation 2.38 the ratio of the determinant of the between class separability and the determinant of the within class variability is maximized. The maximum of this equation is the optimal projection.

However, the LDA approach has difficulties when the training data set is small in comparison to the dimensions of the image. This is called the *small sample size problem* and it causes the scatter matrices to be singular when the number of data items is less than the number of variables. To overcome the small size problem the statistical discriminant method can be used.

The SDM approach consists of two stages. In the first stage the PCA and LDA are used to reduce the dimensionality and find the discriminant directions of the classes. The PCA helps to overcome the small size problem common to stand-alone LDA. In the second stage the most discriminant vectors of our classes are projected back into the original high dimensional space. This back projection will give us a vector for every class in the high dimensional space. These vectors allow us to synthesize an expression for a new face image by moving the the surface points of this new image in the direction of one of the vectors.

The SDM technique can be extended to tensor models by expanding the mode responsible for expressions into a number of modes representing individual expression classes, thereby effectively increasing the dimension of the tensor. The expanded tensor model still retains the quality of independence along the modes of variance. However, due to the expansion the SDM can be applied to the tensor if we flatten the sub-tensors into matrices.

2.2 3D Pose Estimation

The goal of 3D pose estimation is to determine the position and orientation of an object from a 2D image of the object. A special case of 3D pose estimation is the *Perspective-n-Point* (PnP) problem. This class of problems is specified as follows - given are the exact locations of a set of feature points on the object and the positions of points in the 2D image. The points in the 2D image correspond to the 3D feature points. The goal is to find how the object must be moved, rotated, scaled so that when the 3D feature points are projected using a camera model the result will approximate the input set of corresponding 2D points.

2.2.1 Camera Model

The camera model describes a *projective transformation.* This transformation maps a 3D point $\mathbf{X} = (X, Y, Z)^T$ to a 2D image point $\mathbf{x} = (x, y)^T$. The three most important camera models are the perspective model, the orthogonal model and the weak perspective model.

2.2.1.1 Perspective Camera Model

In perspective projection rays coming from points in the scene all enter the *center of projection.* These points are then "projected" onto an imaging surface, the *image plane.* The size of the image relative to the distant object is given by the parameter *focal length.* The focal length is the distance of the center of projection from the image plane. If the image plane is in behind the center of projection then the image of the object on the image plane is inverted. If the image plane is placed in front of the center of projection then the image and the object have the same orientation. The center of projection in essence corresponds to an ideal pinhole camera. The perspective projection in therefore known as the pinhole camera model.

The normal vector to the pinhole plane, which goes through the pinhole itself is called the *optical axis.* The optical axis is usually chosen either the negative or positive z axis and the pinhole is chosen to lie at the origin. The intersection of the optical axis and the image plane is known as the *principal point.* The configuration of the pinhole camera model is shown in figure 2.8.

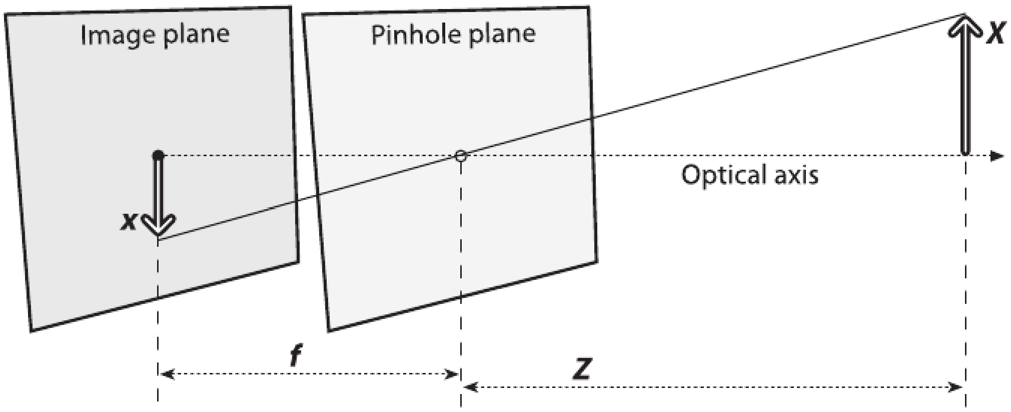


Figure 2.8: Perspective projection camera model with the image plane behind the camera

From figure 2.8, we can see that the image plane is located at $Z = f$ where f is the focal length. The coordinates of the projection of X can therefore be calculated using similar triangles to obtain $-x/f = X/Z$. If the image plane were to be place in front of the camera then the ratio given by the similar triangles will stay the same but the new coordinate will no longer be negative.

Hence, assuming that the principal point is given by $(c_x, c_y)^T$ and the focal length is f, then the projection $(x, y)^T$ of the point $\mathbf{X} = (X, Y, Z)^T$ is given by

$$\begin{pmatrix} x \\ y \end{pmatrix} = \frac{f}{Z} \begin{pmatrix} X \\ Y \end{pmatrix} + \begin{pmatrix} c_x \\ c_y \end{pmatrix} \tag{2.39}$$

The perspective projection is a non-linear transformation which cannot be written as a multiplication of a 3D points in Cartesian coordinates with a matrix. To obtain a matrix formulation, we must use homogeneous coordinates. The homogeneous coordinates of a point are $P_h = [p_x, p_y, p_z, s]^T$. The fourth coordinate component s is called *scale*. We transform a point in homogeneous coordinates into Cartesian coordinates by dividing the first three components by the scale to get the point $P_c = [p_x/s, p_y/s, p_z/s]^T$. Using homogeneous coordinates the pin hole camera model has the form

$$\gamma \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} f_x & 0 & c_x & 0 \\ 0 & f_y & c_y & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \tag{2.40}$$

where $\gamma = 1/s$ is responsible for converting the homogeneous result of the multiplication to Cartesian coordinates. In equation 2.40 two different focal lengths have been used. This is due to the fact that with any camera equipment the image plane will have dimensions in millimeters. The values f_x and f_y therefore incorporate the focal length as well as the conversion from units of the imaging equipment to pixels.

2.2.1.2 Orthogonal Camera Model

In the orthogonal camera model the image of an object is obtained by casting rays parallel to the optical axis from points (X, Y, Z) on the object and intersecting these rays with the image plane. The Z component is thereby dropped entirely. The projected point is thus given by $x = X$ and $y = Y$. The geometry of the orthogonal projection is shown in figure 2.9. Orthographic projection is sometimes called parallel projection, since unlike in perspective projection, parallel lines remain after being projected into the image plane.

2.2.1.3 Weak Perspective Camera Model

The weak perspective camera model is in essence just a scaled orthographic projection. With this camera model it is possible to approximate a perspective transformation under the following conditions.

- The object lies close to the optical axis.
- The object's depth (the difference between the maximum and minimum z value of the object) is small in comparison to the object's average distance from the center of projection.

Since the depths of the different points on the object are close to each other, we make the assumption all these depths are equal. Then a perspective projection of any object point (X, Y, Z) can be approximated as

$$\begin{pmatrix} x \\ y \end{pmatrix} = \frac{f}{Z_{avg}} \begin{pmatrix} X \\ Y \end{pmatrix} + \begin{pmatrix} c_x \\ c_y \end{pmatrix} \tag{2.41}$$

where Z_{avg} is the average distance of the object's point from the camera. Geometrically, we place a plane parallel to the image plane at a distance Z_{avg} from the center of projection. Then the object points orthogonally projected onto this new plane. Finally, we perspectively project these projected points onto the image plane. The geometry of the weak perspective projection is shown in figure 2.9.

Moreover, the weak perspective transformation is a linear transformation unlike the strong perspective transformation, since we are dividing by a constant Z_{avg} and not the Z component of the point.

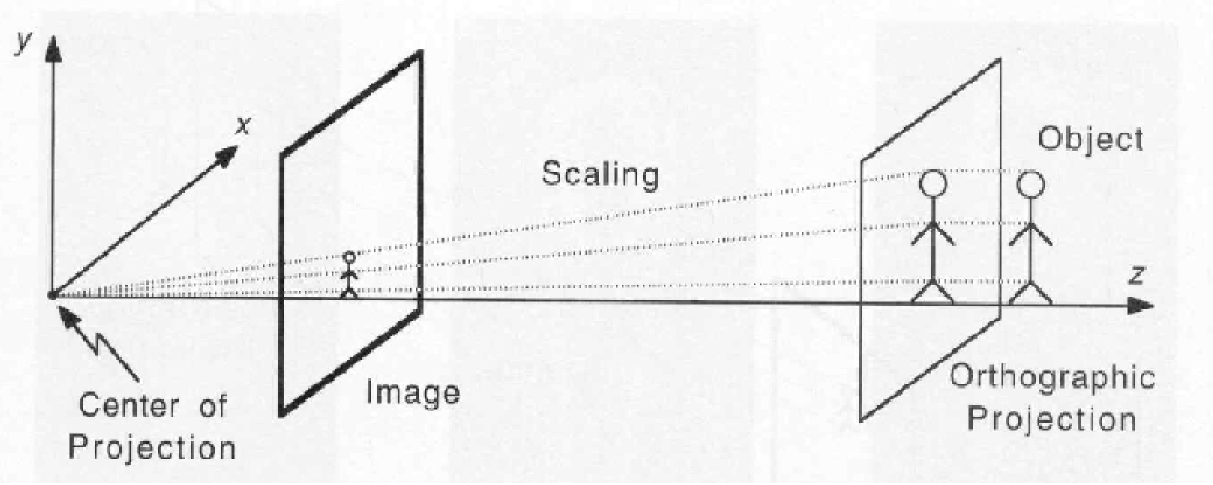


Figure 2.9: Weak perspective projection is a scaled orthographic projection. The image obtained by orthographic projection is scaled by the constant f/Z_{avg}.

2.2.2 Rotation and Translation

It is usual in most 3D rendering frameworks to define two coordinate systems. One coordinate system for the camera and one for the object. This makes it possible to apply transformations to objects independently since the points that make up each object are defined in each object's separate coordinate system. A transformation translates the points from the object coordinate system into the camera coordinate system. In case of n objects there are n independent object coordinate systems thereby making it possible to apply a different transformation to each of the n object coordinate systems and move the objects independently which is a crucial task for 3D rendering frameworks.

The transformations of the objects are based on the degrees of freedom for movement in 3D space. The two classes of transformations are rotation and translation. The origin of the object coordinate system is moved to the origin of the camera coordinate system during translation. Rotation aligns the viewing direction of the camera coordinate system with the viewing direction of the object coordinate system. The mapping from object coordinate space to camera coordinate space is shown in figure 2.10.

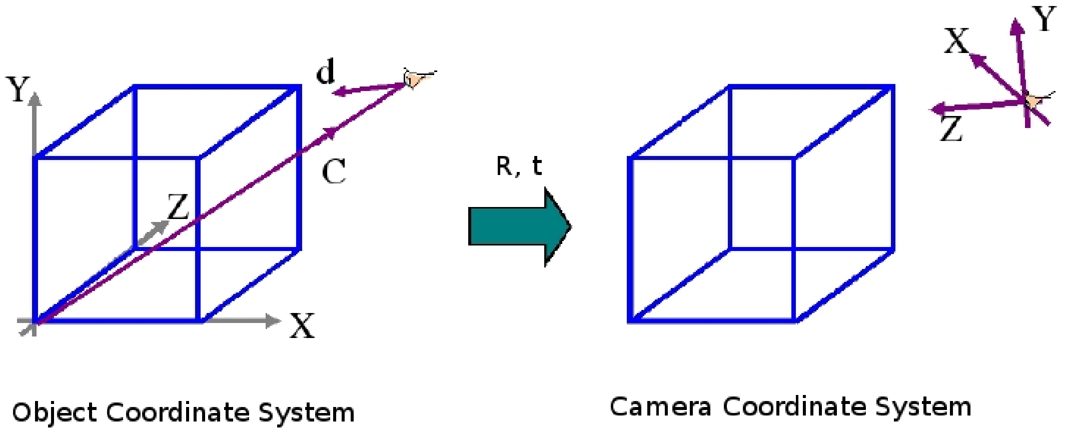


Figure 2.10: Mapping points from object coordinates to camera coordinates using rotation and translation. The displacement of the origin of the camera system from the origin of the object system is **C**. The camera is viewing in the direction of the vector d. Taken from [15]

The rotation of the object in n dimensions can described using *Givens rotations* [16]. A Givens rotation has the form

$$G(i,k,\theta) = \begin{array}{c} \\ \\ \\ i \\ \\ k \\ \\ \\ \end{array}\begin{array}{c}\begin{array}{ccccccc} & & i & & k & & \end{array}\\ \begin{pmatrix} 1 & \dots & 0 & \dots & 0 & \dots & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & \dots & c_\theta & \dots & s_\theta & \dots & 0 \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & \dots & -s_\theta & \dots & c_\theta & \dots & 0 \\ \vdots & & \vdots & & \vdots & \ddots & \vdots \\ 0 & \dots & 0 & \dots & 0 & \dots & 1 \end{pmatrix}\end{array} \tag{2.42}$$

where $c_\theta = \cos(\theta)$ and $s_\theta = \sin(\theta)$. Pre-multiplying with the Givens rotation $G(i,k,\theta)^T$ corresponds to a counterclockwise rotation by θ radians in the (i,k) coordinate plane, pre-multiplying with $G(i,k,\theta)$ rotates the the (i,k) plane by θ radians in clockwise direction.

From the Pythagorean theorem $\cos(\theta)^2 + \sin(\theta)^2 = 1$ and the symmetries of the sine and cosine function $\cos(-\theta) = \cos(\theta)$, $\sin(-\theta) = -\sin(\theta)$, it is obvious that Givens rotations are orthogonal.

$$G(i,k,\theta)^T G(i,k,\theta) = G(i,k,\theta) G(i,k,\theta)^T = \mathbf{I} \tag{2.43}$$

The transposed Givens rotation amounts to a rotation in the opposite direction by the same angle thus the two rotations cancel out.

To describe the rotation of the object in three dimensional space we use the following Givens rotations

$$R_x(\phi) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & c_\phi & s_\phi \\ 0 & -s_\phi & c_\phi \end{pmatrix} \quad R_y(\psi) = \begin{pmatrix} c_\psi & 0 & -s_\psi \\ 0 & 1 & 0 \\ s_\psi & 0 & c_\psi \end{pmatrix} \quad R_z(\theta) = \begin{pmatrix} c_\theta & s_\theta & 0 \\ -s_\theta & c_\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{2.44}$$

Note that the matrix $R_y(\psi)$ is in fact the transposed Givens rotation. This is due to the fact that we are interested in describing a left handed coordinate[1] system with the positive X axis pointing up, the positive Y axis pointing to the right and the Z axis pointing into the page. As such the rotation around the (X,Z) plane needs to be described by the Given a point $\mathbf{x} = (x,y,z)^T$ the pre-multiplication with the Givens rotation $R_z(\theta)$ produces a new point $\mathbf{x}' = (x',y',z')^T$ such that

$$R_z(\theta)\mathbf{x} = \begin{pmatrix} c_\theta & s_\theta & 0 \\ -s_\theta & c_\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} c_\theta x + s_\theta y \\ c_\theta y - s_\theta x \\ z \end{pmatrix} = \begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \mathbf{x}' \tag{2.45}$$

The expressions for the x' and y' component of the new point correspond to a rotation of the (X,Y) coordinate system by θ radians about the Z axis. These expressions are computed using simple trigonometrics where the point is decomposed into its x and y components which are then projected on the rotated (X',Y') axes as can be seen in figure 2.11.

[1] Some authors prefer to instead use the right handed coordinate system to describe an arbitrary 3D rotation.

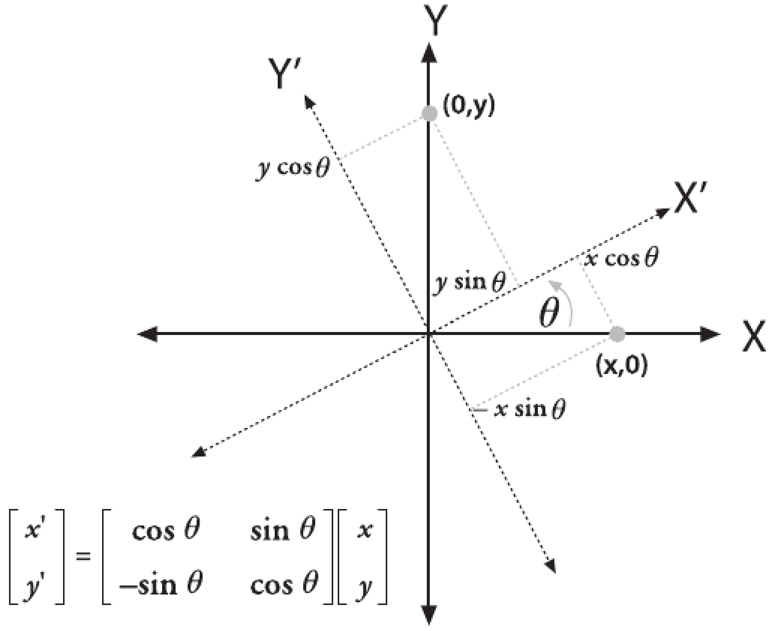

Y
Y′
(0,y)
y cos θ
X′
x cos θ
y sin θ
θ
X
(x,0)
−x sin θ
$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$

Figure 2.11: Transforming one point into another by rotating the coordinate system counterclockwise by θ radians. The point $[x, y]^T$ can be written as $[x, 0]^T + [0, y]^T$. These components are then individually transformed by projecting them onto the axes of the new coordinate system (X', Y') to obtain the new point $[x', y']^T$. The new point is the result of turning the original point clockwise by θ radians.

After the transformation in equation 2.45 the z component of the original point is left unchanged. From similar calculations it can be seen that with matrix R_y the y component, with R_z the z component remain unchanged. It is therefore apparent that the matrix $R_x(\phi)$ rotates by ϕ about the X axis, matrix $R_y(\psi)$ rotates by ψ about the Y axis and $R_z(\theta)$ rotates by θ about the Z axis. Any arbitrary rotation in 3D space can then be achieved using the total *rotation matrix* $R = R_x(\phi)R_y(\psi)R_z(\theta)$ with the appropriate angles. Since all the total rotation matrix is a product of three orthogonal Givens rotations it likewise orthogonal with $R^T R = RR^T = \mathbf{I}$, with $\mathbf{I}$ being the identity matrix. The three angles θ, ϕ and ψ represent the three rotational degrees of freedom (DOF) of the object.

Any position in three dimensional space can be achieved through a linear combination of movements in any of the three unit directions given by the X, Y and Z axes. The *translation vector* allows us to represent the shift from one coordinate system to another. In terms of the camera coordinate system and the object coordinate system, the translation vector is the offset between the origin of the camera system and the object system. Thus, to represent the position of the object we use the translation vector $t = origin_{object} - origin_{camera}$. The displacements in the the the unit directions $(1, 0, 0)^T$, $(0, 1, 0)^T$, $(0, 0, 1)^T$ are given by the components vector $t = (t_x, t_y, t_z)^T$.

Equation 2.46 shows how the rotation matrix and the translation vector allow us to transform a point $P = (p_x, p_y, p_z)^T$ from the object coordinate system to the camera coordinate system.

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = R \begin{pmatrix} p_x \\ p_y \\ p_z \end{pmatrix} + t \tag{2.46}$$

With homogeneous coordinates the 6 degrees of freedom (also known as the extrinsic parameters) can be used to construct a joint rotation-translation matrix which transforms the 3D point $Q = (p_x, p_y, p_z, 1)^T$ in the object coordinate system to a point $\mathbf{x} = (x, y, z, s)^T$ in the camera coordinate system as

$$\begin{pmatrix} x \\ y \\ z \\ s \end{pmatrix} = \begin{pmatrix} R & t \\ \mathbf{0} & 1 \end{pmatrix} \begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix} \tag{2.47}$$

where $\mathbf{0} = (0, 0, 0)$.

2.2.3 Pose estimation with the POSIT algorithm !NEW

Davis and DeMenthon's POSIT algorithm [11], [7] is a simple and efficient method for solving the PnP problem. The POSIT algorithm estimates the rotation matrix R and the translation vector t of an object, given a perspecitve camera with known parameters, at least 4 non-coplanar feature points on the 3D object and the corresponding projections of these points in the 2D image. To compute an approximation of the true R and t, we first expresses the camera coordinate system in terms of the object coordinate system. Since the rotation matrix aligns the axes of the two coordinate systems we therefore know that the rows i, j and k of the rotation matrix correspond to the unit vectors of the camera coordinate system expressed in the object coordinate system.

This is easy when on the example of the camera system unit vector $u = (1, 0, 0)^T$. The rotation matrix R transforms a vector x from the object coordinate system into u, therefore $u = Rx$. The rotation matrix is orthogonal so if we turn in the opposite direction we get $x = R^T u$ as the object system equivalent of u. Since $u = (1, 0, 0)^T$ we obtain the first row of the rotation matrix R as the object coordinate system vector which corresponds to u.

The rotation matrix can therefore be written as

$$R = \begin{pmatrix} i_1 & i_2 & i_3 \\ j_1 & j_2 & j_3 \\ k_1 & k_2 & k_3 \end{pmatrix} \tag{2.48}$$

Since the three vectors i, j and k are all orthogonal we only need to find the first two and obtain the third as the cross product of the other two.

The translation vector is specified as the vector $\overrightarrow{OM_0}$ between the origin of the camera system O and the origin of the object system M_0. The point M_0 can be conveniently picked as one off the known feature points. Unfortunately, the camera coordinate system equivalent of this point is unknown since we do not know the rotation matrix. However, the translation vector can be expressed in terms of the 2D projection of this feature point as $t = \frac{Z_0}{f}\overrightarrow{Om_0}$ where m_0 is the known 2D projection of M_0 and Z_0 is the z-component of the translation vector. Therefore, to estimate the 3D pose we need to find the values of i, j and Z_0.

The POSIT algorithm first makes a rough estimate these values using a *pose from orthography and scaling* (POS) step. In the POS step it is assumed that the points are far enough from the camera so that we can assume a weak-perspective camera model. Using this camera model the values i, j and Z_0 all have closed form solutions. After this first step, the 2D image are then projected back into the object coordinate space with using the true perspective camera and the pose (rotation, translation) estimated in the POS step. These new points are then used in further in POS steps until the algorithm converges, hence the name POS with iterations (POSIT). However, it is important to note that the convergence of the algorithm depends heavily on the weak-perspective assumption that the internal depth of the object is small compared to the distance from the pinhole camera.

The fundamental equations of the POSIT algorithm are

$$\overrightarrow{M_0M_i} \cdot I = x_i(1+\varepsilon_i) - x_0 \tag{2.49}$$

$$\overrightarrow{M_0M_i} \cdot J = y_i(1+\varepsilon_i) - y_0 \tag{2.50}$$

with

$$I = \frac{f}{Z_0}i, \quad J = \frac{f}{Z_0}j \quad \text{and} \quad \varepsilon_i = \frac{1}{Z_0}\overrightarrow{M_0M_i} \cdot k$$

During the POSIT iterations, values are given to ε_i starting with 0 in the first iteration and then computed from $\varepsilon_i = \frac{1}{Z_0}\overrightarrow{M_0M_i} \cdot k$ with the k and Z_0 being guesses from the previous iteration. Then the system of equations given by 2.49, 2.50 is solved to obtain the vectors I and J. These vectors are normalized to get the two rows of the rotation matrix i and j. The z-component of the translation vector Z_0 is given by the norm of either I or J.

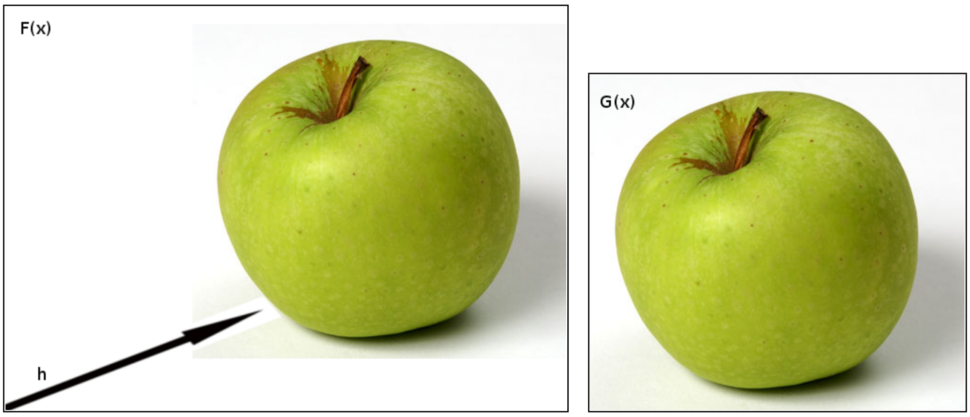
2.3 Feature Point Tracking

As we have seen, most model based recognition techinques rely on a number identified feature (landmark) points to estimate the model parameters. To be able to track a model between successive frames of a video recording, it is therefore necessary to either keep locating the points in every frame or to track the movement points themselves. This problem of tracking the movement points or objects between images has spurred the developement of many algorithms. Most of these algorithms fall into the *optical flow* category since they determine the movement of points based on the intensity changes at pixels. One such algorithm is the Kanade-Lucas-Tomasi feature tracker.

2.3.1 Kanade-Lucas-Tomasi Feature Tracker

Kanade and Lucas first introduced an iterative feature point registration algorithm in [18]. Tomasi and Kanade developed a more generalized version of algorithm in [29], [2]. The derivation of the algorithm is given by Birchfield in [6].

The problem of registering a displaced image can between two images is formalized by assuming there are two images functions $F(\mathbf{x})$ and $G(\mathbf{x})$. The image functions take a vector $\mathbf{x} = (x,y)^T$ as input and output the pixel intensity at $\mathbf{x}$. The goal in image registration is to find a disparity vector $\mathbf{h} = (h_x, h_y)^T$ which minimizes a measure of difference between $F(\mathbf{x}+\mathbf{h})$ and $G(\mathbf{x})$ for a given window of interest $\mathcal{W}$. The windows is required, since it is very difficult to track single pixels as they can be easily confused with other pixels unless they have very distinct brightness. The window of interest contains a number of pixels and should contain sufficient texture. The feature tracking is illustrated in figure 2.3.1 where the apple is the window of interest.



(a) F(x) (b) G(x)

Figure 2.12: The image registration problem. The task is to find the disparity vector h.

The optimal displacement vector $\mathbf{h}$ minimizes the error function E which is given by

$$E(\mathbf{h}) = \int_{\mathcal{W}} (F(x+h) - G(x))^2 \, d\,x \tag{2.51}$$

The naive way to locate this h would be to iterate through all the possible values of h and measure the $E(\mathbf{h})$ norm. A better approach was proposed by Kanade and Lucas. The behavior of the function $F(\mathbf{x})$ is approximated using a first order taylor expansion as

$$F(\mathbf{x}+\mathbf{h}) \approx F(\mathbf{x}) + \mathbf{g}^T\mathbf{h} \tag{2.52}$$

where

$$\mathbf{g} = \left[\tfrac{\partial}{\partial x}\left(\tfrac{F+G}{2}\right) \quad \tfrac{\partial}{\partial y}\left(\tfrac{F+G}{2}\right) \right]^T \tag{2.53}$$

This approximation is used instead of $F(\mathbf{x}+\mathbf{h})$ in the error. Then the minimum of the error function is located by setting the derivative of the error function with respect to $\mathbf{h}$ to zero. Therefore

$$\frac{\partial E}{\partial \mathbf{h}} = \frac{\partial}{\partial \mathbf{h}} \int_{\mathcal{W}} (F(x) + \mathbf{g}^T\mathbf{h} - G(x))^2 \mathrm{d}x = 2\int_{\mathcal{W}} \mathrm{g}(F(x) + \mathbf{g}^T\mathbf{h} - G(x))\mathrm{d}x \tag{2.54}$$

Setting the above to zero and rearranging the terms we get the fundamental equation which relates the spatial intensity to the optimal displacement $\mathbf{h}$ as

$$\left(\int_{\mathcal{W}} \mathbf{g}\mathbf{g}^T \mathrm{d}x\right)\mathbf{h} = \int_{\mathcal{W}} \mathrm{g}(F(x) - G(x))\mathrm{d}x \tag{2.55}$$

In matrix form the equation 2.55 can be expressed as

$$\mathbf{Z}\mathbf{h} = \mathbf{e} \tag{2.56}$$

The iterative Kanade-Lusca algorithm specifies the order in which possible values of $\mathbf{h}$ will be explored. The procedure iteratively improves the guess for $\mathbf{h}$ by solving the equation 2.55 with the spatial intensity gradients evaluated at each point $\mathbf{x}$ in the image. Thus, the algorithm locally searches for the best disparity vector $\mathbf{h}$ by using the information about the gradient at all points in the image. The estimation of the $\mathbf{h}$ and the convergence of the algorithm is further improved by weighing contributions of the points $\mathbf{x}$ inside the window. The weighing function can be a Gaussian or just set as $w(\mathbf{x}) = 1$.

The integrals simplify to sums when we are dealing with pixel values in real images. The iterative scheme for the Kanade-Lucas-Tomasi feature tracking algorithm is therefore:

$$\mathbf{h}_0 = 0 \tag{2.57}$$

$$\mathbf{h}_{k+1} = \mathbf{h}_k + \frac{\sum_{\mathbf{x}\in\mathcal{W}} w(\mathbf{x})\mathbf{g}[G(x) - F(\mathbf{x}+\mathbf{h}_k)]}{\sum_{\mathbf{x}\in\mathcal{W}} w(\mathbf{x})\mathbf{g}\mathbf{g}^T} \tag{2.58}$$

To further improve the technique it can be performed with several resolution levels, using the result from the coarser resolution as starting h_0 of the algorithm with a finer resolution. An effective variation of the algorithm which utilizes this approach was developed by Bouguet [6]. Bouguet's implementation of the Kanade-Lucas-Tomasi algorithm uses several resolution levels from highest to lowest. These resolutions resemble a pyramid of images, which is why this implementation is called the pyramid Lucas-Kanade tracker. At every level of the pyramid the iterations are computed to and the guess is propagated to the lower of the pyramid.

The algorithm can also successfully compute the h even if the region of interest has been rotated or scaled. If the image has been rotated or scaled we express the matching problem as $G(x) = F(xA + h)$ where A is a linear transformation matrix. This transformation allows us to apply the algorithm.

The Kanade-Lucas-Tomasi feature tracker has been successfully applied to tracking facial feature points [35]. Vlasic et al localized feature points in the initial frame manually and then used the Kanade-Lucas-Tomasi feature tracker to obtain the displacements between pairs of frames. From these displacements they were able to derive parameters for their tensor-based multilinear model. This allowed them to transfer expressions to another video performance.

As a local iterative search method, the Kanade-Lucas-Tomasi feature tracker's performance depends heavily on the initial guess of the h. If the initial guess is too far from the region of interest then the algorithm does not perform well, since the scheme was derived using local approximations of functions. These local approximations will not hold for points far from the region of interest.

2.4 Optimization

The task of locating the correct model parameters which match the model to an object in the image is an optimization problem. It may not be possible to find an exact match which would explain the object in the image perfectly. Therefore, we are interested in finding an instance of the model which is most similar to the object in the image. One way to measure disimilarity is through the use of an error function. If the value of the error function is large it means that the two entities are disimilar. To find a model instance which is most similar we therefore need to minimize the error function. Optimization techinques make it possible to locate minima of functions.

2.4.1 Least Squares and Non Negative Least Squares

An important error function, which is often utilized in statistics and engineering, is the sum of squares error. This error function defines the disimilarity between input values y_i and target values t_i as

$$E(y) = \sum_i (y_i - t_i)^2 \tag{2.59}$$

Solving a problem in the least squares sense means minimizing this error function. Given is a linear system of equations defined in matrix notation by $\mathbf{Ax} = \mathbf{b}$ with $\mathbf{A} \in \mathbb{R}^{m\times n}$, $\mathbf{x} \in \mathbb{R}^{n\times 1}$ and $\mathbf{b} \in \mathbb{R}^{m\times 1}$. This linear system may not have an exact solution if it is overdetermined $m > n$, underdetermined $m < n$ or more generally when $\mathbf{A}$ is not invertible. Solving this linear system in the least squares sense means finding a $\hat{\mathbf{x}}$ for which the following error function is minimized

$$E(\hat{\mathbf{x}}) = \frac{1}{2}\|\mathbf{b} - \mathbf{A}\hat{\mathbf{x}}\|^2 \tag{2.60}$$

with $\|.\|$ being the vector norm (distance). To find this minimum we take the derivation of 2.61 with respect to $\hat{\mathbf{x}}$ and set it to zero. The least squares solution is therefore

$$\hat{\mathbf{x}} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{b} \tag{2.61}$$

where the expression $(\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T$ is commonly known as the pseudo-inverse.

It is also possible to find the least squares solution of a linear system using singular value decomposition [23] if the matrix $\mathbf{A}$ is square. Usually the linear system would be solved by inverting $\mathbf{A}$ and pre-multiplying the equation with this inverse. Given the SVD $\mathbf{A} = \mathbf{U\Sigma V}^T$ we can easily invert $\mathbf{A}$ by inverting the right hand side to get $\mathbf{A}^{-1} = \mathbf{V\Sigma}^{-1}\mathbf{U}^T$. If the square $\mathbf{A}$ is not invertible it means that some of the diagonal elements in Σ are 0. However, We obtain a least squares solution by inverting the non-zero elements of Σ and pre-multiplying with $\mathbf{V\Sigma}^{-1}\mathbf{U}^T$.

In some case it may be necessary to enfore constraints on the function. This is particularly important when the variables represent real physical entities, which should not be negative. Such non negativity constraints are often introduced to a linear system. The goal of optimization is then to solve this non negativity linear system in the least squares sense. The problem is therefore to find the $\mathbf{x}^*$ for which holds

$$\mathbf{x}^* = \underset{\mathbf{x}\geq 0}{\operatorname{argmin}} \frac{1}{2}\|\mathbf{b} - \mathbf{Ax}\|^2 \tag{2.62}$$

To optimize this constrained error function Franc et al reformulate the problem as an equivalent quadratic optimization problem [13], [8].

$$\mathbf{x}^* = \underset{\mathbf{x}\geq 0}{\operatorname{argmin}} \Big(\frac{1}{2}\mathbf{x}^T\mathbf{Hx} + \mathbf{x}^T\mathbf{f}\Big) \tag{2.63}$$

where $\mathbf{H} = \mathbf{A}^T\mathbf{A}$ and $\mathbf{f} = -\mathbf{A}^T\mathbf{b}$. Franc et al solve this quadratic optimization problem by first formulating the Karush-Kuhn-Tucker (KKT) conditions and then applying a sequential coordinate descent algorithm. The KKT conditions for the non negative least squares solution are given by

$$\mathbf{Hx} + \mathbf{f} - \mu = \mathbf{0} \tag{2.64}$$

$$\mathbf{x} \geq \mathbf{0} \tag{2.65}$$

$$\mu \geq \mathbf{0} \tag{2.66}$$

$$\mathbf{x}^T \mu = 0 \tag{2.67}$$

The sequential coordinate-wise algorithm (SCA) then computes this solution so that the KKT conditions are satisfied. The SCA non-negativity least squares algorithm is given in 2.4.1. The input matrices are $\mathbf{H} = \mathbf{A}^T\mathbf{A}$, $\mathbf{f} = -\mathbf{A}^T\mathbf{b}$ and $\mathbf{h}_k$ denotes a column of the matrix $\mathbf{H}$.

Algorithm 1 Sequential Coordinate-wise Non-negativity Least Squares Algorithm

```
procedure SCANNLS($\mathbf{H}$,$\mathbf{f}$)
    Output $\mathbf{x}^* \geq 0$ such that $\mathbf{x}^* = \text{argmin}\frac{1}{2}\|\mathbf{Ax} - \mathbf{b}\|^2$
    Initialization $\mathbf{x}^0 = \mathbf{0}$ and $\mu^0 = \mathbf{f}$
    repeat
    for $k = 1 \dots n$ do
        $\mathbf{x}_k^{t+1} = \max\Big(0, \mathbf{x}_k^t - \frac{\mu_k^t}{\mathbf{H}_{k,k}}\Big)$
        $\mathbf{x}_i^{t+1} = \mathbf{x}_i^{t+1} \quad \forall i \neq k$
        $\mu^{t+1} = \mu^t + (\mathbf{x}_k^{t+1} - \mathbf{x}_k^t)\mathbf{h}_k$
    end for
    until stopping criteria (KKT conditions) are met.
end procedure
```

2.4.2 Nelder Mead Downhill Simplex

The downhill simplex is a local optimization method that does not require gradients to identify local extrema. With the downhill simplex method a local optimum is found by means of a sequence of fitness function evaluations.

The downhill simplex method was coined by Nelder and Mead in 1965 and has since proven itself to be comparable in efficiency to the more popular gradient based optimization methods. The method was designed for the optimization of multidimensional, unconstrained functions that have either no gradients or when the gradient exist only for portions of the search space such as in the case of discontinuous functions [26]. However, the drawback of the method is that many fitness function evaluations are required. Therefore, when the computational complexity of the fitness function is very high, other optimization methods need to be considered instead of the Nelder Mead downhill simplex [23].

The algorithm is based on the idea of isolating the minimum by geometrically transforming a *simplex*. The simplex is a convex hull of $N + 1$ vertices, where N is the underlying problem's dimension. In a two dimensional space this simplex would therefore be a triangle, in three dimensions a tetrahedron. The algorithm is initialized so that the simplex encloses a portion of the search space and the goal is to move the simplex along the search space surface and deform so that all of its vertices converge on the local optimum. This is achieved with the help of geometric transformations of the simplex.

The process of transforming a multidimensional simplex, in order to isolate the minimum, is somewhat analogous to bracketing a minimum in a one dimensional search space. The one dimensional search space will have peaks and valleys in places of local optima. The simplex, which is a line in one dimensional space, makes it's way downhill through this search space, in search of a minimum by means of shrinking and stretching. When the simplex finds a local minimum, it shrinks itself to contain only the minimum and the algorithm terminates.

The behavior of the simplex during the algorithm parallels the expanding and collapsing movements of the amoeba organism. The Nelder Mead downhill simplex is in some publications referred to as the *amoeba* method due to this similarity but also to distinguish it from Dantzig's simplex method for linear programming [23].

2.4.2.1 The Downhill Simplex Algorithm

To initialize the downhill simplex algorithm we need a nonlinear fitness function $f : \mathbb{R}^N \to \mathbb{R}$ and an initial point P_0. The simplex will be a $N + 1$ dimensional convex hull. The first vertex of the convex hull is the initial point. The remaining N vertices P_i are derived from the initial point. The shape of the simplex defines the way in which the points P_i are calculated[26].

The simplex shape can be one of the following:

- The simplex can have a regular shape where all sides are equally long. It is up to the user to pick this length.
- The simplex can be right angled in which case the vertices P_i are calculated according to formula 2.68.

  $$P_i = P_0 + \lambda e_i \tag{2.68}$$

  where e_i are unit vectors for the N dimensions and λ is a constant. This constant influences the size of the simplex and represents a guess of the problem's characteristic scale length [23].

After the initialization phase, three crucial steps are repeated until the simplex has encountered the local minimum. These steps are based around moving the vertex of the simplex with the largest fitness function largest value to a new point where the value will be smaller.

1. The first step is to sort the vertices x_i of the simplex from worst to best, where h is the index of the worst vertex, s the second worst index and l the best index.

2. Then the *centroid* of the best side is calculated according to formula 2.69.

$$c = \frac{1}{N} \sum_{j \neq h} x_j \tag{2.69}$$

3. In the final step, the centroid is used to geometrically transform the simplex in order to move the current worst vertex to a better position. To achieve this, the algorithm seeks a replacement point for x_h on the line that connects the worst index x_h and the centroid c. Three different points are then compared and the one with the best fitness value is chosen as the replacement point. These three candidates are obtained using reflection (formula 2.70), expansion (formula 2.71) and either inside or outside contraction (formula 2.72).

$$x_r = c + \alpha(c - x_h) \tag{2.70}$$

$$x_e = c + \gamma(x_r - c) \tag{2.71}$$

$$x_c = \begin{cases} c + \beta(x_r - c) & if\, x_h \leq x_r \\ c + \beta(x_h - c) & else \end{cases} \tag{2.72}$$

In case neither of these three new replacement point candidates has a better fitness value than the worst vertex x_h, then the entire simplex is shrunk towards the best vertex x_l. In this case N new vertices will be computed as follows:

$$x_j = x_l + \delta(x_j - x_l) \;\; j = 0, \ldots, N \wedge j \neq l \tag{2.73}$$

The geometric implications of the transformations reflection, expansion, contraction and shrinking are depicted in figure 2.13.

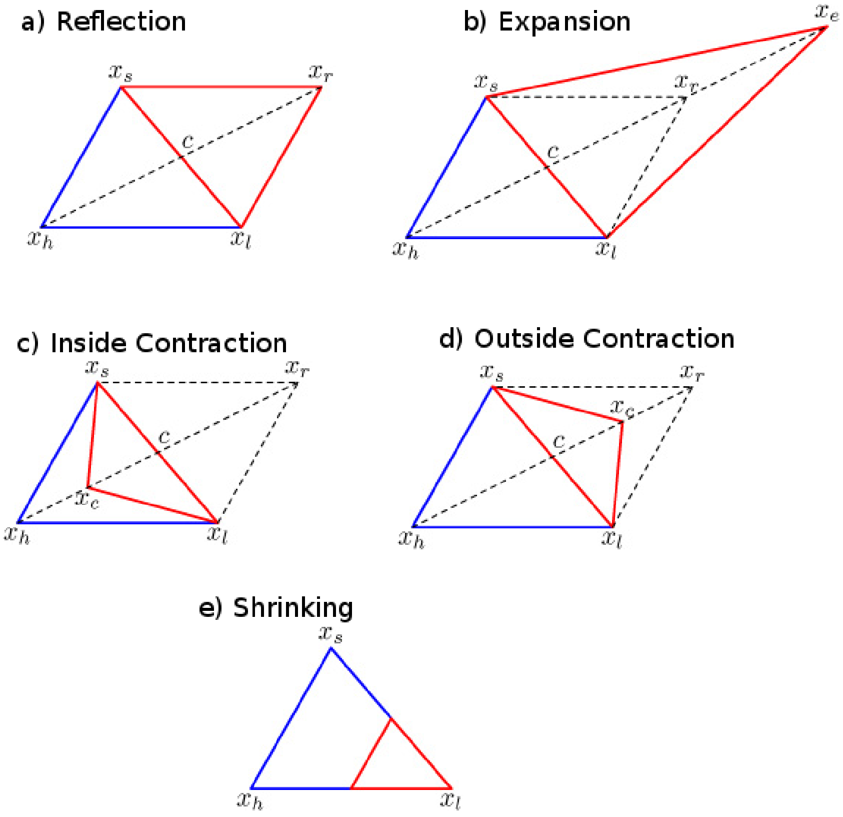


Figure 2.13: Geometric interpretations of the simplex transformations. Taken from [26]

The transformations are controlled by four parameters α for reflection, β for contraction, γ for expansion and δ for shrinking. Most implementations use the standard values $\alpha = 1$, $\beta = \frac{1}{2}$, $\gamma = 2$ and a shrinking by a half with $\delta = \frac{1}{2}$ [23, 26].

Finally, since the algorithm should terminate in finite time, it is necessary to establish a termination criterion. If the execution of the three aforementioned steps is considered one cycle of the algorithm, then possible termination criteria include terminating when the vector distance moved in the cycle was smaller than a constant tolerance *tol*, or when the difference between the fitness value of the newly obtained best and the old best is no larger than a tolerance $ftol$. Since either of these criteria could occur in a single anomalous step, restarts of the downhill algorithm are also sometimes utilized [23].

Chapter 3

Design and Implementation

In this chapter we will design an application which understands the facial dynamics captured in a video recording of an individual. This understanding is accomplished by correctly modelling the individuals face.

The practical purpose of the application is to transfer the expressions between two subjects. Given are video recordings of two persons and a database of faces. One recording is the source of the expressions, henceforth just source. The other recording is the target. The goal is to alter the target recording so that the target person's face is animated to perform the expressions seen in the source.

We approach this problem by constructing a deformable model of a face from the face database. The model is constructed so that every expression corresponds to a specific instance of the model parameters. The main task of the application is to manipulate the expression and identity of the model separately. This suggests that the optimal model should have separate parameters for these two attributes, to allow the user to manipulate the identity independently of the expression or vice versa. The tensor-based model is our deformable model of choice. As discussed in section 2.1.3.2, the tensor-based model possess the very important quality of separability.

Then the model is fitted frame-by-frame to the source recording using an algorithm that estimates the 3D pose of the model, the expression parameters and the identity parameters. The model is also fitted to the target recording to obtain a model of the target face. The parameters from the source model are then used to animate the target model.

The application therefore needs to be designed to support the following functionality:

1. Pre-processing of the face database.
2. Learning a tensor-based model from the face scans in pre-processed database.
3. Implementation of an algorithm to find the expression, identity and pose parameters of the face in the source and target recording.
4. Animating the target face with the expression parameters estimated from the source face.

The implementation of the this functionality needs to be encapsulated in graphical user interface (GUI). This GUI needs to be built with support for a 3D rendering framework since the application needs to display the three-dimensional models and to animate the target face. Additionally, most of the tasks that the application needs to solve either involve linear algebra computations or can be solved with standard computer vision algorithms. The implementations of these computations and algorithms need to be as efficient as possible to ensure that the application is highly responsive to the users actions and commands.

3.1 Design Challenges

While it may seem like a trivial problem to a human, the task of transferring expressions involves many challenges. The computer is attempting to understand a face using its model of what a face should look like, as well as recognize the face in images and follow its movements through successive frames. All the while, the application has to correctly interpret all the observed deformations as corresponding expressions.

The first challenge lies in developing an algorithm that fits a model to a face in the image. Two expressions can differ in only very little detail and yet this small amount of detail can make a huge difference to how a human observer interprets the expression. A slight difference in the position of the corner of the mouth is all that separates a smirk from a genuine smile. However, model based recognition revolves around approximating an object in the image with an instance of the model. So if it is to correctly identify expression, the algorithm needs to operate on a large number of points to minimize the error between the approximation and the observed face as much as possible. This first of all causes the algorithm to take more time, but it also directly leads to the second problem of how these points are to be chosen.

The model fitting is performed with an error minimization algorithm. To perform this minimization, reference measurements are needed against which the minimization is performed. These measurements will be points in the frames which comprise the video sequences. These points should of course be points located on the face. So before the algorithm can be begin fitting a model, it first needs some information about where the face is located in the image. However, it is not only necessary to identify the general position of the face – it is in fact imperative to know the position of points on the face which correspond to points in the model. Depending on how the face is located it may prove to be difficult to obtain more points to use in the fitting.

A convenient approach is to locate the face in only the first frame of the video sequence. Its position in the successive frames is then obtained by tracking the movement of the face from the first frame. Therefore, before applying the fitting algorithm the application needs to perform the following two steps:

1. Locating feature points in the first frame of both the source and target recording.

2. Tracking the movement of the feature points to obtain a guess for feature points in every frame.

Yet this tracking is again an approximation. It often also happens that certain points in the face are lost during the tracking procedure. So again it is impossible to guarantee that the fitting algorithm will be able to recover the sufficient level of detail it needs to correctly estimate the expression.

The final and most crucial problem lies in the fact that usually only very specific areas of the face contribute to expressions. The mouth or the brow are excellent areas for reference points for the fitting algorithm. However, should points these areas be lost during tracking, or misidentified during face location then it is very likely that an incorrect expression will be estimated. Even if only a small number of points is lost, the algorithm can still fail badly if these were points from an important part of the face.

It is clear that that error accumulates in every step of the model fitting. A robust application must therefore attempt to enforce constraint to limit this accumulation. The fitting algorithm and the model therefore need to be design with these considerations in mind.

3.2 Pre-processing

The primary drawback of the tensor-based model is that it requires the faces in the database to be in *correspondence*. To put faces in the database into correspondence so that a tensor model can be constructed we must perform pre-processing of the database data.

3.2.1 Correspondence

For the purpose of this application we consider two polygon meshes to be in correspondence if for any point in one polygon mesh the point in the other polygon mesh, which corresponds to the same position on the object, has the same index value. So for example if the point corresponding to the tip of the nose has an index value of i in one face, then a in the other face there will be a point with perhaps different x, y and z but the same i index that also corresponds to the tip of the nose. So essentially, the order of the indices must be the same in both polygon meshes which represent the faces.

In the case when the two polygon meshes are not in correspondence, then a optimization algorithm must be applied to deform the mesh to correspond to a reference mesh. There may be fewer vertices, which in turn requires generation of new vertices in the mesh. Or the vertices can be have a different ordering than in the reference mesh. To standardize the meshes the vertices in one meshes need to be aligned with vertices in the other mesh.

Correspondence can be computed in two ways. One approach is to use a registration algorithm where the rigid-transformations are located which transform a misaligned mesh into the reference mesh. An popular algorithm for 3D registration is the Iterated Closest Point (ICP) algorithm. Rusinkiewicz and Levoy give a comprehensive overview of efficient implementations of the ICP algorithm in [25]. The other approach involves finding the minimal deformation of the polygon mesh which transforms it into the reference mesh. Sumner and Popovic [28] describe an algorithm to transfer deformation between triangle meshes.

Other irregularities in the polygonal meshes can be present depending on how the 3D face scans in the database were obtained. For example there can be holes or spikes in the mesh, which need to be removed by interpolating new points to fill the holes or by Gaussian smoothing the spikes.

3.2.2 Database

The face database used in the application is the Binghamton database of 3D scans of expressions [36]. This database of 3D face scans was developed at the Department of

Computer Science, State University of New York at Binghamton. The subjects in the face database perform seven expressions – neutral, angry, disgust, fear, happy, sad and surprise. Every subject carried out each of these expressions (except for neutral) in four levels of intensity.

The Binghamton database was constructed with the help of 100 students who participated in the face scans. Various ethnic groups are represented and the database consists of about 60% female and 40% male subjects. There is an unfortunate lack of middle aged to older subjects in the database, due to the fact that mostly students were involved in face scans. For purposes of the face transfer application this is a clear disadvantage since it makes the model we construct much less powerful.

The advantage of a 3D face database lies in the fact that it allows us to construct a 3D model. This 3D model will clearly be more powerful then a 2D model since it has more degrees of freedom. When the task is to transfer expressions between two recordings of 3D faces it is preferable to have a 3D model which should in theory be able to duplicate the exact motions and rotations of the recorded objects. If the face database consisted of only 2D face images then the model would also have to be two-dimensional.

The face scans were captured using the 3DMD digitizer which utilizes structured light to calculate the depth. The captured surfaces contained between 20,000 and 35,000 polygons. Example original face scans, along with the captured texture maps are shown in figure 3.1

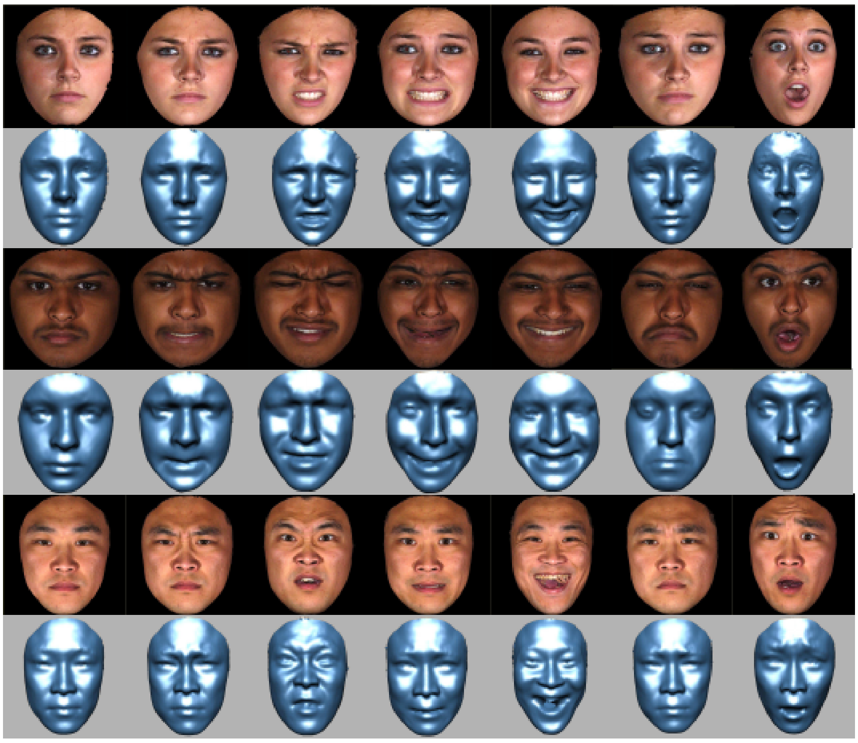


Figure 3.1: Sample 3D face scans from the Binghamton database along with the corresponding texture maps. Taken from [36]

The Binghamton database consists of faces stored as polygon mesh data in VMRL format and texture information in bit map format. For the project the pre-processed Binghamton database was used. This pre-processed database was created by Dr. Lynn-Minoi [20] and consists of VTK files of the pre-processed polygon meshes. The VTK file lists the x, y and z values of every vertex, the topology data of the face scan and the texture coordinates for every vertex. The face scans in pre-processed database are in correspondence, meaning that the order of the vertices in every face scan is the same. Conveniently, having the face scans in correspondence also means that the topology data stays constant.

The final pre-processing step before building a model is to move the center the face scans to the origin of the object coordinate system. This step is not always required. However, the vertices in the pre-processed Binghamton database have a z-component with an average value of $z_{avg} = -1500$. It serves no purpose to have the face at such a distance. It can even cause problems to some algorithms like the POSIT algorithm discussed in 2.2.3. Centering the face scans at the origin of the object coordinate system can be done by merely calculating the average x, y and z of the entire dataset and then subtracting this average from every vertex.

3.3 Model Construction

Given a database of 3D face scans that are in correspondence, the tensor-based model can be constructed by first organizing this data in a tensor and then using multilinear algebra to find a decomposition of this tensor.

3.3.1 Data Tensor

The tensor model should be designed to address expression and identity as sources of variation in the data. We assume that the points are defined by the set I_{pts}, the expressions by the set I_{exp} and the identities by I_{id}. Then we can construct a data tensor $\mathcal{D} \in \mathbb{R}^{I_{pts} \times I_{id} \times I_{exp}}$. Each face scan in the database is vector from $R^{I_{pts}}$. There are $I_{id} I_{exp}$ such vectors in the database. To load data into the the tensor we need to read each vector representing a face into the tensor and index the elements of this vector appropriately. This is done as follows

Algorithm 2 Loading the Data Tensor

```
procedure LOAD(database,$\mathcal{D}$)    ▷ Input the database and the empty data tensor
    for $i \in I_{id}$ do
        for $j \in I_{exp}$ do
            $v \leftarrow$ database$(i, j)$
            for $k \in I_{pts}$ do
                $\mathcal{D}(i, j, k) \leftarrow v_i$
            end for
        end for
    end for
    return $\mathcal{D}$                   ▷ return the initialized data tensor
end procedure
```

Due to the fact that most algorithms operate on matrices instead of tensors, it may be preferable to flatten the data tensor right after loading it. Flattening the tensor along a mode is done by successively fixing the values of the index of the corresponding mode and iterating through the values of the other indices to obtain the data. The procedure for flattening a tensor along the identity mode space is given in **??**.

Algorithm 3 Flattening a Tensor along the identity mode space

```
procedure FLATTEN($\mathcal{D}$, $\mathbf{D_{(2)}}$)        ▷ The input tensor, the output matrix
    for $i = 1 \dots I_{id}$ do
        for $j1 \dots I_{exp}$ do
            for $k1 \dots I_{pts}$ do
                index ← $j * I_{pts} + k$
                $\mathbf{D_{(2)}}(i,\text{index}) \leftarrow \mathcal{D}(i, j, k)$
            end for
        end for
    end for
    return $\mathbf{D_{(2)}}$        ▷ return the flattened tensor
end procedure
```

The data tensor itself will need to be loaded once and flattened three times for the computation of the HOSVD decomposition.

3.3.2 Computing the HOSVD !! NEW

The generative tensor model for the face transfer application is given by the formula

$$\mathbf{f} = \mathcal{M} \times_2 \mathbf{w}_i^T \times_3 \mathbf{w}_e^T \tag{3.1}$$

where $\mathbf{f}$ is the generated 3D face scans in form of a vector and $\mathbf{w}_i$ with $\mathbf{w}_e$ are column vectors of identity and expression parameters respectively. Ideally, we should be able to generate new faces using equation 3.1 by choosing values for the identity and expression parameters and obtaining a 3D face scan in form of the $\mathbf{f}$ vector. To do this it is necessary to compute the multilinear model $\mathcal{M}$ tensor which governs this mapping between the vector spaces.

As discussed in equation 2.35 the core tensor is obtained by reversing the HOSVD decomposition. The multilinear model tensor $\mathcal{M}$ is obtained by the mode product of the core tensor and the mode matrix of the points. This is equivalent to not factoring along the point mode space during the HOSVD. To construct the model we therefore have to implement the following formula

$$\mathcal{M} = \mathcal{C} \times_1 \mathbf{U}_{pts}^T = \mathcal{D} \times_2 \mathbf{U}_{id}^T \times_3 \mathbf{U}_{exp}^T \tag{3.2}$$

which relates the data tensor $\mathcal{D}$ with the mode matrices $\mathbf{U}_{id}$ and $\mathbf{U}_{exp}$. The column vectors of $\mathbf{U}_{id}$ span the space of identities and its row vectors encode an identity and point invariant representation of each identity seen in the database. Similarly, the column vectors of $\mathbf{U}_{exp}$ form the basis of the space of expressions and the rows are invariant representations of the various expressions. These matrices are obtained through singular value decomposition of the flattened matrices as

$$\mathbf{D}_{(2)} = \mathbf{U}_{id}\mathbf{\Sigma}_{id}\mathbf{V}_{id}^T \tag{3.3}$$

$$\mathbf{D}_{(3)} = \mathbf{U}_{exp}\mathbf{\Sigma}_{exp}\mathbf{V}_{exp}^T \tag{3.4}$$

However, computing the mode matrices using the above equations is often not feasible due to computer memory constraints. This can be easily seen from the magnitude of the matrices involved in the computation. Consider the dimensions of the flattened matrices seen in 3.3 and 3.4. The data tensor we described in section 3.3.1 has dimensions $I_{pts} \times I_{id} \times I_{exp}$. The flattened matrix $\mathbf{D}_{(2)}$ is flattened along the identity mode and has dimensions $I_{id} \times I_{exp} I_{pts}$. For the SVD of this matrix we therefore need three matrices $\mathbf{U}_{id}$, $\mathbf{\Sigma}_{id}$ and $\mathbf{V}_{id}$ of dimensions $I_{id} \times I_{id}$, $I_{id} \times I_{exp} I_{pts}$ and $I_{exp} I_{pts} \times I_{exp} I_{pts}$ respectively. Flattening along the expression mode produces the $\mathbf{D}_{(3)}$ matrix with dimensions $I_{exp} \times I_{pts} I_{id}$. The matrices obtained from the decomposition of this matrix are $\mathbf{U}_{exp}$, $\mathbf{\Sigma}_{exp}$ and $\mathbf{V}_{exp}$ with dimensions $I_{exp} \times I_{exp}$, $I_{exp} \times I_{pts} I_{id}$ and $I_{pts} I_{id} \times I_{pts} I_{id}$.

The memory requirements are best illustrated on an actual application. For instance, consider an application with 56 identities, 7 expressions and 5090 vertices, the data tensor has three dimensions and a size $56 \times 7 \times (5090 * 3)$. A matrix flattened along the expression mode space then has two dimensions and a size $7 \times (56 * 5090 * 3)$. Decomposing this flattened matrix into $\mathbf{U}_{exp} \mathbf{\Sigma}_{exp} \mathbf{V}_{exp}^T$ requires the allocation of memory for matrices of sizes 7×7 for $\mathbf{U}_{exp}$, 7 non-zero singular values in $\mathbf{\Sigma}_{exp}$ and a matrix of size $(56 * 5090 * 3) \times (56 * 5090 * 3)$ for $\mathbf{V}_{exp}$. This means that $\mathbf{V}_{exp}$ matrix needs to be represented using $855120 * 855120$ float values. The memory requirement of this is approximately $4 * 700 * 10^9$ bytes. Clearly the $\mathbf{V}_{exp}$ matrix is too large and exceeds the memory capacity of most personal computers.

We are not interested in the matrix $\mathbf{V}$ so ideally we should try to avoid computing it and thereby avoid needing to allocate memory for it. Note that to compute a simplified and less memory demanding version we can post-multiply the matrix with its transpose. To illustrate this approach lets take a matrix $\mathbf{A}$ with dimension $m \times n$. The SVD of this matrix is $\mathbf{A} = \mathbf{U \Sigma V}^T$, where $\mathbf{U}$ is a $m \times m$ orthonromal matrix, $\mathbf{\Sigma}$ a $m \times n$ diagonal matrix and $\mathbf{V}$ is a $n \times n$ orthonormal matrix. If the goal is to obtain the left singular matrix U then finding this matrix by means of SVD of $\mathbf{A}$ will be memory inefficient if $m \ll n$. However, note that

$$\mathbf{A}^T = (\mathbf{U \Sigma V}^T)^T = (\mathbf{V}^T)^T (\mathbf{U \Sigma})^T = \mathbf{V \Sigma}^T \mathbf{U}^T = \mathbf{V \Sigma U}^T \tag{3.5}$$

Therefore if we post-multiply $\mathbf{A}$ with $\mathbf{A}^T$ we obtain

$$\mathbf{A A}^T = \mathbf{U \Sigma V}^T \mathbf{V \Sigma U}^T \tag{3.6}$$

Because $\mathbf{V}$ is orthonormal this means that $\mathbf{V}^T \mathbf{V} = \mathbf{I}$, where $\mathbf{I}$ is the identity matrix. Using this relationship equation 3.6 simplifies to

$$\mathbf{A A}^T = \mathbf{U \Sigma}^2 \mathbf{U}^T \tag{3.7}$$

Since the square of a diagonal matrix is again a diagonal matrix. Therefore $\mathbf{\Sigma}^2$ is diagonal, which means that we can calculate the left singular matrix $\mathbf{U}$ in a memory efficient way by computing the SVD of $\mathbf{AA}^T$. The approach defined by equation 3.7 is crucial for the face transfer application in the first step of the HOSVD algorithm.

Once the mode matrices $\mathbf{U}_{id}$ and $\mathbf{U}_{exp}$ have been computed, the next step of the HOSVD algorithm is to calculate the multilinear model using equation 3.2. This can be done by utilizing the relationship between the Kronecker product and mode-n multiplication. Starting from the HOSVD decomposition of the data tensor which is given by

$$\mathcal{D} = \underbrace{\mathcal{S} \times_1 \mathbf{U}_{pts}}_{\mathcal{M}} \times_2 \mathbf{U}_{id} \times_3 \mathbf{U}_{exp} \tag{3.8}$$

where $\mathcal{S}$ is the core tensor and $\mathcal{M}$ is the multilinear model we are interested in computing. The HOSVD decomposition can then be rewritten using Kronecker products by flattening the core tensor and the data tensor along the first mode. The

$$\mathbf{D}_{(1)} = \underbrace{\mathbf{U}_{pts}\mathbf{S}_{(1)}}_{\mathbf{M}_{(1)}}(\mathbf{U}_{id} \otimes \mathbf{U}_{exp})^T \tag{3.9}$$

As seen in **??**, the Kronecker product of orthogonal matricies is again orthogonal. Therefore equation 3.9 can be post-multiplied with with $\mathbf{U}_{id} \otimes \mathbf{U}_{exp}$ to give

$$\mathbf{D}_{(1)}(\mathbf{U}_{id} \otimes \mathbf{U}_{exp}) = \mathbf{M}_{(1)}(\mathbf{U}_{id} \otimes \mathbf{U}_{exp})^T(\mathbf{U}_{id} \otimes \mathbf{U}_{exp}) \tag{3.10}$$

$$\mathbf{D}_{(1)}(\mathbf{U}_{id} \otimes \mathbf{U}_{exp}) = \mathbf{M}_{(1)} \tag{3.11}$$

$$\mathbf{M}_{(1)} = \mathbf{D}_{(1)}(\mathbf{U}_{id} \otimes \mathbf{U}_{exp}) \tag{3.12}$$

Equation 3.12 is essentially equation 3.2 rewritten using Kronecker products. It is not necessary to compute the multilinear model tensor $\mathcal{M}$ from the flattened multilinear model matrix $\mathbf{M}_{(1)}$. This is due to the fact that the flattened multilinear model is obtained from the decomposition of the flattened data tensor $\mathbf{D}_{(1)}$ as seen in equation 3.9. This flattened data tensor is composed of the 3D face scans arranged as the columns of the matrix. The goal of the model is to be able to synthesize face. These new faces should be generated as linear combinations of the original faces. This implies that generating new faces can be done by linear combination of columns of the flattened matrix $\mathbf{D}_{(1)}$. Since the flattened multilinear model $\mathbf{M}_{(1)}$ is part of the decomposition of $\mathbf{D}_{(1)}$, it can therefore be used to perform this linear combination.

The two steps described can now be combined into the HOSVD algorithm for the face transfer application. The steps of the algorithm are as follows:

- **Step 1.** Flatten the data tensor along all modes to obtain the matrices $\mathbf{D}_{(1)}$, $\mathbf{D}_{(2)}$ and $\mathbf{D}_{(3)}$.
- **Step 2.** Compute the identity mode matrix $\mathbf{U}_{id}$ using SVD as

$$\mathbf{D}_{(2)}\mathbf{D}_{(2)}^T = \mathbf{U}_{id}\mathbf{\Sigma}_{id}^2\mathbf{U}_{id}^T \tag{3.13}$$

- **Step 3.** Compute the expression mode matrix $\mathbf{U}_{exp}$ using SVD as

$$\mathbf{D}_{(3)}\mathbf{D}_{(3)}^T = \mathbf{U}_{exp}\mathbf{\Sigma}_{exp}^2\mathbf{U}_{exp}^T \tag{3.14}$$

- **Step 4.** Finally, calculate the multilinear model flattened along the first mode using the Kronecker product as

$$\mathbf{M}_{(1)} = \mathbf{D}_{(1)}(\mathbf{U}_{id} \otimes \mathbf{U}_{exp}) \tag{3.15}$$

3.3.3 Generating new faces

As previously mentioned, the general idea that motivates the use of statistical analysis techniques when building models, is that novel faces can be obtained from linear combinations of the faces in the 3D face scan database. This new face will look realistic given that the new face is close in terms of standard deviation to the mean. Therefore, one approach to generating new faces from the data tensor would be to flatten the tensor along the points mode to obtain $\mathbf{D}_{(1)}$ and synthesize new faces using a linear combination of the columns of this matrix. A face model would therefore be by choosing a set of parameters $\mathbf{w}$ and generating a new face $\mathbf{f}$ as

$$\mathbf{f} = \mathbf{D}_{(1)}\mathbf{w} \tag{3.16}$$

The HOSVD algorithm calculates the multilinear model $\mathbf{M}_{(1)}$ along with the mode matrices $\mathbf{U}_{id}$ and $\mathbf{U}_{exp}$. These matrices form the decomposition of the data matrix $\mathbf{D}_{(1)}$ as per equation 3.9. Thus, the generative model becomes

$$\mathbf{f} = \mathbf{M}_{(1)}(\mathbf{U}_{id} \otimes \mathbf{U}_{exp})^T\mathbf{w} \tag{3.17}$$

Since we are free to choose the coefficients $\mathbf{w}$, they can be thought of as composed through a Kronecker product as $\mathbf{w} = \mathbf{w}_{id} \otimes \mathbf{w}_{exp}$. The transpose of a Kronecker product is equal to transposing the operands of the product. Therefore

$$\begin{aligned}\mathbf{f} &= \mathbf{M}_{(1)}(\mathbf{U}_{id} \otimes \mathbf{U}_{exp})^T(\mathbf{w}_{id} \otimes \mathbf{w}_{exp}) && (3.18)\\ &= \mathbf{M}_{(1)}(\mathbf{U}_{id}^T \otimes \mathbf{U}_{exp}^T)(\mathbf{w}_{id} \otimes \mathbf{w}_{exp}) && (3.19)\\ &= \mathbf{M}_{(1)}(\mathbf{U}_{id}^T\mathbf{w}_{id} \otimes \mathbf{U}_{exp}^T\mathbf{w}_{exp}) && (3.20)\end{aligned}$$

where we have used the relationship between multiplication and the Kronecker product shown in 2.27. So to recover the i-th of the original faces from the multilinear model, the parameter vector $\mathbf{w}$ will have a 1 at the i-th position. When this parameter vector is multiplied with the flattened data tensor, the i-th column will be recovered. This i-th column is the i-th face scan in the database.

Use of the multilinear model allows more powerful recovery. For example to obtain the face which corresponds to the j-th identity and k-th expression. The parameter vector is constructed from the identity and expression vectors. The identity parameter vector will have a 1 at the j-th position and the expression parameter vector will have a 1 at the k-position. Taking the Kronecker product of these two will produce the parameter vector $\mathbf{w}$ with a 1 at the $(j * I_{exp} + k)$-th position which reconstructs the face with the j-th identity and the k-th expression when the parameter vector is multiplied with the flattened data tensor as per equation 3.16.

Furhermore, any arbitrary interpolation, or indeed extrapolation of the faces in the database can be obtained from the multilinear model based on the choice of the attribute parameters. This is because any parameter vector can be written as a sum of vectors which all have zeros at all positions except one. If we define $\mathbf{e}_i$ to be a vector which has a 1 at the i-th position and a zero everywhere else then $\mathbf{w} = a_1\mathbf{e}_1 + a_2\mathbf{e}_2 + \ldots + a_n\mathbf{e}_n$ where the a_i are arbitrary coefficients. Therefore to show that the model produces any arbitrary linear combination of the original faces we proceed as

$$\begin{aligned}\mathbf{M}_{(1)}(\mathbf{U}_{id} \otimes \mathbf{U}_{exp})^T\mathbf{w} &= \mathbf{M}_{(1)}(\mathbf{U}_{id} \otimes \mathbf{U}_{exp})^T(a_1\mathbf{e}_1 + a_2\mathbf{e}_2 + \ldots + a_n\mathbf{e}_n) && (3.21)\\ &= a_1\mathbf{D}_{(1)}\mathbf{e}_1 + a_2\mathbf{D}_{(1)}\mathbf{e}_2 + \ldots + a_n\mathbf{D}_{(1)}\mathbf{e}_n && (3.22)\end{aligned}$$

This is due to the fact that the multiplication of $\mathbf{D}_{(1)}\mathbf{e}_i$ recovers the i-th face. Therefore any linear combination can be generated using the multilinear model.

If the goal is to interpolate a face from the original faces, then the coefficients a_i must all be greater or equal to 0 and they must all sum to 1. This interpolation can equivalently be done by choosing the $\mathbf{w}_{id}$ and $\mathbf{w}_{exp}$ to have all components greater or equal to 0 and both weight vectors summing to 1. The Kronecker product of two such vectors again has components all greater or equal to 0 and summing to 1. If the Kronecker product is the vector $\mathbf{w} = \mathbf{w}_{id} \otimes \mathbf{w}_{exp}$, then the elements of this vector are all possible combinations of $\mathbf{w}_{id_i}\mathbf{w}_{exp_j}$ which are the i-th and j-th components of the identity and expression vectors. The sum of the elements of $\mathbf{w}$ is therefore

$$\sum_i \Big(\mathbf{w}_{id_i} \sum_j \mathbf{w}_{exp_i} \Big) = \sum_i \mathbf{w}_{id_i} = 1$$

All the elements of the vector $\mathbf{w}$ are also trivially greater or equal to 0 as well.

If we are not interested in interpolating between the original faces, then it is not necessary to store the mode matrices $\mathbf{U}_{id}$ and $\mathbf{U}_{exp}$. These mode matrices are invertible and thus form a basis for the subspaces $\mathbb{R}^{I_{id}}$ and $\mathbb{R}^{I_{exp}}$ respectively. Thus, the operands $\mathbf{U}_{id}^T\mathbf{w}_{id}$ and $\mathbf{U}_{exp}^T\mathbf{w}_{exp}$ of the Kronecker in equation 3.20 can generated any vector from these subspaces. Since any vector from these subspaces can be obtained inside the Kronecker product, we can safely drop the mode matrices to obtain the generative model.

$$\mathbf{f} = \mathbf{M}_{(1)}(\mathbf{w}_{id} \otimes \mathbf{w}_{exp}) \tag{3.23}$$

The matrices computed through HOSVD thus make it possible to synthesize a novel face by choosing the identity and expression model parameters $\mathbf{w}_{id}$ and $\mathbf{w}_{exp}$. This multilinear models allows the recovery of faces based on either of the identity and expression attributes. The expressive power of this model depends solely on the variance of the faces in the database from which the new faces are generated as linear combinations.

3.4 Fitting Algorithm

Once the face model is constructed, the objective becomes to correctly adjust the parameters that control this model in order to match a face in an image. The fitting algorithm must also allow the face in the image to be scaled, rotated or moved. So in addition to estimating the model parameters it needs to approximate the 3D pose of the object. As discussed in section 2.2 the 3D pose is defined by a rotation matrix $\mathbf{R}$ and a translation vector $\mathbf{t}$. There are 3 parameters which uniquely describe a rotation matrix and 3 components in a translation vector. The total number of model parameters depends on the amount of persons in the database and on how many expressions these individuals perform. In the model construction section we defined these numbers to be I_{id} and I_{exp} respectively. Thus, to fit the model to an image there are $6 + I_{id} + I_{exp}$ parameters which need to be estimated. To approximate these parameters the fitting algorithm will need to minimize the the difference between the face in the image and the model with the changed pose.

However, the model is three dimensional and the images are just two dimensional. Therefore, to be able to estimate the parameters in an optimization algorithm, the 3D points of the model must be projected to a 2D space using a camera model.

After the model has been projected, an error between the projection and the face in the image is calculated. To be able to calculate this error, it is necessary to know a set of 2D reference points in the image. These reference points, or feature points, correspond to known points on the model. The idea behind the fitting algorithm is to find how the model needs to be deformed and what 3D pose it needs to have so that these corresponding points on the 3D model are projected onto the feature points. The feature points are therefore crucial to determining the difference between the model and the face in the image. Minimizing this difference is the purpose of the fitting algorithm. Finding feature points is not a trivial task since they need to correspond to points on the model.

The goal of the face transfer application is to understand the facial dynamics of an individual from a video recording – so therefore a sequence of images. Since we are dealing with a video it means that the fitting algorithm will need to be performed for each frame of the video recording and to do that it will need reference points in each and every frame. The application therefore needs to track the movement of the feature points so that the fitting algorithm can be automatically applied to every frame.

3.4.1 Locating Feature Points

The first problem that needs to be addressed before the fitting algorithm is designed, is how the application locates feature points. These feature points need to correspond to points in the 3D model so that they can be used as reference points in the parameter optimization step. The feature points therefore cannot be picked randomly or even independently of the model.

One approach is to designate which points on the model will be used as correspondences to the feature points during the model construction step. In the case of the face transfer application these points should be chosen so that their positions can be used to determine expressions. These points can be the corners of the mouth, the tip of the nose and other easily distinguishable features on the face model. Given this predefined set of points on the model, the feature points which correspond to them are located in the image. This can be done either manuallly by a user who selects the locations of these feature points one by one. Reference points selection in the image could likewise be perfromed with an automatic feature detector such as the Viola-Jones object detection framework [34].

The manual approach is more flexible, simple and less error prone. The flexibility comes from the fact that adding a new feature point to the model does not necessitate the writing of a new feature detector. However, it can be tedious for the user to have to select a large number of points in the image and it is certainly not as impressive as when these points are located automatically.

The drawback of using automatic feature detectors is the fact that they introduce error. The feature locations they detect are guesses. One of the challenges that the expression transfer application faces is that estimating the correct expression requires alot of detailed information. Any amount of error is significantly counterproductive since it will inevitably increase during the feature tracking stage.

For the sake of flexibility and simplicity the expression transfer application is designed with a fixed set of feature points on the model. The user is then asked to provide the corresponding feature points on the first frame of the video. The default feature points on the model are shown in figure 3.2.

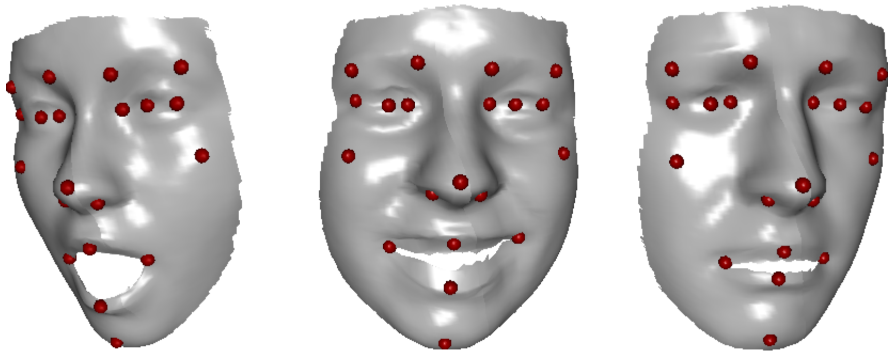
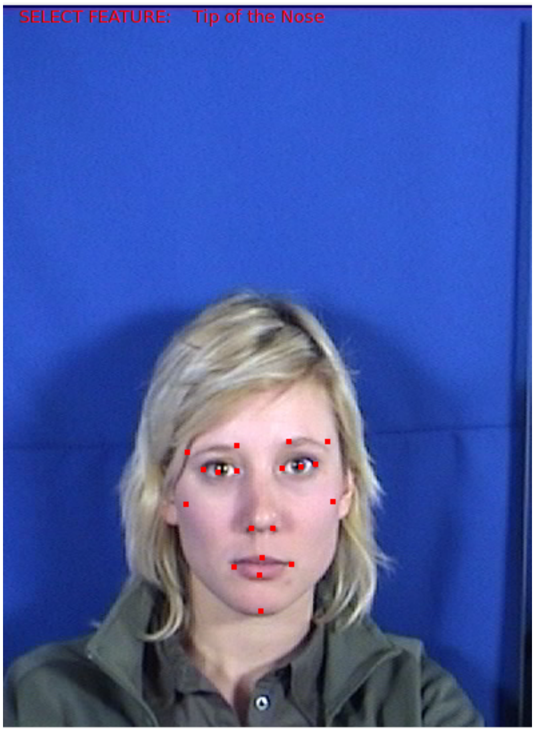


Figure 3.2: Sample 3D face scans from the Binghamton database along with the corresponding texture maps. Taken from [36]

The user selects the feature points simply by clicking on the correct pixel in the image. During the selection process, he is guided with a label which informs him which feature point he should select next. This design is shown in figure 3.3.



SELECT FEATURE: Tip of the Nose

Figure 3.3: Sample 3D face scans from the Binghamton database along with the corresponding texture maps. Taken from [36]

3.4.2 Tracking Feature Points

3.4.3 Camera Parameters

The parameters of this camera model must be obtained by calibrating the camera which was used to capture the image. Since calibration can only be performed if the application has access to the camera, it must also be possible for the user to input camera parameters into the application.

3.4.4 Optimizing the Parameters

Error Function. If there are N feature points f_i, which correspond to

3.5 System Architecture

Chapter 4

Results and Evaluation

Chapter 5

Conclusion

Bibliography

[1] Klaus-Juergen Bathe. *Finite Element Procedures.* Prentice Hall, 1996.

[2] Stan Birchfield. Derivation of kanade-lucas-tomas tracking equation. [http://www.ces.clemson.edu/ stb], January 1997.

[3] Christopher M. Bishop. *Patter Recognition and Machine Learning.* Springer Science + Business Media, LLC, 2006.

[4] V. Blanz, C. Basso, T. Poggio, and T. Vetter. Reanimating faces in images and video. *EUROGRAPHICS*, 22(3), 2003.

[5] V. Blanz, K. Scherbaum, T. Vetter, and H.P Seidel. Exchanging faces in images. *EUROGRAPHICS*, 23(3), 2004.

[6] Jean-Yves Bouguet. Pyramidal implementation of the lucas kanade feature tracker. The paper is included into OpenCV distribution.

[7] Gary Bradski and Adrian Kaehler. *Learning OpenCV.* O'Reilly Media, Inc., 2008.

[8] Donghui Chen and Rober J. Plemmons. Nonnegativity constraints in numerical analysis. *Conference Proceedings of the Symposium on the Birth of Numerical Analysis*, October 2007.

[9] T.F. Cootes and C.J. Taylor. Statistical Models of Appearance for Medical Image Analysis and Computer Vision. *Proc. SPIE Medical Imaging*, 2001.

[10] T.F. Cootes and C.J. Taylor. Statistical Models of Appearance for Computer Vision. mar 2004.

[11] Daniel F. DeMenthon and Larry S. Davis. Model-based object pose in 25 lines of code. *Proceedings of the European Conference on Computer Vision*, pages 335–343, 1992.

[12] Paul L. Fackler. Notes on matrix calculus. North Carolina State University, September 2005.

[13] Vojtech Franc, Vaclav Hlavac, and Mirko Navara. Sequential coordinate-wise algorithm for non-negative least squares problem. *Research Report CTU-CMP-2005-06*, February 2005.

[14] D.F. Gillies. Intelligent data and probabalistic inference lecture notes. *Imperial College London*, 2009.

[15] Duncan F. Gillies. Graphics lecture notes 1. Imperial College London, 2010.

[16] Gene H. Golub and Charles F. Van Loan. *Matrix Computations: Third Edition.* The Johns Hopkins University Press, 1996.

[17] L. De Lathauwer. *Signal Processing based on Multilinear Algebra.* Phd dissertation, Katholieke Universitiet Leuven, Belgium, 1997.

[18] Bruce D. Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision. *International Joint Conference on Artificial Intelligence*, pages 674–679, 1981.

[19] Dimitri Metaxas and Demetri Terzopoulos. Shape and nonrigid motion estimation through physics-based synthesis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(6):580–591, 1993.

[20] Jacey-Lynn Minoi. *Geometric Expression Invariant 3D Face Recognition using Statistical Discriminant Models.* Phd dissertation, Imperial College London.

[21] J.L. Minoi and D.F. Gillies. 3d face and facial expression recognition using tensor-based discriminant analysis methods.

[22] Alex Pentland and Stan Sclaroff. Closed-form solutions for physically based shape modelling and recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(7):715–726, 1991.

[23] W.H. Press, S.A. Teukolsky, W.T. Vetterlin, and B.P. Flannery. *Numerical Recepies in C: The Art of Scientific Computing 2nd Edition.* Cambridge University Press, 1992.

[24] M. Reiter. Neural computation ws 2006/2007 lecture notes. *Vienna University of Technology*, 2006.

[25] Szymon Rusinkiewicz and Marc Levoy. Efficient Variants of the ICP Algorithm. In *Third International Conference on 3-D Digital Imaging and Modeling*, 2001.

[26] J. Nelder S. Singer. Nelder-Mead algorithm. 2009. [http://www.scholarpedia.org/article/Nelder-Mead_algorithm].

[27] W.-H. Steeb. *Matrix Calculus and Kronecker Product with Applications and C++ Programs.* World Scientific Publishing Co. Pte. Ltd., 1997.

[28] Robert W. Sumner and Jovan Popovic. Deformation Transfer for Triangle Meshes. *SIGGRAPH*, 2004.

[29] Carlo Tomasi and Takeo Kanade. Detection and tracking of point features. *Carnegie Mellon University Technical Report CMU-CS-91-131*, April 1991.

[30] M. Turk and A. Pentland. Eigenfaces for Recognition. *Journal of Cognitive Neuroscience*, 3(1):71–86, 1991.

[31] M. Alex O. Vasilescu and Demetri Terzopoulos. Multilinear analysis of image ensambles: Tensorfaces. In *Proc. European Conf. on Computer Vision*, pages 447–460, May 2002.

[32] M. Alex O. Vasilescu and Demetri Terzopoulos. Multilinear independent components analysis. In *IEEE Computer Vision and Pattern Recognition Conference*, San Diego, CA, June 2005.

[33] M. Alex O. Vasilescu and Demetri Terzopoulos. Multilinear projection for appearance-based recognition in the tensor framework. pages 1–8, October 2007.

[34] Paul Viola and Michael Jones. Robust real-time object detection. *Second International Workshop on Satistical and Computational Theories of Vision – Modeling, Learning, Computing and Sampling*, July 2001.

[35] D. Vlasic, M. Brand, H. Pfister, and J. Popovic. Face Transfer with Multilinear Models. *ACM Transactions on Graphics*, 24(3), 2005.

[36] Lijun Yin, Xiaozhou Wei, Yi Sun, Jun Wang, and Matthew J. Rosato. A 3d facial expression database for facial behavior research.