# 2010

# University of North Carolina Wilmington

# Master of Science in

# Computer Science and Information Systems

# Proceedings

https://csbapp.uncw.edu/mscsis

# CONSTRUCTING A 3D MORPHABLE FACE FROM A SINGLE PICTURE BY ESTABLISHING A CORRESPONDENCE FROM 2D ACTIVE APPEARANCE MODEL PARAMETERS

Seiken Higashionna

A Thesis Submitted to the
University of North Carolina Wilmington in Partial Fulfillment
of the Requirements for the Degree of
Master of Science

Department of Computer Science
Department of Information Systems & Operations Management

University of North Carolina Wilmington

2010

Approved by

Advisory Committee

Dr. Karl Ricanek

Dr. Ulku Yaylacicegi

Dr. Eric Patterson
Chair

Accepted by

Dean, Graduate School

TABLE OF CONTENTS

ABSTRACT

This paper presents the construction of 3D face models using a correspondence established between 2D Active Appearance Models (AAM) and 3D Morphable Models (3DMM).The hypothesis is that a 3D face model may be generated via a 2D-to-3D corresponds between Active Appearance parameters and 3D Morphable Model parameters. There are a few different approaches that are currently used to construct 3D morphable face models. This work, in contrast though, is geared toward simplicity and speed in establishing a rough likeness that could be applied in a variety of 3D face software such as games, animation, etc. In theory, this technique will construct a 3D morphable face significantly faster compared to one of the primary techniques thus far, Blanz and Vetter's approach described in their previous work. The quality of generated 3D models is analyzed subjectively and quantitatively to conclude that this has merit.

ACKNOWLEDGEMENTS

I would like to thank my committee members: Dr. Karl Ricanek, Dr. Douglas Kline and Dr. Ulku Yaylacicegi. This work would not have been possible without their support. They have always provided me with good feedback when I asked them questions or had concerns. Dr. Ricanek introduced me to the OpenCV library that helped me a lot. Dr. Kline taught me about NN and gave ideas about how to cope with a situation when a number of data samples is not enough. Also, I would like to thank Dr. Ron Vetter for informing me and helping me proceed through graduate-school procedure.

I would especially like to thank Dr. Eric Patterson for his patience and his guidance. He understands me and directs me to possible paths when I had difficulties. He has also provided me several ideas to improve my thesis. Without Dr. Patterson, I would not be able to complete this work.

Finally, I would like to thank my parents. I would not be able to come this far without their support both financially and with encouragement.

DEDICATION

This thesis is dedicated to all of my committee members. Without their support, this work could not be done. Also, I would like to dedicate this work to students who are trying to do similar work. I will be grateful and honored if this work helps them in any way.

LIST OF TABLES

LIST OF FIGURES

# 1 INTRODUCTION

2D face recognition has been around over 20 years. Several methods have been proposed to conduct 2D face recognition [11]. Transforming 2D to 3D has also become attractive to many researchers. One reason is that 3D morphable models can be applied to many areas such as animation, biometrics, 3D gaming and selecting the fitted size of human wears such as glasses. Especially for the biometrics area, 3D morphable models can provide useful detailed information such as the height of a nose. However, due to the high cost of 3D laser scanner and the limit of measuring environment, it is difficult for individuals to create 3D morphable models and apply these for other applications. This work is geared toward rapidly generating a reasonable 3D likeness with ease, requiring no scanning hardware. This work consists of 2 realms of face processing with a correspondence established between the two. The first is to construct a 2-D face space using Active Appearance Model (AAM) which represent shape and texture using Principle Component Analysis (PCA) methods. The second is to build a 3D Morphable Model (3DMM), also using PCA methods, but applied to 3D shape and texture, lastly establishing a leaned correspondence between the two models using machine-learning techniques.

## 1.1 Hypothesis

There are several methods to construct 3DMMs from a single image, and their results are impressive. Based on these successful works, I introduce a new approach using direct machine learning. This is an unproven technique. My goal is to investigate if this approach will give a satisfactory results. If the result is satisfactory, I will test this work to see if it meets two hypotheses.

1. Constructing a 3DMM face from a single face image can be done very rapidly using machine-learned correspondence between 2D and 3D parameters. Possibly it will be suitable enough to use in real-time application.

2. This technique can construct a 3DMM that provides a suitable likeliness from a single image that is not from a perfectly frontal angle.

## 1.2 Overview of Work

Section 2 describes the database that is used in this work. Section 3 introduces related work and background material and describes the difference between these works and this work. Section 4 describes the methodology used here. Mainly, this work made up of two parts. The first part is 3D processing that creates 3DMM using PCA technique.

The second part is 2D processing that takes texture images and creates eigen textures for use in the 3DMM. Section 5 is about Neural Networks (NN) and Support Vector Machine (SVM), the machine-learning techniques used to establish the 2D-to3D correspondence. Section 6 discusses results from various experiments. In this work, I have tried several methods to improve accuracy. Section 7 discusses the conclusions from this work and possible ways to improve accuracy.

# 2 RERATED WORKS

## 2.1 Blanz and Vetter

There are three main background papers related to the development of this work. Blanz *et.al* [33] talks about constructing 3DMM with using 200 laser scans data acquired by a Cyberware laser scanner. Then they use optical flow to establish better correspondence, finally uses a synthesis by analysis approach to map 2D images into the 3DMM. The process of creating the basic 3DMM in this thesis work is based on their paper. One main difference is that Blanz and Vetter have 200 sample data whereas my dataset contains 100 samples. Since my data has good correspondence, I can omit the adjusting correspondence process unless I add new data that has different data values. Blanz and Vetter uses an iterative approach for the mapping

process. Synthesis by analysis works well and gives a great result. A downside of this approach is a slow non-linear process that takes about a minute to process. One of my goals is to minimize processing time. For this reason I use a different approach for mapping process instead using synthesis by analysis. It is hypothesized that a direct machine-learning may be use to map from 2D face parameters to 3D face parameters.

### 2.2 Sherrah, Faggian and Paplinski

The second paper is by Sherrah *et.al* [27]. Their work is based on Blanz's work. Blanz suggested using AAM to reduce fitting process time. However, Blanz's suggested method required that a user must define a matrix V and shape vector r. V is a matrix that maps the smaller set of 2D coordinates in a image to corresponding 3D vertices in the 3DMM. The shape vector r is obtained from 3DMM shape vector. This second paper focuses a way to automate the fitting process, which undefined values can be obtained by tracking generated 3D models as a base with an AAM built from the projected 3D data constraining 30 degrees of yaw rotations. Another contribution of their work is to show that constructing a 3D face in real-time can be achieved by using automated AAM fittings. Even though they automated the fitting

process, they still have to manually put landmarks on faces. Instead of tracking generated 3D models, I use parameters from a 2D AAM toolkit and map to parameters from 3DMM.

### 2.3 Cootes, Edwards and Taylor

The third paper is by Cootes *et.al* [15]. It describes the original AAM method to match model appearance to an image. This paper is an extension of Active Shape Models (ASM), but AAM search procedure is more robust than ASM search. Although, AAM search is slightly slower than ASM search. Key contributions of this paper are its search speed and number of parameters to describe the shape. Using 400 images to train, it requires only 55 parameters to represent 95% of the observed variance. This is less parameters compared to other 2D face methods on average. Their paper also mentions AAM search speed. They use new face images to search for faces. Active appearance methods form the basis of 2D image representation in this work. Through a linear mapping of the principle components of a well-trained AAM, any face image may be represented in these.

# 3 METHODOLOGY

This work consists up of two major parts that are a 3D processing part and a 2D processing part. It is possible to process each part separately [27]. Although this work involves a texturing process asa well, my main interest is to create a 3D face mesh from a single face image. The 3D processing creates these meshes using PCA and an AAM approach. For coding, I use the OpenCV library to calculate eigenvectors and do other necessary matrix operations for obtaining parameters of each faces to facilitate development [1].

Another tool called AAM toolkit is needed to apply AAM method and to obtain AAM parameters of each faces [14]. After finishing training, this toolkit creates parameters of each faces using a 2D shape and texture PCA methods.

Because I have a different number of parameters and different values of parameters, and these are in a 2D space, a technique is needed to map these as 2D AAM parameters to parameters obtained for 3D PCA. It is theorized that Machine-Learning techniques may learn this mapping. Here, I will experiment with Neural Network and Support Vector Regression.

### 3.1 USF Face Database

The database used in this work is provided by University of Southern Florida (USF) [8]. It contains 100 face samples, and each sample has exact number of vertices and same values of vertex texture and surface . An advantage of using this dataset is that USF has located and adjusted the meshes for a correspondence for each face. A vertex {x1, y1, z1} of one face directly corresponds to {x1, y1, z1} of other faces. This dataset has good correspondence, so this facilitates corresponding a 3DMM without having to first perform this correspondence step [6] [5]. A downside of this dataset is that less than 10% of samples are female. Constructing a female 3DMM using this dataset will not have good result due to lack of female samples. This dataset uses .rgb file for textures. To open this rgb file requires special software. To facilitate to access image data, files are converted for first to jpeg or tiff before processing texture information.

### 3.2 PCA Overview

Principal Components Analysis is a way to identify patterns in the dataset. PCA is used in all forms of analysis from neuroscience to computer graphics because of its simplicity. Once the patterns in the data are identified, this data can be compressed by throwing away some of data that does not influence overall data presentation [22].

PCA is very powerful method to analyze the data.

Here are 5 basic steps of PCA:

1. Calculate mean of the data.
2. Subtract mean from the data.
3. Calculate the covariance matrix of the data.
4. Calculate the eigenvectors and eigenvalues of the covariance matrix.
5. Choose components and form a feature vector if the data needs to be reduced.
6. Derive the new data set: FinalData = transpose(FeatureVector) $\times$ DataAdjust.

The benefits of reducing vectors are to spare disk space and to increase processing speed [31] [11].

### 3.3 PCA steps on 3DMM

For the further information concerning to PCA, the reader is addressed to "A tutorial on Principal Components Analysis" [30] [20].

#### 3.3.1 Create Dataset of 3D Meshes to Build 3DMM

In my database, there are 100 face samples, and each sample contains 75,972 vertices. Vectorsare created that represent each faces by

concatenating x, y, and z coordinates. Since these are 3D data and contain 75,972 vertices, I multiply 75,972 by 3. The size of a vector that can represents one face is 227,916. So, the vectors are as such: face[227916] = $\{x_0, \cdots, x_{75972}, y_0, \cdots, y_{75972}, z_0, \cdots, z_{75972}\}$. Because I have 100 samples, I create 2D matrix with size of 227,916 by 100. Thus, the first column represents the first sample face, and the second column represents the second sample face, and so on.

$$data[227916][100] = \begin{bmatrix} face_{0,0} & \cdots & face_{0,99} \\ \vdots & \vdots & \vdots \\ face_{227915,0} & \cdots & face_{227915,99} \end{bmatrix} \tag{3.0}$$

### 3.3.2 Subtract Mean

After creating the data matrix, the average of data is subtracted from the data matrix to shift the data to the origin. In order to do so, I need to create an average face that is done by adding each face and dividing by a number of faces.

$$average = \sum_{i=0}^{227915} \left( \frac{\sum_{j=0}^{99} face_{i,j}}{100} \right) \tag{3.0}$$

The size of average face array is 227,916 by 1 that looks an equation 3.

$$data[227916] = \begin{bmatrix} face_0 \\ \vdots \\ face_{227915} \end{bmatrix} \tag{3.0}$$

Subtract this average face from original data matrix that is created in 4.2.1 to get a new 227,915 by 100 matrix.

Average 3DMM

This is the average face result. The data vector from equation 3 was used to make an obj file. An obj file is a geometry definition file format containing information of vertex coordinates and texture coordinates. Maya software that is a high end software for 3D modeling and rendering to render the result.

Figure 3.1: Average face front

Figure 3.2: Average face right side

### 3.3.3 Calculate the Eigenvectors and Eigenvalues of Adjusted Data

Before doing this step, there is actually one more step that is to calculate covariance. OpenCV may be used to facilitate this calculation.

OpenCV has a function called EigenVV that gives both eigenvectors and eigenvalues. The problem of using EigenVV is simply not having enough memory space since it squares data matrix in order to calculate eigenvector. The size of memory that would be required to do this operation would be about 49 GigaBytes.

$$memorysize = \frac{227916^2}{1024^3} \approx 48.4GB$$

There is an alternative approach that is Singular Value Decomposition (SVD), used to calculate eigenvectors and eigenvalues without squaring a matrix [25]. U, $\Sigma$, and V from SVD are closely related to eigenvectors and eigenvalues. In [7], the final form of the decomposition is $\mathrm{X} = \mathrm{U}\Sigma V^T$.

> Find some orthonormal matrix P where $\mathrm{Y} = \mathrm{PX}$ such that $\mathrm{C}_Y \equiv \frac{1}{\sqrt{n-1}} YY^T$ is diagonalized. The rows of P are the principal components of X [28].

Let's define matrix Y as an $n \times m$ matrix, and following.

$$Y \equiv \frac{1}{\sqrt{n-1}} X^T$$

where each column of Y has zero mean. The definition of Y becomes clear by analyzing $Y^T$Y.

$$\begin{aligned} Y^T Y &= \left(\frac{1}{\sqrt{n-1}} X^T\right)^T \left(\frac{1}{\sqrt{n-1}} X^T\right) \\ &= \frac{1}{\sqrt{n-1}} X^{TT} X^T \\ &= \frac{1}{\sqrt{n-1}} X X^T \\ Y^T Y &= C_X \end{aligned}$$

By construction, $Y^T$Y equals the covariance matrix of X. The principal components of X are the eigenvectors of Cx. If I calculate the SVD of Y, the columns of matrix V contain the eigenvectors of $Y^T$Y = Cx. Therefore, the columns of V are the principal components of X. If V spans the row space of Y $\equiv \frac{1}{\sqrt{n-1}} X^T$, then V must also span the column space of $\frac{1}{\sqrt{n-1}} X$. Therefore, finding the principal components corresponds to finding an orthonormal basis that spans the column space of X.

Fortunately, OpenCV has a function that performs SVD. Also, according to the OpenCV API, cvSVD function is faster and more accu-

rate than cvEigenVV function. To use this function, I need to create two extra matrices that can hold eigenvalues and eigenvectors. I also need adjusted data matrix that is created in 4.2.2 as an input. Here is how to use this function.

$$cvSVD(dataset, eigenvalue, eigenvector, 0, 0)$$

The first parameter is source matrix that is created in 4.2.2. The second parameter can be either m $\times$ n matrix or a vector to hold eigenvalues of the solution. The third parameter can be either m $\times$ m or m $\times$ n matrix to hold eigenvectors of the solution. The forth parameter is an optional right orthogonal matrix. The last parameter is an optional flag. Without putting values other than 0, this function tries to square matrix that will gives an error due to memory shortage.

Using the SVD function, I can save a significant amount of memory and processing time because the size of eigenvectors are small compared to using cvEigenVV.

$$memorysize = \frac{227916 \times 100}{1024^3} \approx 0.21GB$$

### 3.3.4 Choosing Components and Forming a Feature Vector

If there is no necessary to reduce extra dimension of the data, I can skip this step. Although this work is not about data compression, I

choose to reduce a size of the data but not much.

Nice thing about using cvSVD is that eigenvectors are placed in descending order according to corresponding their eigenvalues. An eigenvector that has a lowest eigenvalue is least important, so it is placed at end of matrix. Thus, it is easy to reduce data.

I have decided to reduce number of eigenvectors to 70 from 100. I did this process visually by reducing eigenvectors by 10 and comparing the results.

#### 3.3.5 Deriving the New Dataset

This is final step and is also the easiest. Once I have finished calculating eigenvectors and reducing the number of eigenvectors, I can get parameters of this dataset which I have created in step 2 by multiplying two matrices.

$$parameter = eigenvector^T \times dataset$$

### 3.4 PCA Steps on Texture

These steps are the same as the mesh PCA calculations, but the data conversion must be performed before performing PCA on image textures.

1. Convert .rgb file format to .jpg format so that OpenCV can read texture file.

2. After reading all texture files, convert each image format to matrix format so that I can use cvSVD function.

3. Use the same steps as the mesh PCA to get parameters of the textures.

4. After generating a new texture or re-constructing a texture, the data needs to be converted back to image format so that I can use it as a texture image.

### 3.4.1 Convert File Format

The textures that are provided are saved as .rgb file format. This rgb format contains a color bitmap image format created by Silicon Graphics [34] [16]. Unfortunately, cvLoadImage in OpenCV does not support this file format. I need to use an image file format converter that can convert .rgb file format into most common file format such as .jpg. I used Autodesk's fcheck application to convert .rgb file format images into .jpg file format [4].

Texture file

These are original images that come with the dataset.

Figure 3.3: Texture of first face in dataset.

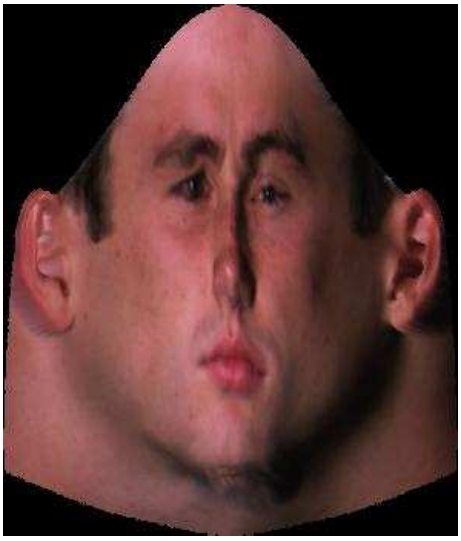

Figure 3.4: Texture of second face in dataset.

### 3.4.2 Convert Image Format to Matrix

The image data can be read easily by using cvLoadImage function There is a function called cvCalcEigenObjects that calculates eigenvectors and eigenvalues. Unfortunately, I have to read images as gray-scale in order to use this function to obtain eigenvectors and eigenvalues, otherwise, it gives an error because cvCalcEigenObjects does not take a vector that has multiple channels as a parameter.

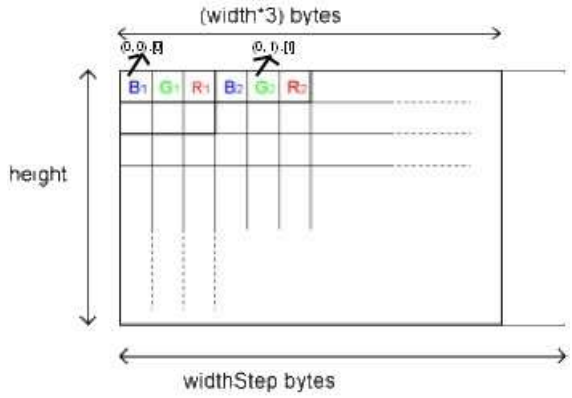


Figure 3.5: Color image is stored like this.

Basically, a color image is stored $N \times M \times 3$ which is length, width and RGB values. In order to perform PCA on color texture images, I decide to create data matrix that is described in section 4.2.1 and perform PCA on this data matrix. I can access a pixel in image file by doing image->ImageData[i] where i represents a position that stores the data. Below is a sample code for creating the data matrix.

partial code

```
for ( int i = 0; i < image->ImageSize; i++ ){
    for ( int j = 0; j < 100; j++ ){
        cvmSet(data, i, j, image[j]->ImageData[i]);
    }
}
```

The size of image is 308,136, and there are 100 texture images. So,

the size of image data matrix is 308,136 by 100 which looks like below.

$$imageDataMatrix[308136][100] = \begin{bmatrix} texture_{0,0} & \cdots & texture_{0,99} \\ \vdots & \vdots & \vdots \\ texture_{308135,0} & \cdots & texture_{308135,99} \end{bmatrix} \quad (3.5)$$

### 3.4.3 Perform PCA to Get Parameters

To get parameters, I take the same steps as I described in 4.2.3 and 4.2.5. (This process takes about 63 seconds on 2.13GHz Intel Core 2 Duo with 4GB memory.)

### 3.4.4 Turn Matrix into Image Format

This process is basically opposite of 4.3.1. A new image is created by using cvCreateImages(cvSize(image->hight, image->width), image->depth, image->nchannels). This creates image variable and allocates memory space. Then, I copy all values to an image variable and save the result after reconstruct texture image based on parameters that are give by a user.

partial code

```
for ( int j = 0; j < 100; j++ ){
    for ( int i = 0; j < image->imageSize; i++ ){
```

```
            Image[j]->ImageData[i] = cvmGet(data, i, j);
        }
        cvSaveImage(saveDirectory, image[j]);
    }
```

When using cvLoadImage, OpenCV reads and stores an image in BGR order. An image format must be converted back to RGB format if an image format is converted during image processing.

### 3.5 Active Appearance Model

To get AAM parameters, I use AAM toolkit provided by Dr. Cootes site. Unfortunately there is no access to the original source code. So, I do not have a way to access other data such as each eigenvector except writing out AAM parameters on files and using these parameters to reconstruct images.

This tool provides a GUI to make landmarks on images; then, builds an AAM and creates parameters. After training is done, this generates 3 types of parameters that are for textures, for shapes and for combination of texture and shape. AAM also uses PCA to generate parameters. Before writing out parameters on files, this program seems to normalize parameters.

### 3.5.1 Setup

In [15], AAM uses less parameters to represent a large portion of a image. I first use total of 84 landmarks that are put around eyes, a nose, a lip, chin, and outline. However, I increase a number of landmarks from 84 to 161 and representing areas so that AAM can generate more accurate parameters of images.

Trying to increase accuracy, additional 510 2D face image are added to AAM database. GUI looks like below.

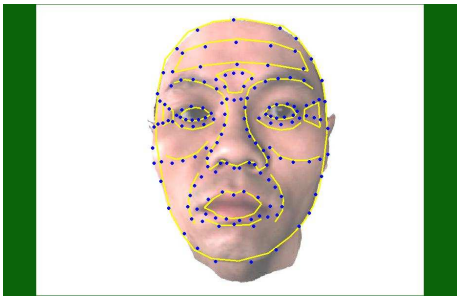

Figure 3.6: Markup sample 1.

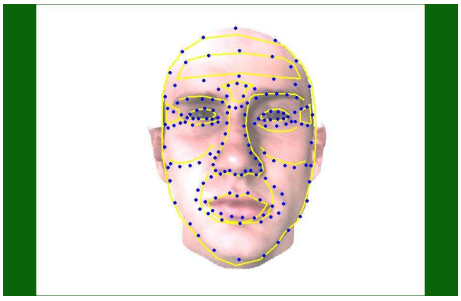

Figure 3.7: Markup sample 2.

### 3.5.2 Get AAM Parameters

There is a configuration file where a user can specify model accuracy, shape accuracy, and texture accuracy. If model accuracy is set to a lower percentage, this program generates fewer parameters, and training time takes much less. I set model accuracy to 97% and get

207 parameters by typing following command, and training time is about 15 minutes for both apm and aam.

```
./am_get_param -i Model.smd -d /home/params/
```

where Model.smd is source file and /home/params/ is directory where generated parameters are saved. I only need `.b_app` files because these are the combination of textures and shapes. (The other two files are only for textures and shapes.) The parameter files represent parameters of the appearance (PCA of shape and texture) components to reconstruct face images.

### 3.5.3 Reconstructing Images

I reconstruct AAM images to see if the AAM toolkit can reconstruct models that resemble original images. If the AAM modeled built can reconstruct models that resemble the original images, I know that the AAM works as expected.

#### Contents of File

`.b_app` file is a text file. It contains just parameter values.

0.034

$-0.002$

$-0.43$

$\vdots$

0.008

In oder to reconstruct the image with the AAM toolkit software, the number of parameters and file name need to be added on top of `.b_app` file. The model is then used to generate an appearance model instance with the parameters, which is saved to the given file name. The file looks like below.

*result.jpg*

207

0.034

$-0.002$

$-0.43$

$\vdots$

0.008

Result of Constructed Image

The following figures are some of reconstructed images with 97% model accuracy.

Figure 3.8: Reconstructed image of first face in dataset.

Figure 3.9: Original image of first face in dataset.

Figure 3.10: Reconstructed image of Seiken.

Figure 3.11: Original image of Seiken.

Figure 3.12: Reconstructed image of Bar Refaeli.

Figure 3.13: Original image of Bar Refaeli.

Figure 3.14: Reconstructed image of Barack Obama.

Figure 3.15: Original image of Barack Obama.

# 4 MACHINE LEARNING TECHNIQUE

The previous work [33] and [27] use an iterative approach to fit 3DMM into 2D image. They have very successful results, but the downside of iterative approach is that it takes time to get accurate results. To overcome this issue, I use two machine-learning techniques, Neural Network and Support Vector Regression, to generate shapes and textures for new images. Assuming network training is completed, this approach will generate parameters of a new image less than a second.

## 4.1 Neural Network

In [23], Michael Negnevitsky discusses Neural Network (NN) algorithms and applications. The most popular algorithm is simple back-propagation (BK) because BK works well for most real life problems. Because of its simple algorithm, training time takes less compare to other fancy algorithms.

For using NN, the first question that comes to my mind is finding the number of layers and neurons that are need. Basically, I can have as many neurons and layers as I want, but increasing the number of layers and neurons becomes computationally expensive with no added improvements. Blanz *et al.* [7] discuss that 3DMM can be constructed by linear combination of a large number of 3D face data. Thus, I use a three-layer network because three-layer network works well for continuous problems. As discussed in [23] [26], the network should not need more than four layers unless it is for experimental purpose because each additional layer increases computational burden exponentially.

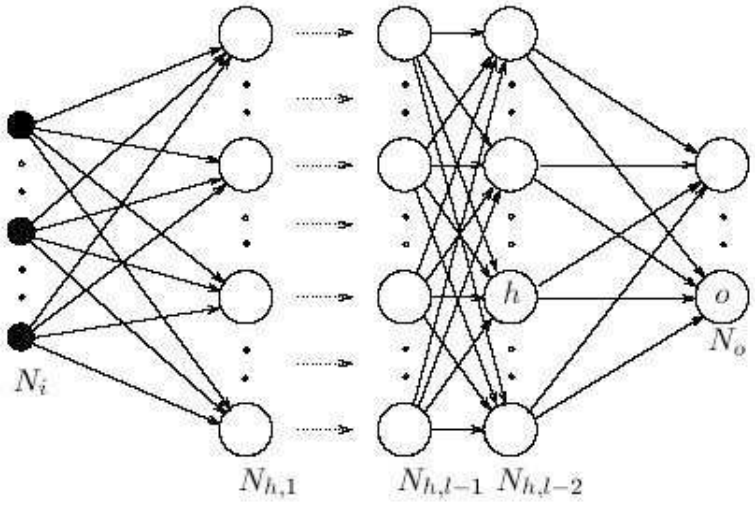


Figure 4.1: Neural Network Setup.

OpenCV has a Neural Network function called CvANN_MLP, but this function has one problem. Its maximum iterations are 1000 epochs which are not enough to generate new parameters for 3DMMs and tex-

tures.

I use MATLAB to process NN because MATLAB has a more sophisticated NN function than CvANN_MLP. On MATLAB, it is easier to create Neural Network by typing following command.

```
net = newff(in, [206 50], {'sigmoid', 'purelin'}, 'trainrp')
```

where in represents input vectors, [206 50] means that this network creates 206 neurons in a middle layer and 50 outputs. Next parameters are types of function to calculate values. This network uses sigmoid function in a middle layer and linear function to calculate output values. The last parameter represents a training algorithm.

In my setup, there are 401 neurons in middle layer and 100 outputs. After Neural Network setup, train(net, data1, data2) function trains network until one of criteria that is either to reach goal or to reach maximum iteration is met. A maximum iteration is set to 10000 epochs, and goal is set to less than 0.0001. Lastly, I use sim(trained net, new input) function to obtain the results.

### 4.1.1 NN Training Algorithm

There are more training algorithms available to use. Other available algorithms are shown in following figure.

| Acronym | Algorithm | |
|---|---|---|
| LM | trainlm | Levenberg-Marquardt |
| BFG | trainbfg | BFGS Quasi-Newton |
| RP | trainrp | Resilient Backpropagation |
| SCG | trainscg | Scaled Conjugate Gradient |
| CGB | traincgb | Conjugate Gradient with Powell/Beale Restarts |
| CGF | traincgf | Fletcher-Powell Conjugate Gradient |
| CGP | traincgp | Polak-Ribiére Conjugate Gradient |
| OSS | trainoss | One Step Secant |
| GDX | traingdx | Variable Learning Rate Backpropagation |

Figure 4.2: Available NN training algorithm in MATLAB

According to the MATLAB manual, they are listed in an ascending order. On the top of list, the algorithm takes less time to calculate, but it requires more memory so that it cannot be used with large data inputs.

On the other hand, the algorithm on the bottom does not have a memory problem, but the downside is that a learning rate is extremely slow. For more rapid training, I choose to use Resillient Backpropagation algorithm. I would like to use Levenberg-Marquaedt algorithm or BFGS Quasi-Newton algorithm, but these algorithms give memory errors due to the size of input data.

- A trainlm is a combination of gradient descent and Gauss-Newton algorithm that is used to salve non-linear least squares problem. A trainlm is designed to approach second-order training speed without calculating a Hessian matrix because it can estimate a Hessian matrix from a Jacobian matrix [21] [24].

- A trainbfg is a type of the Quasi-Newton method that is based on Noeton's method to find local maxima and minima of a function where the gradient is 0. A trainbrg does not require to calculate Hessian matrix of second derivatives of the function.

- A trainrp is a learning heuristic method that the signs act as weights. It trains network by updating values when a sign is changed. Since the signs act as weights, this algorithm does not require a large memory space.

- A trainscg is based on the Conjugate Gradient Algorithm (CGA) that adjusts the weights in the steepest direction. A Scaled Conjugate Gradient Algorithm is designed to avoid a problem of the CGA by combining the model trust region approach with the CGA.

- A traincgb is a type of the Conjugate Gradient Algorithm. This algorithm resets the search direction if a condition, $|g_k^T - {}_1g_k| \geq 0.2||g_k||^2$, is satisfied.

- A traincgf is a type of Conjugate Gradient Algorithm with update function $\beta = \frac{g_k^T g_k}{g_{k-1}^T gk-1}$.
- A traincgp is a similar to the traincgf with different update function $\beta = \frac{\Delta g_k^T - 1 g_k}{g_{k-1}^T gk-1}$.
- A trainoss uses the secant method that is approximate a root of the function by using a secant line.
- A traingdx is a network training function that updates weight and bias values according to gradient descent momentum and an adaptive learning rate.

## 4.2 Support Vector Machine

Support Vector Machine (SVM) is one of the best classification methods available today. It is not only used for classification but also used for regression. SVM uses a set of hyperplane in an infinite dimensional space to classify data.

### 4.2.1 SVM Basics

As I mentioned above, SVM constructs hyperplanes to separate data [10] [32] [31]. A following figure describes basic idea of SVM.

The larger margin space will decrease generalization error.

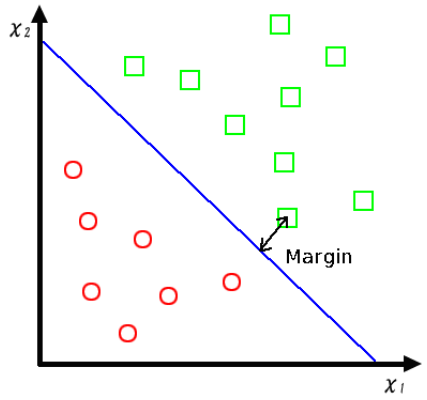


Figure 4.3: SVM linear separation

In a real situation, there are many problems that cannot be linearly separable such as following figure.

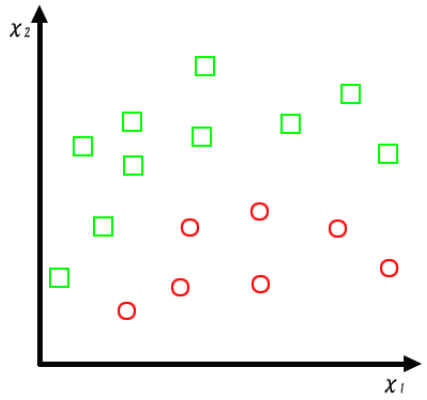


Figure 4.4: Linearly unseparable

In this case, there is no linear line that can separate these data. In order to fix this issue, $X_2$ values are multiplied by $(X_1 + 1)$ to transform like a figure below.

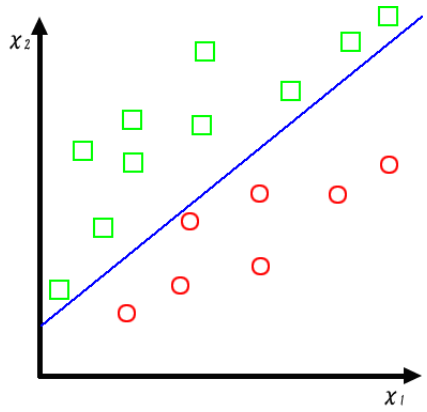


Figure 4.5: Linearly separable

Instead using a function to transform data, SVM defines a kernel function that represents inner product of patterns on n dimensional feature spaces.

### 4.2.2 Learning

Generally, the learning caliber decides an identification plane using a known pattern by updating the value of the parameter of the learning device. The problem called "the local most suitable solution" (local minima) occurs in the neural networks using the error reverse propagation method by a process of this learning. Learning stops with the local best value in some ranges instead with the absolute best value that this is the whole space where the parameter of the learning device forms on.

In SVM, the process of learning finally results in the issue of second

optimization by using an undecided multiplier of Lagrange. Because SVM ends up in global minimum, SVM does not have a problem that is to trap in local minima [18].

### 4.2.3 OpenCV SVM Function

OpenCV has a SVM function that generally perform well. Unlike CvANN_MLP function, SVM function does not have a limited number of iterations. To set the SVM up, all I have to do is to declare SVM variable by CvSVM svm. I use this variable to train SVM.

```
svm.train(training_data, response, NULL, NULL, params);
```

where training_data are input data, and responses is a vector that is a target outcome. According to the API, next 2 parameters are *var_idx* and *sample_idx* that specify which specific data to train. I need to train all samples, so I put NULL to indicate using all variables and samples using for training. The last parameter is training params that a user can set: what kind of kernel to use, how big a margin of hyperplane will be, and other rules.

### 4.2.4 SVM Parameters

There are 5 types of SVM [13].

- CvSVM::C_SVC - this svm does n-class classification (n>=2), and allows imperfect separation of classes with a penalty multiplier C

for outliers. Then this regularizes support vector classification using standard algorithm.

- CvSVM::NU_SVC - this svm does n-class classification with possibly imperfect separation. Parameter nu (in the range 0..1, the larger the value, the smoother the decision boundary) is used instead of C that is the penalty multiplier. This automatically regularizes support vector classification.

- CvSVM::ONE_CLASS - this svm is one-class SVM; All the training data are from the same class; SVM builds a boundary (hypersphere) that separates the class from the rest of the feature space.

- CvSVM::EPS_SVR - this uses a regression. The distance between feature vectors from the training set and the fitting hyperplane must be less than p (margin). This kernel trains the network until error rate reaches a value that is given by a user. For outliers the penalty multiplier C is used.

- CvSVM::NU_SVR - this uses a regression. This kernel automatically minimizes epsilon value. A value of nu controls the number of support vectors.

Also, there are four kernel types available in OpenCV's implementation.

- CvSVM::LINEAR - no mapping is done, linear discrimination (or regression) is done in the original feature space. It is the fastest option. d(x, y) = x · y == (x, y)

- CvSVM::POLY - polynomial kernel: d(x, y) = (gamma × (x · y) + coef0)$^{degree}$. This kernel uses polynomial function to fit a non-linear relationship between value of x and the corresponding value of y.

- CvSVM::RBF - radial-basis-function kernel: d(x, y) = exp(-gamma × |x - y|$^2$). The data is transformed into a Hilbert space of infinite dimension. This kernel suits for solving complex non-linear classification problem.

- CvSVM::SIGMOID - sigmoid function is used as a kernel: d(x,y) = tanh(gamma × (x · y)+ coef0). This kernel is similar to RBF if x, y and coef are in certain range, but this kernel is generally not better than RBF [19].

The OpenCV API states that RBF kernel is good choice for most applications [9]. Since 3DMM linearly transforms one face to another, choosing a linear kernel may be good enough. However, I chose linear

kernel and polynomial kernel for comparison purpose.

Degree, gamma and coef0 are the parameters of the kernel and can be set by a user. C, nu and p are used in the generalized SVM optimization problem.

Increasing the value of p, the result will be more generalized. Moreover, training time decreases as the value of p increases. If p is close to zero value, the result will be more detailed, but training time increases significantly.

*params.crit* is the termination procedure for iterative SVM training procedure. A user can set the followings:

- epsilon - if an error rate is less than epsilon, SVM training stops. ( use as params.crit.epsilon = 10000)
- max_itr - SVM training stops if iteration reaches maximum iteration that is set by a user. ( uses as params.crit.max_itr = 0.001)
- type - set which termination criteria to use. if a user sets param.crit.type = `CV_TERMVRIT_EPS` | `CV_TERMCRIT_ITER`, SVM training stops after one of goals are met.

### 4.2.5 Kernel Parameters of SVM

For linear kernel, I set c = 800 and p = 1 to get the best result. If I increase p value or decrease c value, the result will be closer to averaged face. Since no mapping is done, linear kernel is the fastest kernel available using SVM.

As for the polynomial kernel, I use 2nd degree polynomial and 3rd degree polynomial with different parameter settings. Since polynomial kernel does mapping, d(x,y) = (gamma $\times$ (x $\cdot$ y) + coef0)$^{degree}$, I have to set coef value also. I set c = 200, p = 100 and coef = 12 for 2nd degree polynomial kernel. I set c = 25, p =100 and coef =12 for 3rd degree polynomial kernel.

As described in [12], I performed several experiments to choose these values subjectively for the best visual results. In my experiments, Support Vector Regression (SVR) is used. SVR is based on computation of regression in the higher dimensional spaces. SVR is based on the concept of SVM that characterizes the properties of learning machines enabling them to generalize well to unseen data [2].

### 4.2.6 Prediction

In OpenCV, a prediction using SVM is different from NN prediction. For NN, I pass an empty output vector to hold the result. However, because the response of SVM classification must be one vector, I have to train and make predictions one at a time for a number of desired output.

partial code

```
for (int p = 0; p < nparam; p++) {
   cvGetCol(pcaTrans, &temps, p);
   svm.train(aamTrans, &temps, NULL, NULL, params);
   answer[p] = svm.predict(sample_transpose);
}
```

All input vectors and matrices are row based, so I have to transpose input data matrix and response data matrix before I use the training function. As the name describes, pcaTrans and aamTrans are matrices that are already transposed. The vector temp is a one-dimensional vector that holds response value. I use cvGetCol to get first response values and save them to a temp vector. The vector `sample_transpose` is new input data, and I use this new input to predict a new value based on response values.

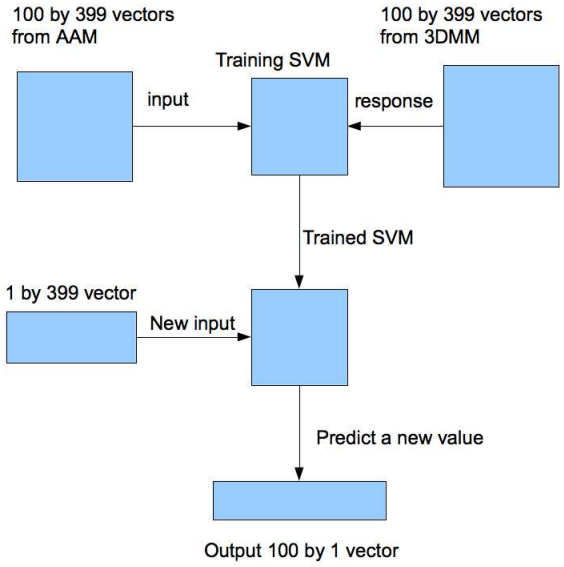


Figure 4.6: SVM processing image

The figure above is a summary of this process. This whole process loops for a number of parameters. (e.g. if I have 50 parameters, this process repeats 50 times to get 50 new outcomes. )

Because Dr. Cootes combines shape parameters and texture parameters to create `.b_app` file, I also combine shape parameters and texture parameters to accommodate myself to Dr. Cootes. Basic work flow is shown in a figure below.

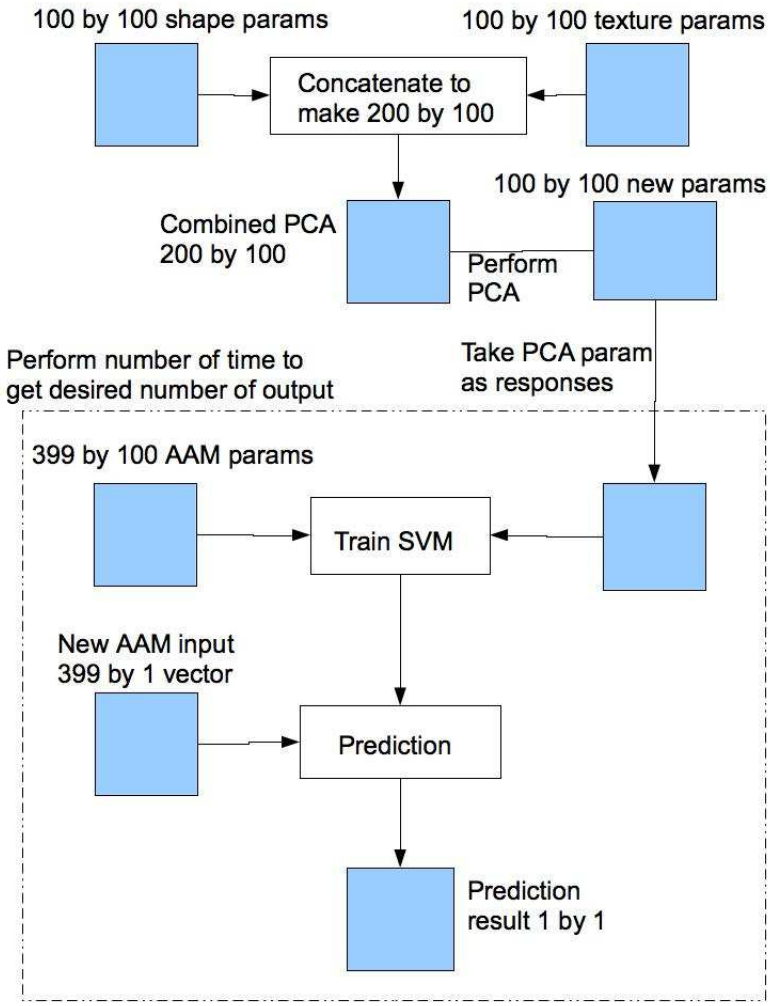


Figure 4.7: Overall process so far

After combining these parameters, I apply PCA on these combined parameters to make 100 by 100. Then I use Support Vector Regression to train and to generate a new parameter. A result after this process is 100 by 1 vector. In order to get shape and texture parameters of a new input, I have to perform reverse PCA on this vector. A final result is 200 by 1 parameter vector, which half of this is a shape parameter and other half is a texture parameter of a new input.

#### 4.2.7 Other Advantage of Using SVM

In 5.2.2, I mentioned an advantage of SVM was that SVM would not have multiple solutions associated with local minima. Other reasons are followings [3].

- SVM provides a good generalization by using appropriate kernel and choosing C and margin.
- SVM takes significantly less time to train and provides a decent result. Using macbook with 2.13GHz Intel Core 2 Duo and 4GB memory, training time and getting a result take less than 1 second.
- By choosing an appropriate kernel, SVM gains flexibility that data does not have to be linear and also needs not to have the same functional form for all data.
- SVM require less samples compared to Back-Propagation Neural

Network.

- Also, a most common advantage using Support Vector Machine/Regression is to avoid difficulties of using linear function in the high dimensional feature space.

# 5 RESULTS

## 5.1 Experimental Methodology

After I get new AAM parameters, I run Support Vector Regression with 3 different kernels and Neural Network using these new AAM parameters as inputs. Then predict a new 3DMM parameter based on 100 data samples.

Since I do not have enough samples to predict new texture, I use MAYA to render final image with original image projected. A reason why I do this is that I can compare texture part separately. Then I should be able to estimate how good/bad it is by comparing 3DMM with projected original image to 3DMM with reconstruct texture.

The AAM parameters themselves may be needed to quantify face reconstruction Euclidean distance between the AAM parameters of a

3D rendered face may be compared to original 2D AAM parameters.

### 5.1.1 Comparing Parameters

After I render a frontal image of constructed 3DMM, I use AAM toolkit to landmark and get new parameters for the image. I now have two parameters that are original one and constructed one to get quantitative measurement.

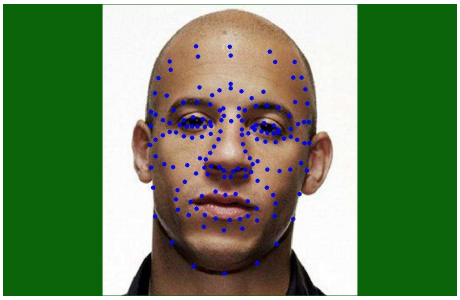

Figure 5.1: Original image

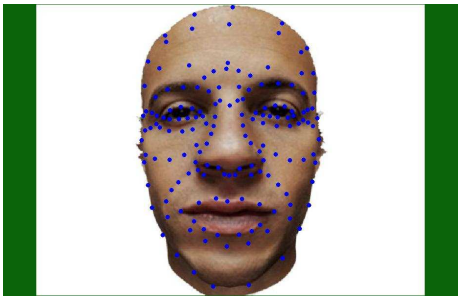

Figure 5.2: Image of construceted 3DMM

### 5.1.2 Euclidean Distance

Euclidean Distance is a straight distance between two points. In this case, it measures distance between original AAM parameters and AAM parameters from constructed 3DMM.

$$Distance = |\mathbf{x} - \mathbf{y}| = \sqrt{\sum_{i=0}^{398} |x_i - y_i|^2} \tag{5.2}$$

Equation (5) is am Euclidean Distance formula. A range of answer is between 0 and $\infty$. An answer that is close to 0 is better in terms of the numerical value.

## 5.2 Rendered Results

I chose five celebrity images to test these methods. Original subject images are shown below figure.

Figure 5.3: From left: Barack Obama, Chris Rock, Christopher Reeves, Jet Li and Vin Diesel

Barack Obama

Figure 5.4: Top row: Original image projected. Bottom row: constructed texture used.

From left column, a linear kernel, 2nd polynomial kernel, 3rd polynomial kernel and Back-propagation Neural Network are used to construct 3D mesh.

Looking at top row, images are recognizable as Obama face because they have an original 2D image for their texture. Polynomial kernel makes slightly rounder face compared to linear kernel. Also, linear kernel creates smoother face because ears of other images are jaggy. Compared to an original image, Obama's lips become wider.

As texture, it is simply not enough samples to construct new texture. It seems that a texture result using higher degree of polynomial kernel becomes worse. A polynomial result is further away from an averaged texture.

Chris Rock

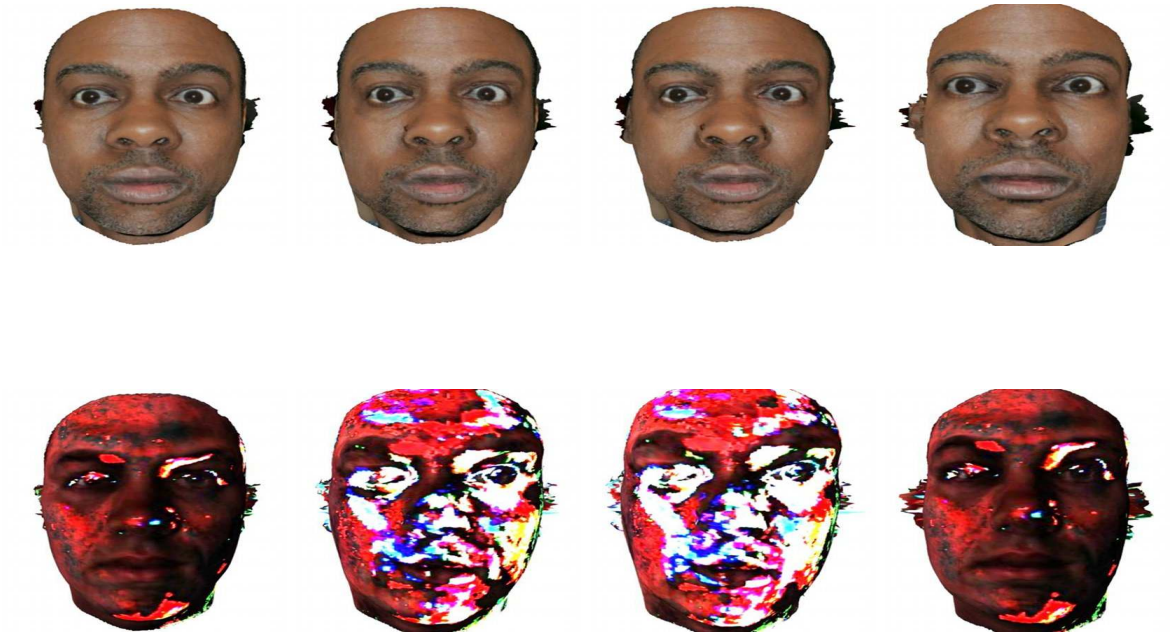

Figure 5.5: Top row: Original image projected. Bottom row: constructed texture used.

From left column, a linear kernel, 2nd polynomial kernel, 3rd polynomial kernel and Back-propagation Neural Network are used to construct 3D mesh.

Unfortunately, the results of Chris Rock are not recognizable. It is hard to see resemblance to an original image. It has trouble to construct a wider chin because there are not enough samples in dataset. Compare between these three, 3rd degree polynomial is closer to an original in terms of shape.

As for texture, results from polynomial are worse than a result from linear just like results of Obama. A race is other possible reason why these results are not good. Majority of sample is Caucasian male, so constructing darker skin will have troubles.

Christopher Reeves

Figure 5.6: Top row: Original image projected. Bottom row: constructed texture used.

From left column, a linear kernel, 2nd polynomial kernel, 3rd polynomial kernel and Back-propagation Neural Network are used to construct 3D mesh.

Judging from previous 2 results, polynomial kernel tends to make rounder face shape, but this is extreme example. Looking at the original image, a shape of Christopher Reeves is more like left column. A reason may be that I did not make AAM landmark well. Of course, data sample size will unfortunately be always an issue.

Because Christopher Reeves has light skin, results of texture are not as bad as Obama and Rock.

Jet Li

Figure 5.7: Top row: Original image projected. Bottom row: constructed texture used.

From left column, a linear kernel, 2nd polynomial kernel, 3rd polynomial kernel and Back-propagation Neural Network are used to construct 3D mesh.

Shapes of Jet Li are not bad, but it seems polynomial kernel shapes shrink a little. Actually, all shapes of Jet Li looks okay except lips. The basic face shape of Jet Li is captured well. It looks acceptable except for the lips of constructed face are little wider than an original image.

As for texture, it seems that Jet Li's texture has less trouble than Christopher Reeves's although this is still far from decent texture construction.

Vin Diesel

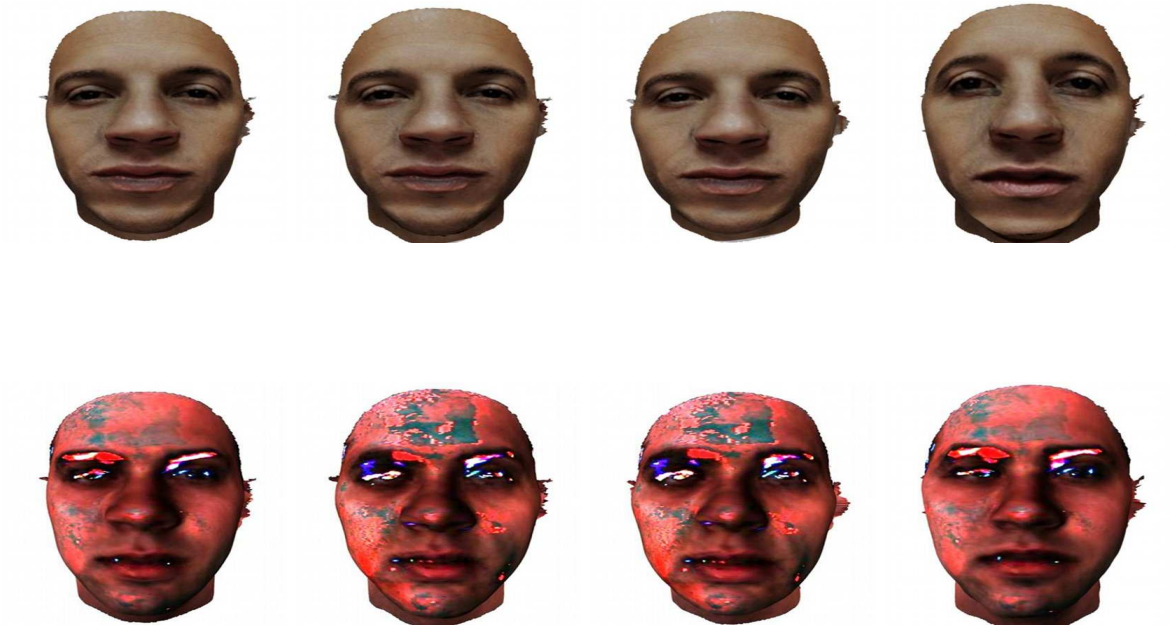

Figure 5.8: Top row: Original image projected. Bottom row: constructed texture used.

From left column, a linear kernel, 2nd polynomial kernel, 3rd polynomial kernel and Back-propagation Neural Network are used to construct 3D mesh.

The results of Vin Diesel turned out to be the best outcomes. There are not much differences between each kernels. One of the reason is Vin Diesel probably has a face that is not far away from an averaged face. Compare to the other results, a shape from polynomial kernel is not jaggy, and also a texture from linear kernel is not as different as a texture from 3rd degree polynomial kernel.

## 5.3 Quantitative Results

I use Euclidean distance to measure distance of parameters between constructed results and their original. Lower points are better results because lower value means it is closer to the original.

Table 1: Quantitative results with projected images

| | Vin Diesel | Barack Obama | Jet Li | Chris Rock | Christopher Reeves |
|---|---|---|---|---|---|
| linear | 0.8475 | 0.9105 | **0.6631** | 0.9505 | 0.7812 |
| 2nd degree poly | 0.8142 | 0.9451 | 0.6844 | **0.8298** | 0.7058 |
| 3rd degree poly | 0.8205 | 0.9502 | 0.7006 | 0.8379 | 0.6582 |
| neural network | **0.7457** | **0.7283** | 0.8399 | 0.9738 | **0.6250** |

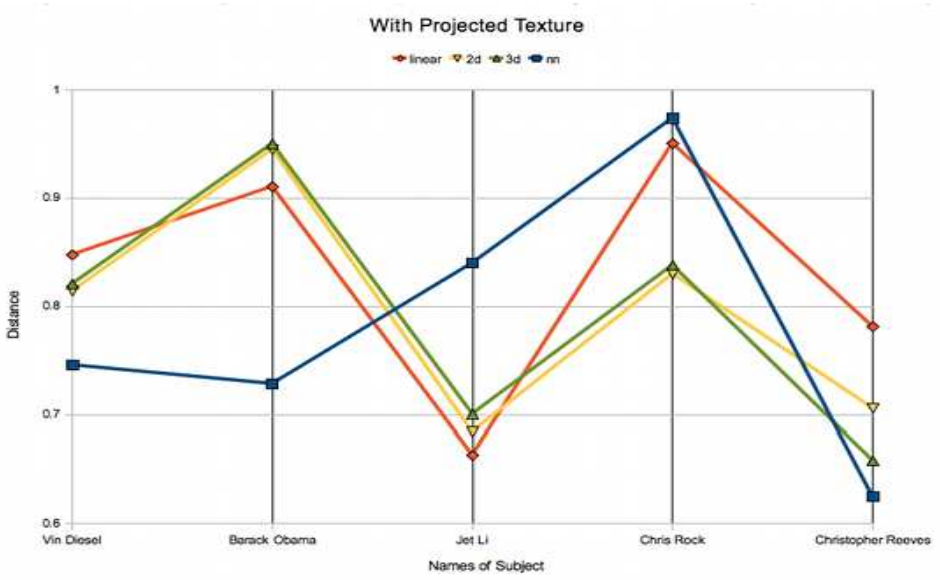


Figure 5.9: With Image Projected

Contrary to my initial expectations, Neural Network performs well quantitatively. Because experimenting with just 5 faces is not enough for broad conclusions, there is a room for argument. However, it is

still impressive that 3 results from Neural Network are better than other methods. See figure 6.16, for Neural Network results the blue line indicate good performance constructing Obama's face.

Judging from a whole graph, SVM with 2nd degree polynomial (2nd poly) is better choice for this work. Table 1 shows that the results of 2nd poly lies between the best and the worst. This shows that 2nd poly generalizes better than other methods. Because SVM parameters affect tje outcome, it is possible that these results could be improved by more experimentation guided toward choosing optimal parameter [13].

Table 2: Quantitative results with constructed textures

| | Vin Diesel | Barack Obama | Jet Li | Chris Rock | Christopher Reeves |
|---|---|---|---|---|---|
| linear | 1.0924 | 1.1255 | **1.0178** | 1.2658 | 1.0238 |
| 2nd degree poly | 1.0843 | 1.2807 | 1.0724 | 1.3557 | 1.0355 |
| 3rd degree poly | 1.0649 | 1.2967 | 1.0406 | 1.3484 | 1.0744 |
| neural network | **0.9407** | **1.0252** | 1.1235 | **1.1663** | **0.9351** |

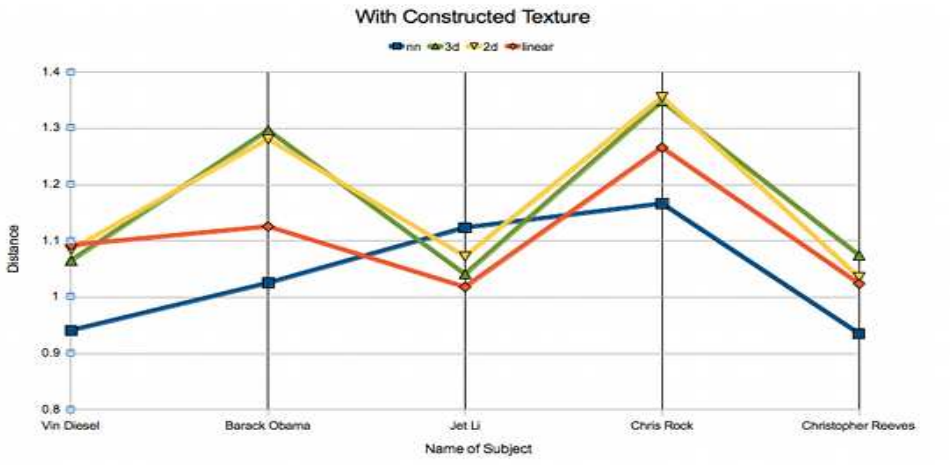


Figure 5.10: With Constructed Texture

Since textures are not good, it is easy to understand why these results are bad. One interesting point is Neural Network now has 4 best results. Comparing between SVM kernel, a linear kernel performs better than polynomial kernels. Because of lack of samples, textures that are not far from an average are better than unique textures that are generated by polynomial kernels. PCA calculates Euclidean distance as the measurement of dissimilarity among objects [29].

### 5.3.1 Original Images and the Best Rendered Images

According to quantitative measurement, these are the original images and best-rendered images produced.

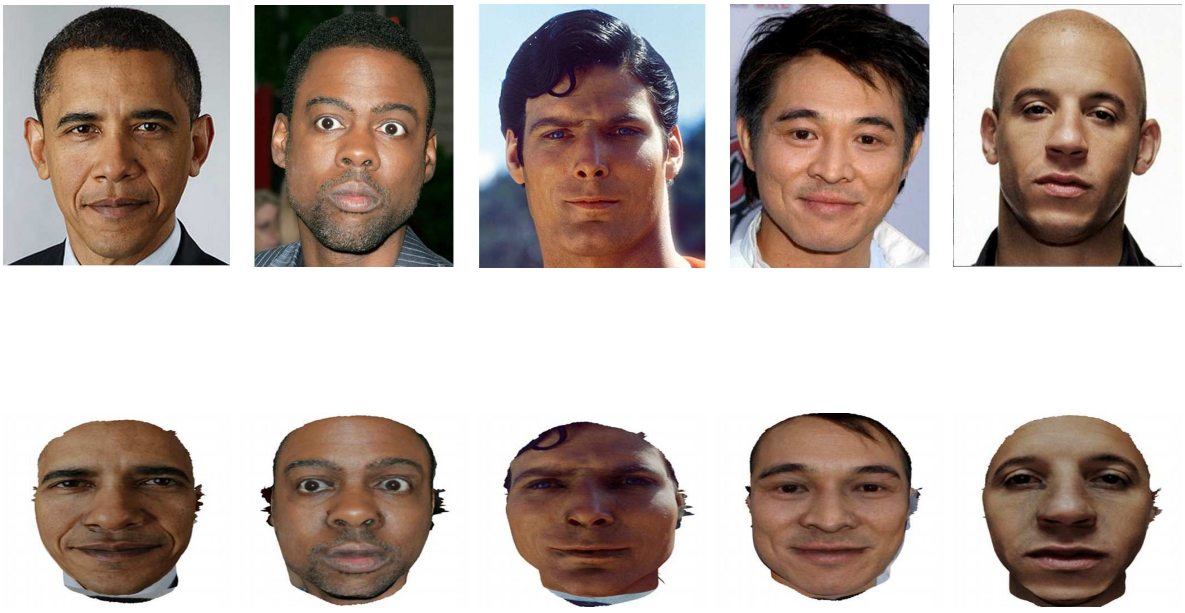

Figure 5.11: From left: Barack Obama, Chris Rock, Christopher Reeves, Jet Li and Vin Diesel

- Barack Obama - Neural Network gives a best result.
- Chris Rock - SVM with 2nd degree polynomial kernel gives a best result.
- Christopher Reeves - Neural Network gives a best result.
- Jet Li - SVM with Linear kernel gives a best result.
- Vin Diesel - Neural Network gives a best result.

I take AAM parameters from constructed 3DMM and generate images using AAM toolkit. The purpose of doing this is to see how images of constructed 3DMM may be seen by AAM toolkit.

Barack Obama AAM image

Figure 5.12: Top row: projected image used. Bottom row: constructed image used.

An original image and images from a linear kernel, 2nd polynomial kernel, 3rd polynomial kernel and Back-propagation Neural Network are used.

By looking at final rendered Obama images, it is hard to visually decide which one is better. However, it is a bit easier to decide which one is better by looking at AAM images. As quantitive measurements indicate, a result from Neural Network is clearly better.

Chris Rock AAM image

Figure 5.13: Top row: projected image used. Bottom row: constructed image used.

An original image and images from a linear kernel, 2nd polynomial kernel, 3rd polynomial kernel and Back-propagation Neural Network are used.

Because of shape of lips, it is clear to see SVM with polynomial kernel gives better results. However, it is hard to see which polynomial kernel is better by looking at final images and AAM images. This is because the difference between these two is so small. Human vision may not be able to detect the difference.

Christopher Reeves AAM image

Figure 5.14: Top row: projected image used. Bottom row: constructed image used.

An original image and images from a linear kernel, 2nd polynomial kernel, 3rd polynomial kernel and Back-propagation Neural Network are used.

The results of Christopher Reeves surprise me because I understand the result from Neural Network is better by looking at final rendered results. But by looking at AAM images, I think the results from SVM with 3rd degree polynomial kernel looks closer to the original.

Quantitative measurement indicates the result from Neural Network is better although their difference is about 0.033.

Jet Li AAM image

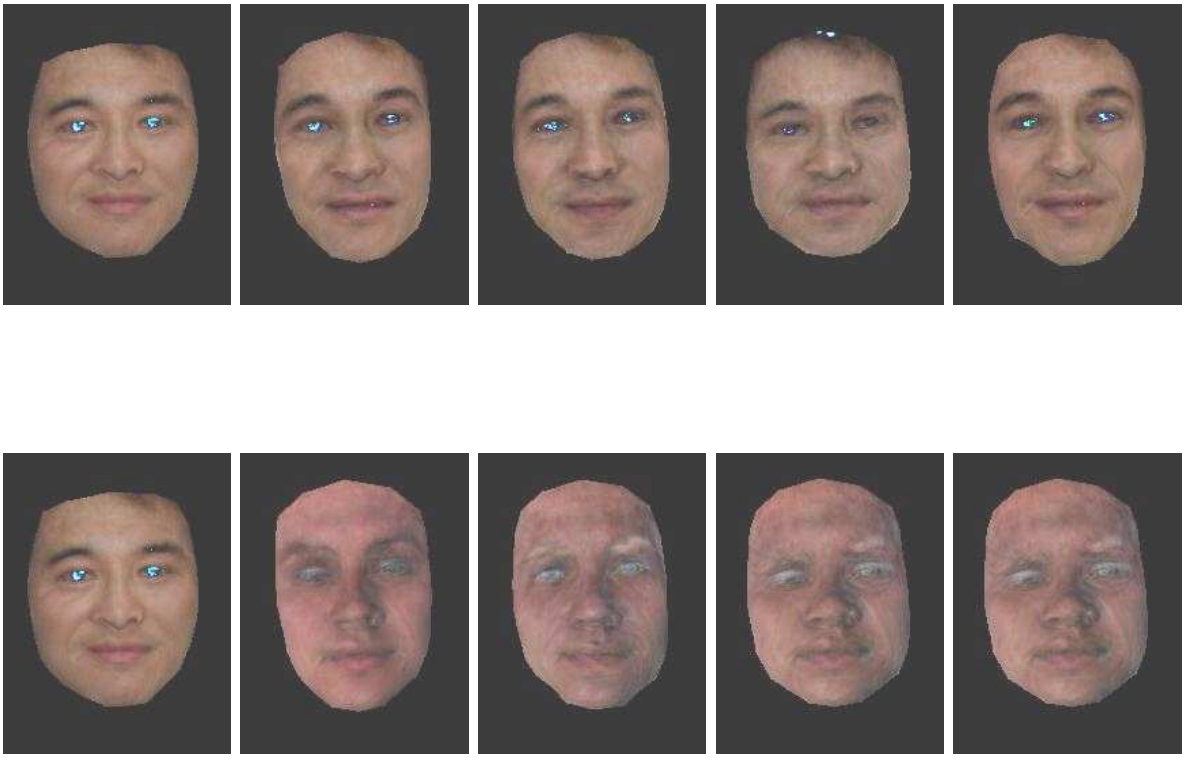

Figure 5.15: Top row: projected image used. Bottom row: constructed image used.

An original image and images from a linear kernel, 2nd polynomial kernel, 3rd polynomial kernel and Back-propagation Neural Network are used.

From quantitative measurement, SVM with linear kernel is better one. I think SVM with 3rd degree polynomial is better by looking at AAM images because a basic figure of face from 3rd degree polynomial kernel looks closer to the original. According to measurement,

worst one is Neural Network. Unlike SVMs' results, it looks like a bit feminine.

Vin Diesel AAM image

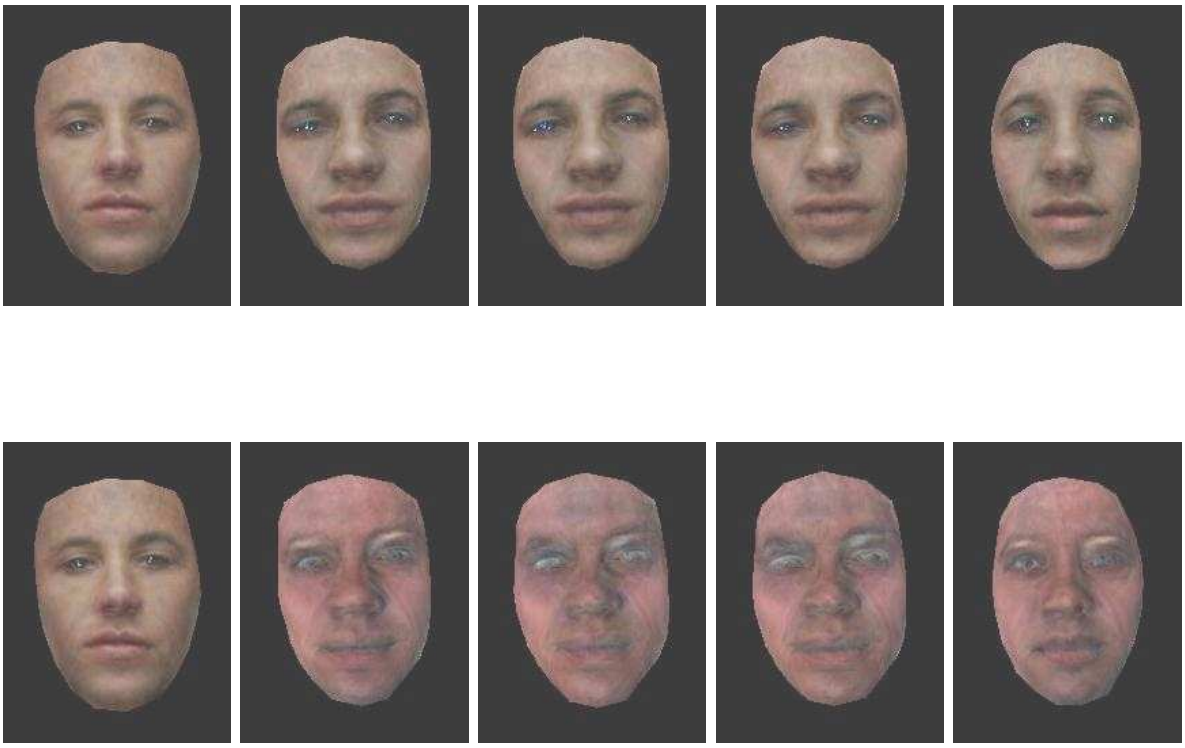

Figure 5.16: Top row: projected image used. Bottom row: constructed image used.

An original image and images from a linear kernel, 2nd polynomial kernel, 3rd polynomial kernel and Back-propagation Neural Network are used.

The results of AAM images surprise me because rendered result from Neural Network seems to have a kind of rounder face shape, but AAM image displays shape is an oval. According to quantitative measurement, Neural Network gives a better result although my initial expectation was 2nd degree polynomial kernel gives a better result.

# 6 DISCUSSION AND CONCLUSION

## 6.1 Discussion

Looking at final images, the shapes of tested faces actually appear well-formed for only having 100 training sample used. However, as I expected, textures are not so good due to lack of samples. From Table 1, all measurements are less than 1 unit apart. I use Maya software to render final images. So, these scores of measurement can be improved by manipulating rendering parameters and settings of Maya.

As I can see final images with constructed texture, the scores of these images are worse than final images with original image projected. In Table 2, the scores of most cases are more than 1 unit apart. Comparing to scores from Table 1, all scores goes up roughly 0.2. I am surprised by this because I had expected that their differences would be greater.

I take these final images and landmark them by using the AAM toolkit. Then I make images of these final images to see what AAM toolkit may construct the images of these final images. AAM images of Christopher Reeves, Jet Li and Vin Diesel still retain averaged texture although textures of 2nd degree and 3rd degree polynomial kernels are

bad.

As for the results of quantitative measurement, my expectation is SVM surpasses Neural Network in accuracy. The reason I think SVM gives better results is that SVM is currently one of the best classification methods [3] [17] [10]. According to table 1, it seems that Neural Network did outperform SVM. However, SVM with 2nd degree polynomial provides better generalization.

In order to improve accuracy, more samples are definitely needed. I would estimate that about 500 samples are needed to generate new textures. To show this point, I use AAM toolkit to generate same test faces using only 100 samples.

Figure 6.1: On top: AAM uses 609 samples. On bottom: AAM uses 100 samples.

Figure 6.1 shows that AAM image construction is not good if it uses only 100 samples although AAM toolkit does better job constructing textures compared to my results. Also, the shapes of these faces are different. Thus, the results of this work will improve significantly if more samples are used.

## 6.2 Limitation

Neural Network (NN) gives 3 best results out of 5 experiments, but it also gives 2 worst results. It seems that NN has ups and downs because it might fall into a local minima that is a main problem of

NN. Thus, the results will probably depend on the characteristic of new inputs.

In order to build a stronger connection between averaged face and sampled faces during correspondence training, input parameters are multiplied by $\pm 0.25$, $\pm 0.5$, and $\pm 0.75$ and reintroduced into the training set. Then the parameters of the averaged face are all added to make total of 801 training samples for NN and SVM. This was done in an attempt to argument the training set.

Using these 801 samples as training input, the NN takes 30 minutes to 40 minutes to do training. It may appear that the NN fails to meet the goal of generating parameters in real time; however, the advantages of NN are that NN will generate new parameters instantly once a training is done, and NN does not have to be trained every time unless new samples are added to database. As for SVM, each kernel takes less than 25 second to complete training. Unlike NN, SVM needs to be trained and needs to predict the result each time. Thus, generating a large number of 3D mesh using SVM can take more time than using NN, but this still faster than iterative methods in [7] [27].

Another limitation of this work is a lack of diversity in samples.

Robust 3D database are only now beginning to be collected, so such data is difficult to obtain. All methods in this paper have more difficulty to construct dark skin texture because more than 80% of samples are European descent white males. Because only 5% are females, constructing a 3D mesh of females using this dataset will give the result that looks more a male figure.

## 6.3 Conclusion

A purpose of establishing a correspondence between 3DMM parameters and 2D AAM parameters is to bypass iterative mapping processes that are used in the previous work [7] [27]. Although using Machine-Learning technique to map from 2D parameters to 3D parameters is suggested, no one has tried this technique. By introducing this technique, I also need to investigate if this unproven technique can give satisfactory results and can finish mapping process in less time than a previous work [7].

Constructing a 3D mesh from a single picture in a reasonable amount of time can be useful in many areas such as biometric fields and a 3D animation. Previous techniques require more time to construct a 3D mesh, so their usage will be limited to the situation which a time is not issue. However, by using Machine-Learning technique, a 3D mesh can

be constructed in a reasonable amount of time suitable for realtime applications such as a gaming and biometric identifications.

Table 1 shows distance from original image to constructed 3D mesh using Euclidean distance. All methods give the results that are less than 1 unit away from original image. These results are impressive because only 100 samples are used to get these results. As figure 6.24 shows, the results will significantly improve if more samples are used. A training time will increase if more samples are used. However, Machine-Learning technique is still faster than iterative methods.

To get more accurate results, more samples need to be added as described in discussion. However, there is one possibility to improve the results without adding more samples. The results of SVM depends on choosing right parameters for the problems [12]. By tweaking these parameters and finding right parameters, the accuracy may improve. Perhaps, SVM may take less time to train. Considering the quantitative results, Machine-Learning technique shows a potential to results and visual image quality, this machine-learning-based technique demonstrates potential as a method for 3D face generation from single 2D face images.

[1] "OpenCV Wiki". `http://opencv.willowgarage.com/wiki/`.

[2] Shihab Asfour, Arzu Onar, and Nandita Kaundinya. "Tool Breakage Detection Using Support Vector Machine Learning in a Milling Process". *International Journal of Machine Tools and Manufacture*, 45(3):241–249, March 2005.

[3] Laura Auria and Rousla A. Moro. "Support Vector Machines (SVM) as a Technique for Solvency Analysis". *Discussion Papers of DIW Berlin 811*, 2008.

[4] Autodesk. "Overview of FCheck", 2009. `http://download.autodesk.com/us/maya/2009help/index.html?url=FCheck_Start_FCheck.htm,topicNumber=d0e617547`.

[5] Volker Blanz. "Face Recognition Based on a 3D Morphable Model". *7th IEEE International Conference on Automatic Face and Gesture Recognition (FG'06)*, page 617, 2006.

[6] Volker Blanz, Curzio Basso, and Thomas Vetter. "Regularized 3D Morphable Models". *First IEEE International Workshop on Higher-Level Knowledge in 3D Madeling and Motion Analysis*, page 3, 2003.

[7] Volker Blanz and Thomas Vetter. "A Morphable Model For The Systhesis Of 3D Faces". *SIGGRAPH 99*, pages 187–194, 1999.

[8] Kevin Bowyer and Sudeep Sarkar. "USF 3D Face Databace". `http://marathon.csee.usf.edu/HumanID/`.

[9] Gary Bradski, Trevor Darrell, Irfan Essa, Jitendra Malik, Pietro Perona, Stan Sclaroff, and Carlo Tomasi. "Machine Learning Reference".

[10] Christopher J.C. Burges. "A Tutorial on Support Vector Machines for Pattern Recognition". *Data Mining and Knowledge Discovery, 2, 121-167*, 1998.

[11] Vinod Chandran, Chris McCool, Jamie Cook, and Sridha Sridharan. "Feature Modeling of PCA Difference Vectors for 2D and 3D Face Recognition". *2006 IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS'06)*, page 57, 2006.

[12] Chih-Chung Chang, Chih-Jen Lin, and Chih-Wei Hsu. "A Practical Guide for Support Vector Machines". Technical report, 2001. http://www.csie.ntu.edu.tw/ cjlin/libsvm.

[13] Vladimir Cherkassky and Yunqian Ma. "Practical Selection of SVM Parameters and Noise Estimation for SVM Regression". *Neural Networks*, 17(1):113–126, January 2004.

[14] Timothy F. Cootes. "Active Appearance Model Toolkit". `http://personalpages.manchester.ac.uk/staff/timothy.f.cootes/`.

[15] Timothy F. Cootes, Gareth J. Edwards, and Christopher J. Taylor. "Active Appearance Models". *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(6), June 2001.

[16] FileInfo. "RGB File Extension", 2007. `http://fileinfo.com/extension/rgb`.

[17] Steve R. Gunn. "Support Vector Machines for Classification and Regression ". Technical report, University of Southampton, May 1998.

[18] Vikramaditya Jakkula. "Tutorial on Support Vector Machine (SVM)". Technical report, Washington State University.

[19] Hsuan-Tien Lin and Chih-Jen Lin. "A Study on Sigmoid Kernels for SVM and the Training of non-PSD Kernels by SMO-type Methods". Technical report, National Taiwan University.

[20] Rasmus Elsborg Madsen, Lars Kai Hansen, and Ole Winther. "Singular Value Decomposition and Principal Component Analysis". Technical report, Technical University of Denmark, Febrauray 2004. http://www2.imm.dtu.dk/pubdb/views/edoc_download.php/4000/pdf/imm400.pdf.

[21] MathWorks. "Neural Network Toolbox". http://matlab.izmiran.ru/help/toolbox/nnet/backpr59.html.

[22] Masaaki Mochimaru, Kaori Yoshiki, and Hideo Saito. "Reconstruction of 3D Face Model from Single Shading Image Based on Anatomical Database". *18th International Conference on Pattern Recognition (ICPR'06)*, pages 4:350–353, 2006.

[23] Michael Negnevitsky. *"Artificial Intelligence"*. Pearson Education, second edition edition, 2005.

[24] Jim Peterson. "Numerical Analysis: Adventures in Frustration". http://www.ces.clemson.edu/~petersj/Agents/MatLabNA/index.html.

[25] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *"NUMERICAL RECIPIES The Art of Scientific Computing"*. Cambridge University Press, third edition edition, 2007.

[26] Stuart Russell and Peter Norvig. *"Artificial Intelligence A Modern Approach"*. Pearson Education, second edition edition, 2003.

[27] Jamie Sherrah, Nathan Faggian, and Andrew Paplinski. "Active Appearance Models for Automatic Fitting of 3D Morphable Models". *2006 IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS'06)*, page 90, 2006.

[28] Jonathan Shlens. "A Tutorial on Principal Components Analysis". Technical report, University of California, San Diego, 2005.

[29] Ashish Singhal and Dale E. Seborg. "Matching Patterns from Historical Data Using PCA and Distance Similarity Factors". *Proceeding of the American Control Conference*, pages 1759–1764, 2001.

[30] Lindsay I. Smith. "A Tutorial on Principal Components Analysis". Technical report, University of Otago, February 2002. `http://www.cs.otago.ac.nz/cosc453/student\_tutorials/principal\_components.pdf`.

[31] Bhavani Thuraisingham and Pallabi Parveen. "Face Recognition using Multiple Classifiers". *18th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'06)*, pages 179–186, 2006.

[32] Vladimir Naumovich Vapnik. *"The Nature of Statistical Theorey"*. Springer-Verlag, 1995.

[33] Thomas Vetter and Volker Blanz. "Face Recogniton Based on Fitting a 3D Morphable Model". *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 25:1063–1074, 2003.

[34] Wikipedia. "Raw Image Format". `http://en.wikipedia.org/wiki/Raw\_image\_format`.

```
In order to use, parameter files are required.
This processes support vector classification.

#include <OpenCV/OpenCV.h>
#include <iostream>
#include <fstream>
#include <string>
#include <sstream>
#include <cmath>
#include <time.h>

using namespace std;

int main (int argc, char * const argv[]) {

    cout << "atart\n";

string directory;
directory = "../../../params/";

string shape[100], texture[100];
string number, name;
CvMat *data = cvCreateMat(200, 100, CV_32F);
double element = 0;

/* read values from a file */
for (int i = 0; i < 100; i++) {

stringstream paramcount;
paramcount << i+1;


if (i < 9) {
name = "00" + paramcount.str();
```

```
}
if (i >= 9 && i < 99) {
name = "0" + paramcount.str();
}
if (i >= 99) {
name = paramcount.str();
}

shape[i] = directory + name + "s.txt";
texture[i] = directory + name + "t.txt";
}

for (int i = 0; i < 100; i++) {

fstream ins( texture[i].c_str() );

if (!ins) {
cout << "there is no such file as " << shape[i].c_str() << endl;
return 0;
}

for (int j = 0; j < 100; j++) {
ins >> element;
cvmSet(data, j, i, element);
}



ins.close();

}

element = 0;
for (int i = 0; i < 100; i++) {
fstream ins2( shape[i].c_str() );
```

```
if (!ins2) {
cout << "there is no such file as " << texture[i].c_str() << endl;
return 0;
}

for (int j = 0; j < 100; j++) {
ins2 >> element;
cvmSet(data, 100 + j, i, element);
}
ins2.close();
}

/* get mean and adjust data matrix */
CvMat *mean = cvCreateMat(data->rows, 1, CV_32F);
CvMat *adjusted = cvCreateMat(data->rows, data->cols, CV_32F);

float value = 0;
for (int i = 0; i < data->rows; i++) {
for (int j = 0; j < data->cols; j++) {
value =+ cvmGet(data, i, j)/data->cols;
}

cvmSet(mean, i, 0, value);
value = 0;
}


for (int i = 0; i < data->rows; i++) {
for (int j = 0; j < data->cols; j++) {
cvmSet(adjusted, i, j, cvmGet(data, i, j) - cvmGet(mean, i, 0));
}
}

/* calculate eigen values */
CvMat *eigenValue = cvCreateMat(data->rows, 1, CV_32F);
CvMat *eigenVector = cvCreateMat(data->rows, data->cols, CV_32F);
```

```
cout << "calc eigen\n";
cvSVD(adjusted, eigenValue, eigenVector, 0, 0);

// reduce eigenvector ( if necessary )
CvMat *eig_red =
     cvCreateMat(eigenVector->rows, eigenVector->cols, CV_32F);
CvMat *eig_trans =
     cvCreateMat(eig_red->cols, eig_red->rows, CV_32F);
for (int i = 0; i < eig_red->rows; i++) {
for (int j = 0; j < eig_red->cols; j++) {
cvmSet(eig_red, i, j, cvmGet(eigenVector, i, j));
}
}

/* get parameter */
CvMat *parameters =
     cvCreateMat(eig_trans->rows, adjusted->cols, CV_32F);


cvTranspose(eig_red, eig_trans);

cout << "mult\n";
cvMatMul(eig_trans, adjusted, parameters);

for (int i = 0; i < 3; i++) {
for (int j = 0; j < 5; j++) {
cout << cvmGet(parameters, i, j) << "\t";
}
cout << endl;
}

cout << "read aam param" << endl;

CvMat *aamParam = cvCreateMat(399, 801, CV_32F);
CvMat *aamTrans = cvCreateMat(801, 399, CV_32F);
```

```
string dir = "../../../aam_data399/";
string nameOfFile, num;

float elem = 0;

for (int i = 0; i < 801; i++) {
stringstream count;

count << i+1;
if ( i < 9 )
num = "00" + count.str();
if ( i >= 9 && i < 99 )
num = "0" + count.str();
if ( i >= 99 )
num = count.str();

name = dir + num + ".txt";

ifstream is(name.c_str());
if (!is) {cout << "no file" << endl; cout << name.c_str() << endl;}

for (int row = 0; row < 399; row++) {
is >> elem;
cvmSet(aamParam, row, i, elem);
elem;
}
is.close();
}
cout << "read is done " << endl;
cvTranspose(aamParam, aamTrans);

int column = 100;
CvMat *pcaParam = cvCreateMat(column, 801, CV_32F);
CvMat *pcaTrans = cvCreateMat(801, column, CV_32F);
```

```
string pcadir = "../../../combined801/";
string names, nums;
float el = 0;

for (int i = 0; i < 801; i++) {
stringstream counts;

counts << i + 1;
if( i < 9 )
nums = "00" + counts.str();
if( i >= 9 && i < 99 )
nums = "0" + counts.str();
if( i >= 99 )
nums = counts.str();

names = pcadir + nums + ".txt";
ifstream isn(names.c_str());
if( !isn ) {cout << "no file\n";}

for (int row = 0; row < pcaParam->rows; row++) {
isn >> el;
cvmSet(pcaParam, row, i, el);
}
isn.close();
}
cvTranspose(pcaParam, pcaTrans);

cout << "cvm\n";

CvSVMParams params;
params.svm_type = CvSVM::EPS_SVR;
params.kernel_type = CvSVM::POLY;
params.gamma = 1;//500;
params.nu = 0.9;
params.C = 25;
params.degree = 3;
```

```
params.coef0 = 12;
params.p = 100;//200;
params.term_crit.epsilon = 50;
params.term_crit.max_iter = 100000;

//CV_TERMCRIT_ITER|CV_TERMCRIT_EPS;
params.term_crit.type = CV_TERMCRIT_EPS;

CvSVM svm;

clock_t init, final;



cout << "train svm" << endl;

CvMat *sample_transpose = cvCreateMat(399, 1, CV_32F);
CvMat temps;
double answer[100] = {0};

double val = 0;

ifstream in("../../../all_params399/vin-picture.b_app");
if (!in)
cout << "no file" << endl;
for (int i = 0; i < 399; i++) {
in >> val;
cvmSet(sample_transpose, i, 0, val);
}
in.close();

cout << "sample value" << endl;
for (int i = 0; i < 5; i++) {
cout << cvmGet(sample_transpose, i, 0) << endl;
}
```

```
cout << "train" << endl;
init = clock();

for (int p = 0; p < column; p++) {

cvGetCol(pcaTrans, &temps, p);
svm.train(aamTrans, &temps, NULL, NULL, params);
answer[p] = svm.predict(sample_transpose);
}

cout << "training is done" << endl;

final = clock() - init;

/* export projected data */
CvMat *prediction = cvCreateMat(parameters->rows, 1, CV_32F);
CvMat *newData = cvCreateMat(data->rows, 1, CV_32F);


for (int i = 0; i < prediction->rows; i++) {
cvmSet(prediction, i, 0, answer[i]);
}

cvMatMul(eig_red, prediction, newData);

double np[200] = {0};

for (int i = 0; i < newData->rows; i++) {
np[i] = cvmGet(newData, i, 0) + cvmGet(mean, i, 0);
}

ofstream of1("../../../newobj/tex.txt");
ofstream of2("../../../newobj/shape.txt");

for (int i = 0; i < 100; i++) {
of1 << np[i] << endl;
```

```
of2 << np[i+100] << endl;
}
of1.close();
of2.close();



cout << "time: " << (double)final/((double)CLOCKS_PER_SEC) << endl;
//Automatically run other code after finishing SVM part
system("../../../CVtest/build/Debug/CVtest");
system("../../../ImageTest/build/Debug/ImageTest");

//opens MAYA
//system("open /Applications/Autodesk/maya2009/Maya.app");

cvReleaseMat(&data);
cvReleaseMat(&mean);
cvReleaseMat(&adjusted);
cvReleaseMat(&eigenValue);
cvReleaseMat(&eigenVector);
cvReleaseMat(&parameters);
cvReleaseMat(&aamTrans);
cvReleaseMat(&aamParam);
cvReleaseMat(&sample_transpose);
cvReleaseMat(&pcaParam);
cvReleaseMat(&pcaTrans);
cvReleaseMat(&eig_red);
cvReleaseMat(&eig_trans);

return 0;
}
```

```
This constructs 3D mesh from a parameter files given by a user.
A text file that contains material information and
     texture coordinates is needed to create obj file.

#include "commonLib.h"
#include "readInData.h"
#include "calculation.h"
#include <ctime>


using namespace std;

 * this is main method.
 * basically, this creates variables and
 * passes these variables to functions to process.
 */

int main (int argc, char * const argv[]) {


clock_t start = clock(); // start timer

cout << "construct 3D face" << endl;

/* declare necessay variables to read files. */
FILE *fp = NULL;
char buffer[255], filename[30];
int count = 0, numOfLines = 0, numOfFaces = 100;
double element = 0;

string dir = "../../../face/";
string param_dir = "../../../newobj/";
string root_dir = "../../../newobj/";
string name;

name = dir + files[0];
```

```
cout << name << endl;

/* reading file one time to get number of lines to read.
 * by doing this, I do not have to do hard coding.
 */

fp = fopen(name.c_str(), "r");
if (!fp){
cout << "no file";
}
else {
while (!feof(fp)){
fgets(buffer, 255, fp);
if ( strncmp("v ", buffer, 2) == 0 )
count++;
numOfLines++;
}
fclose(fp);
}

cout << "lines " << count << endl;

/* create variables that hold calculation data. */
CvMat *dataMat = cvCreateMat(3 * count, numOfFaces, CV_32F);
CvMat *adjust_data = cvCreateMat(3 * count, numOfFaces, CV_32F);
CvMat *average = cvCreateMat(3 * count, 1, CV_32F);
CvMat *evalue = cvCreateMat(numOfFaces, 1, CV_32F);
CvMat *evector = cvCreateMat(3 * count, numOfFaces, CV_32F);


/*
 * call read file method to read data.
 * this function takes source, number of lines to read,
 * and number of files.
 */
```

```
cout << "read data" << endl;
readDataFile(dataMat, count, numOfFaces);

/* calle adjust function.
 * this function takes source and
 * calculates average face that saves to average.
 * also calculates values that are subtracted average
 * form source and saves to adjust_data.
 */

cout << "adjusted" << endl;
adjusted(dataMat, adjust_data, average, numOfFaces);

/* call OpenCV function that calculates eigenvecors and eigenvalues.
cout << "calculate eigenvector" << endl;
cvSVD(adjust_data, evalue, evector, 0, 0);

clock_t middle = clock(); // read time

/* calculate processing time so far. */
cout << "Time after calculating eigenvector: ";
cout << (middle - start) / CLOCKS_PER_SEC << " seconds" << endl;

/* printout for debugging purpose. */
cout << "data" << endl;
for (int i = 0; i < 10; i++) {
for (int j = 0 ; j < 5; j++) {
cout << cvmGet(dataMat, i, j) << " ";
}
cout << endl;
}

cout << "eigenvectors" << endl;
for (int i = 0; i < 10; i++) {
for (int j = 0; j < 5; j++) {
```

```
cout << cvmGet(evector, i, j) << " ";
}
cout << endl;
}

/*
 * create a matrix that holds reduced eigenvectors and
 * call function to reduce eigenvectors.
 */

CvMat *reduced =
     cvCreateMat(evector->rows, (evector->cols), CV_32F);
cout <<"eigenvector reducing" << endl;
reduceEig(evector, reduced);

/*
* declare necessay variables that holds new data such as
* new parameters that is given by a user.
*/
CvMat *reduce_data = cvCreateMat(adjust_data->rows, 1, CV_32F);
CvMat *newParam = cvCreateMat(reduced->cols, 1, CV_32F);
CvMat *reduce_and_avg = cvCreateMat(average->rows, 1, CV_32F);

/*
 * this process reads new parameters that a user specified and
 * makes obj files.
 * this process continues until a user types 'quit'.
 */

while (strncmp(filename, "quit", 4) != 0) {

cout << "Enter new parameter file name ";
cout <<  "without extension. ('quit' to exit )" << endl;
cin >> filename;

if (strncmp(filename, "quit", 4) != 0) {
```

```
name = param_dir + filename + ".txt";
ifstream ins(name.c_str());

if (ins){

for (int i = 0; i < newParam->rows; i++) {
ins >> element;
cvmSet(newParam, i, 0, element);
element = 0;
}
ins.close();

cvMatMul(reduced, newParam, reduce_data);

for (int i = 0; i < reduce_and_avg->rows; i++) {
cvmSet(reduce_and_avg, i, 0,
     cvmGet(average, i, 0) + cvmGet(reduce_data, i, 0));
}

cout << "create obj file" << endl;
name = root_dir + filename + ".obj";

ofstream os(name.c_str());
for (int i = 0; i < count; i++) {
os << "v " << cvmGet(reduce_and_avg, i, 0) << " "
     << cvmGet(reduce_and_avg, i+count, 0) << " "
     << cvmGet(reduce_and_avg, i+(2*count), 0) << endl;
}

ifstream fin("../../../obj.txt");
while (fin.good()) {
fin.getline(buffer, 255);
os << buffer << endl;
}
fin.close();
```

```
os.close();
}
else {
cout << "file not found" << endl;
cout << "Enter file name. e.g. face1 instead face1.txt" << endl;
}
}
}

// release memory
cvReleaseMat(&adjust_data);
cvReleaseMat(&evalue);
cvReleaseMat(&evector);
cvReleaseMat(&dataMat);
cvReleaseMat(&average);
cvReleaseMat(&reduce_data);
cvReleaseMat(&newParam);
cvReleaseMat(&reduce_and_avg);
cvReleaseMat(&reduced);

clock_t end = clock(); // reads end time

/* calculate processign time, */
cout << "time: ";
cout << (end - start) / CLOCKS_PER_SEC << " seconds" << endl;

return 0;
}
```

```
This just reads file name.

#include "name_strings.h"

std::string files[100] =
{"train_02463_1.obj","train_03500_1.obj","train_03501_4.obj",
"train_03511_1.obj","train_03513_1.obj","train_03515_1.obj",
"train_03516_1.obj","train_03516_4.obj","train_03521_1.obj",
"train_03521_4.obj","train_03523_4.obj","train_03524_4.obj",
"train_03527_1.obj","train_03532_4.obj","train_03543_1.obj",
"train_03544_1.obj","train_03545_1.obj","train_03552_1.obj",
"train_03553_1.obj","train_03554_1.obj","train_03555_1.obj",
"train_03556_1.obj","train_03557_1.obj","train_03559_1.obj",
"train_03562_1.obj","train_03563_1.obj","train_03569_1.obj",
"train_03577_1.obj","train_03585_1.obj","train_03586_1.obj",
"train_03588_1.obj","train_03589_1.obj","train_03591_1.obj",
"train_03594_1.obj","train_03596_1.obj","train_03597_1.obj",
"train_03600_1.obj","train_03603_1.obj","train_03605_1.obj",
"train_03605_4.obj","train_03608_4.obj","train_03618_1.obj",
"train_03621_4.obj","train_03629_4.obj","train_03634_1.obj",
"train_03634_4.obj","train_03636_1.obj","train_03643_4.obj",
"train_03647_1.obj","train_03653_1.obj","train_03655_4.obj",
"train_03657_1.obj","train_03659_4.obj","train_03669_1.obj",
"train_03673_1.obj","train_03676_1.obj","train_03680_1.obj",
"train_03684_1.obj","train_03691_1.obj","train_03693_1.obj",
"train_03694_4.obj","train_03699_1.obj","train_03704_1.obj",
"train_03705_1.obj","train_03706_1.obj","train_03707_1.obj",
"train_03707_4.obj","train_03708_1.obj","train_03709_1.obj",
"train_03710_1.obj","train_03711_1.obj","train_03714_1.obj",
"train_03715_1.obj","train_03717_1.obj","train_03718_1.obj",
"train_03720_1.obj","train_03721_1.obj","train_03722_1.obj",
"train_03727_1.obj","train_03729_1.obj","train_03731_1.obj",
"train_03740_1.obj","train_03741_4.obj","train_03742_1.obj",
"train_03744_1.obj","train_03746_1.obj","train_03751_1.obj",
"train_03752_1.obj","train_03754_1.obj","train_03767_4.obj",
"train_03771_4.obj","train_03772_4.obj","train_03773_4.obj",
```

"train_03775_4.obj","train_03776_4.obj","train_03777_4.obj",
"train_03782_4.obj","train_03784_4.obj","train_03786_4.obj",
"train_03793_4.obj"};

```
This reads actual data according to file names.

#include "readInData.h"
#include "commonLib.h"

void printElement(){
cout << "test" << endl;

for ( int i = 0; i < 100; i++ )
cout << "file " << i + 1 << ": " << files[i] << endl;
}

void readDataFile(CvMat *data, int numOfElem, int nface){
float x = 0, y = 0, z = 0;
int n = 0;


FILE *fp = NULL;
char buffer[1024];

//fp = fopen("/Users/sh0110/Documents/testfile.txt", "r");
string str = "/Users/sh0110/face/";
string name;

for (int i = 0; i < nface; i++) {

name = str + files[i];
//cout << i << " " << name << endl;
fp = fopen(name.c_str(), "r");

n = 0;

while (!feof(fp)){
fgets(buffer, 1024, fp);
//cout << "ok" << endl;
if ( strncmp("v ", buffer, 2) == 0 ){
```

```
sscanf((buffer + 1), "%f%f%f", &x, &y, &z);
cvmSet(data, n, i, x);
cvmSet(data, n + numOfElem, i, y);
cvmSet(data, n + (2 * numOfElem), i, z);
n++;

}
}
fclose(fp);
}

}
```

```
This calculates average face and data matrix
     that is subtracted by average face.

#include "calculation.h"

void adjusted(CvMat *original, CvMat *adjust,
               CvMat *avg, int numOfFace){
double value = 0, val = 0;

cout << "rows: " << original->rows << " cols: ";
cout << original->cols << endl;

for ( int i = 0; i < original->rows; i++) {
for (int j = 0; j < original->cols; j++) {
value += cvmGet(original, i, j);
}
cvmSet(avg, i, 0, (value/numOfFace));
value = 0;
}

for (int r = 0; r < original->rows; r++) {
for (int c = 0; c < original->cols; c++) {
val = cvmGet(original, r, c) - cvmGet(avg, r, 0);
cvmSet(adjust, r, c, val);
val = 0;
}
}
}

void reduceEig(CvMat *source, CvMat *dest){

for (int i = 0; i < source->rows; i++) {

for (int j = 0; j < dest->cols; j++) {
cvmSet(dest, i, j, cvmGet(source, i, j));
}
```

```
}
}
```

```
These are necessary headers.

Header for common libraries.
#ifndef common_libraries_
#define common_libraries_

#include <OpenCV/OpenCV.h>

#include <iostream>
#include <fstream>
#include <sstream>

using namespace std;

#endif

Header for calcluating average face and and other.
#include "commonLib.h"

extern void adjusted(CvMat *original,
                        CvMat *adjust, CvMat *avg, int numOfFace);
extern void reduceEig(CvMat *source, CvMat *dest);

Header for name.
#ifndef name_strings_
#define name_strings_

#include "commonLib.h"

extern string files[100];

#endif

Header for reading data.
#include "name_strings.h"
```

```
extern void printElement();
extern void readDataFile(CvMat *data, int numOfElem, int nface);

\edn{verbatim}
\newpage

\begin{verbatim}
An image processing main file.

#include "common.h"
#include "svmfile.h"
#include "texture.h"
#include "neuralnet.h"
#include <time.h>

using namespace std;
int main (int argc, char * const argv[]) {

clock_t start = clock();

/* image code */
calcTexture();

clock_t end = clock();

cout << "time elapsed: ";
cout << (end - start) / CLOCKS_PER_SEC << "seconds" << endl;
    return 0;
}
```

```
This is work file.
#include "texture.h"
#include "common.h"
#include <ctime>
#include <cstdlib>
#include <string>
#include <sstream>

using namespace std;

void calcTexture(){

string nameOfDir = "../../../texture/";
string nameOfFile, number;

IplImage **img = 0;
IplImage *avgimg = 0;

int face = 100;

CvMat *imagePrams = cvCreateMat(10, 9, CV_32FC1);

img = (IplImage** ) cvAlloc(sizeof(IplImage*) * face);

for (int i = 0; i < face; i++) {
stringstream counts;
counts << i+1;

number = counts.str();

nameOfFile = nameOfDir + number + ".jpg";

img[i] = cvLoadImage(nameOfFile.c_str(), 1);

}
```

```
IplImage *sample;
sample = cvCreateImage(cvGetSize(img[0]),
                  img[0]->depth, img[0]->nChannels);

// image color: converted bgr to hsv.
// opencv uses bgr format when images are loaded.

/*IplImage *hsv_img[100];
for (int i = 0; i < face; i++) {
hsv_img[i] = cvCreateImage(cvGetSize(img[0]),
                        img[0]->depth, img[0]->nChannels);
}


cout << "start\n";
//CvScalar s1, s2;

for ( int i = 0; i < face; i++ ){

cvCvtColor(img[i], hsv_img[i], CV_BGR2HSV);
}*/

cout << "image size of sample: " << sample->imageSize;
cout << " width step " << sample->widthStep << endl;

/*create data matrix and calculate average*/

CvMat *datamat = cvCreateMat(img[0]->imageSize, face, CV_32F);
CvMat *adjusted = cvCreateMat(datamat->rows, datamat->cols, CV_32F);


int p = 0;
for (int num = 0; num < sample->imageSize; num++) {
for (int i = 0; i < face; i++) {
p = p + (uchar)(img[i]->imageData[num]);
```

```
cvmSet(datamat, num, i, img[i]->imageData[num]);
//p = p + (uchar)(hsv_img[i]->imageData[num]);
//cvmSet(datamat, num, i, hsv_img[i]->imageData[num]);
}

sample->imageData[num] = (p/face);
p = 0;
}

cvSaveImage("../../../newobj/avg_test.jpg", sample);
cout << "here?\n";
CvScalar rgb, hsv;
rgb = cvGet2D(img[0], 173, 148);
//hsv = cvGet2D(hsv_img[0], 173, 148);


/*cvNamedWindow("test", CV_WINDOW_AUTOSIZE);
cvShowImage("test", sample);
cvWaitKey(0);*/

cout << "number of channels: " << sample->nChannels << endl;
cout << "image width: " << sample->width << endl;
cout << "image height: " << sample->height << endl;
cout << "image depth: " << sample->depth << endl;

cout << "converted" << endl;

CvMat *average = cvCreateMat(sample->imageSize, 1, CV_32F);

for (int i = 0; i < sample->imageSize; i++) {
cvmSet(average, i, 0, (uchar)sample->imageData[i]);
}

for (int i = 0; i < adjusted->rows; i++) {
for (int j = 0; j < adjusted->cols; j++) {
cvmSet(adjusted, i, j,
```

```
                cvmGet(datamat, i, j) - cvmGet(average, i, 0));
}
}

for (int i = 0; i < face; i++) {
cvReleaseImage(&img[i]);
//cvReleaseImage(&hsv_img[i]);
}

CvMat *evector = cvCreateMat(adjusted->rows, adjusted->cols, CV_32F);
CvMat *evalue = cvCreateMat(adjusted->cols, 1, CV_32F);

cout << "calculate eigenvector" << endl;

cvSVD(adjusted, evalue, evector, 0, 0);

CvMat *reduced = cvCreateMat(evector->rows, evector->cols, CV_32F);

for (int i = 0; i < reduced->rows; i++) {
for (int j = 0; j < reduced->cols; j++) {
cvmSet(reduced, i, j, cvmGet(evector, i, j));
}
}

CvMat *transeigen =
     cvCreateMat(reduced->cols, reduced->rows, CV_32F);
CvMat *params =
     cvCreateMat(transeigen->rows, adjusted->cols, CV_32F);

cout << "eigen transpose" << endl;
cvTranspose(reduced, transeigen);

cout << "get params" << endl;
cvMatMul(transeigen, adjusted, params);

cout << "params" << endl;
```

```
cout << "reduced row: " << reduced->rows;
cout << " reduced col: " << reduced->cols << endl;

CvMat *im = cvCreateMat(adjusted->rows, 1, CV_32F);
CvMat *np = cvCreateMat(reduced->cols, 1, CV_32F);

cout << "\nim: " << im->rows << " np: " << np->rows << endl;
/*choose param*/
cvGetCol(params, np, 20);


IplImage *constructed;
constructed = cvCreateImage(cvGetSize(sample),
                        IPL_DEPTH_8U, sample->nChannels);
IplImage *rgb_form;
rgb_form = cvCreateImage(cvGetSize(sample),
                    IPL_DEPTH_8U, sample->nChannels);

int values = 0;
double coef = 1;

string param_dir = "../../../newobj/";
string name;
char filename[30];
double element = 0;

while ( strncmp(filename, "quit", 4) != 0) {

cout << "enter file name. 'quit' to exit." << endl;
cin >> filename;

if ( strncmp(filename, "quit", 4) != 0){

name = param_dir + filename + ".txt";
```

```
ifstream ins(name.c_str());

if (ins){
for (int i = 0; i < np->rows; i++) {
ins >> element;
cvmSet(np, i, 0, element);
element = 0;
}
ins.close();
}

cout << "mult" << endl;
cvMatMul(reduced, np, im);


cout << "create image" << endl;
for (int i = 0; i < im->rows; i++) {
values = ((uchar)cvmGet(im, i, 0) + (uchar)cvmGet(average, i, 0));

constructed->imageData[i] = (uchar)values;
}


name = param_dir + filename +".jpg";
cout << name << endl;


//cvCvtColor(constructed, rgb_form, CV_HSV2BGR);
cvSaveImage(name.c_str(), constructed);
//cvSaveImage(name.c_str(), rgb_form);
}
}

IplImage *etc;
etc = cvCreateImage(cvGetSize(sample),
          sample->depth, sample->nChannels);
```

```
for (int i = 0; i < im->rows; i++) {
etc->imageData[i] = cvmGet(reduced, i, 0) * 1000;
}

/*for (int i = 0; i < im->rows; i++) {
cout << "test " << cvmGet(reduced, i, 0) << endl;
}*/

cvNamedWindow("test", CV_WINDOW_AUTOSIZE);
cvShowImage("test", constructed);
cvWaitKey(0);

cvReleaseImage(&etc);
cvReleaseMat(&transeigen);
cvReleaseMat(&params);
cvReleaseMat(&im);
cvReleaseMat(&np);
cvReleaseMat(&reduced);
cvReleaseImage(&avgimg);
cvReleaseMat(&imagePrams);
cvReleaseImage(&sample);
cvReleaseMat(&average);
cvReleaseMat(&datamat);
cvReleaseMat(&adjusted);
cvReleaseMat(&evalue);
cvReleaseMat(&evector);
cvReleaseImage(&constructed);
cvReleaseImage(&rgb_form);

}
```

```
Header file.

#include <OpenCV/OpenCV.h>
#include <OpenCV/highgui.h>
#include <OpenCV/ml.h>
#include <OpenCV/cvaux.h>
#include <OpenCV/cv.h>
#include <iostream>
#include <fstream>
#include <string>
#include <sstream>
#include <cmath>

extern void calcTexture();
```