NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

SCHOOL OF SCIENCE

DEPARTMENT OF INFORMATICS AND TELECOMMUNICATION

BSc THESIS

Assessing the URxD 3D Face Recognition Algorithm on Synthetic Facial Data

Panteleimon D. Kanellis

Supervisors: Theoharis Theoharis, Professor
Antonios Danelakis, Dr

ATHENS
OCTOBER 2020

ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Αξιολόγηση του 3Δ Αλγορίθμου Αναγνώρισης Προσώπων URxD σε Συνθετικά Δεδομένα Προσώπου

Παντελεήμων Δ. Κανέλλης

Επιβλέποντες: Θεοχάρης Θεοχάρης, Καθηγητής
Αντώνιος Δανελάκης, Δρ

ΑΘΗΝΑ
ΟΚΤΩΒΡΙΟΣ 2020

BSc THESIS

Assessing the URxD 3D Face Recognition Algorithm on Synthetic Facial Data

Panteleimon D.Kanellis

S.N.: 1115201600055

Supervisors: Theoharis Theoharis, Professor
Antonios Danelakis, Dr

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Αξιολόγηση του 3Δ Αλγορίθμου Αναγνώρισης Προσώπων URxD σε Συνθετικά Δεδομένα Προσώπου

Παντελεήμων Δ. Κανέλλης

Α.Μ.: 1115201600055

Επιβλέποντες: Θεοχάρης Θεοχάρης, Καθηγητής
Αντώνιος Δανελάκης, Δρ

ABSTRACT

Real-time face recognition on real faces has been an extensive research topic in computer science in the last three decades. One of the most successful face recognition techniques, that has dominated performance evaluations for almost 15 years, is URxD. This thesis examines the performance of URxD in computer generated faces, and compares the results with real human faces. More specifically, a 3D database containing 100 synthetic faces is generated. Each face has a neutral and a random expression with random intensity. The source code of URxD is recompiled and edited in order to run in modern machines. Later on, the synthetic 3D database is used as input for URxD in order to compute the success rate, that is how many faces were recognised correctly. The result of URxD is considered successful for one face, if the program correlates the neutral expression of the face with the random expression of the same face. Finally, URxD was tested on 100 real faces, solely for comparison purposes.

SUBJECT AREA: Face Recognition

KEYWORDS: synthetic faces, 3D database, facial expressions,
face recognition algorithm, recognition rate

ΠΕΡΙΛΗΨΗ

Η αναγνώριση ανθρωπίνων προσώπων σε πραγματικό χρόνο έχει αποτελέσει ένα εκτεταμένο θέμα έρευνας στην επιστήμη των υπολογιστών τις τελευταίες τρεις δεκαετίες. Μια από τις πιο επιτυχημένες τεχνικές αναγνώρισης προσώπων που έχει κυριεύσει τις επιδόσεις αξιολόγησης για περίπου 15 χρόνια, είναι το URxD. Η συγκεκριμένη πτυχιακή εξετάζει την απόδοση του URxD σε πρόσωπα παραγόμενα από υπολογιστή, και συγκρίνει τα αποτελέσματα με πραγματικά ανθρώπινα πρόσωπα. Πιο συγκεκριμένα, παράγεται μια 3Δ βάση δεδομένων, η οποία περιέχει 100 συνθετικά πρόσωπα. Κάθε πρόσωπο έχει μια ουδέτερη και μια τυχαία έκφραση με τυχαία ένταση. Ο πηγαίος κώδικας του URxD μεταγλωττίστηκε εκ νέου και τροποποιήθηκε, έτσι ώστε να μπορεί να τρέξει σε μοντέρνα μηχανήματα. Στην συνέχεια, η συνθετική 3Δ βάση δεδομένων χρησιμοποιείται σαν είσοδο για το URxD με σκοπό να υπολογιστεί το ποσοστό επιτυχίας, δηλαδή πόσα πρόσωπα αναγνωρίστηκαν σωστά. Ένα αποτέλεσμα του URxD θεωρείται επιτυχές για ένα πρόσωπο, αν το πρόγραμμα συσχετίζει την ουδέτερη έκφραση του προσώπου με την τυχαία έκφραση του ίδιου προσώπου. Τέλος, το URxD ελέγθηκε σε 100 πραγματικά πρόσωπα, για σκοπούς σύγκρισης και μόνο.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Αναγνώριση Προσώπων

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: συνθετικά πρόσωπα, 3Δ βάση δεδομένων, εκφράσεις προσώπων, αλγόριθμος αναγνώρισης προσώπων, ποσοστό αναγνώρισης

CONTENTS

1. INTRODUCTION

Face recognition is an important task that solves many real life problems. From criminal identification to identity verification, a good face recognition algorithm must be able to confront all these problems. This thesis challenges the ability of URxD to produce stable results for a synthetic database. The following questions are posed:

1. Are there any criteria to choose among the different synthetic face generators? How are these random faces generated?
2. Since randomly computer generated faces might be strongly related (some faces might look alike), can the algorithm score a high recognition rate?
3. Is there a guarantee that the algorithm will produce satisfying results for any 3D database, even for faces that ”don’t exist”?
4. Are there any methods that could potentially improve the algorithm’s success rate for synthetic faces?

Chapter 3 answers the first part of the first question, by comparing two face reconstruction programs. The usability of the programs, the quality of the generated faces, and the variety of facial expressions are three of the most important choosing factors. The second part of the first question is answered in chapter 4, where the workflow of the chosen reconstruction program is thoroughly analysed in a top level approach. For a more technical description of the random face generator, see Appendix. The questions 3 and 4 are examined in chapter 5, where URxD is tested on the synthetic 3D database and the performance results are extracted. Since the random face generator can produce some faces with slightly exaggerated features (with a certain probability), the 3D database serves as a stress test for URxD. Finally, chapter 6 examines the last question, by introducing the obstacles that were faced by URxD, and proposing a few techniques that might improve the performance of the algorithm.

2. RELATED WORK & BACKGROUND

Blanz et al. [1], introduced the 3D Morphable Model (3DMM) concept. Assuming there is full correspondence between all the faces of the database, the algorithm creates four segments for each face as seen in Figure 2.1 . Every other face and expression can be generated as a linear combination of these four segments. One year later, the same authors [2] proposed a method of face recognition using the 3DMM, where a 2D image (either frontal or side view, with different poses and illumination) is used to compute the 3D shape and texture of the face. A 3DMM is fitted in the 2D image by combining and altering facial features of an existing 3D database. After creating a 3D representation of each face, the algorithm compares two faces by taking into account the coefficients obtained from images of the same face.
Georgios Passalis et al.[3] introduced a 3D face recognition method, using the Annotated Face Model (AFM), which is a 3D geometry with distinctly marked facial features, as seen in Figure 2.2. For example, the forehead has a different colour (annotated area) than the mouth or the eyes.
URxD operates in two phases: Enrollment and Authentication. The Enrollment stage described in Figure 2.3, consists of 4 steps:

1. **Acquisition :** The data are collected in point cloud form, derived by laser scans. The point cloud representation (also referred as raw data), is transformed into a 3D polygonal representation through a Preprocessing pipeline.

2. **Alignment :** The 3D data are aligned into a unified coordinate system, so that metadata can be extracted in the next steps.

3. **Geometry Image Analysis :** The AFM is created once and it is used for the fitting process of the face.

4. **Fitting :** The fitting process deforms the shape of the AFM so that it can wrap around the 3D face. Later, the geometry image of the fitted model is derived and used as a unique signature for the face.

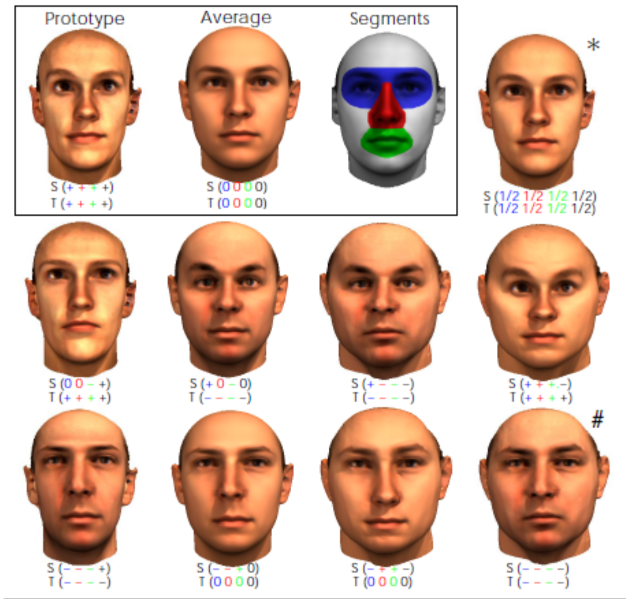


Figure 2.1: The prototype and the average face are used to generate new faces. The amount of deformation for each segment can influence the end result.[1]

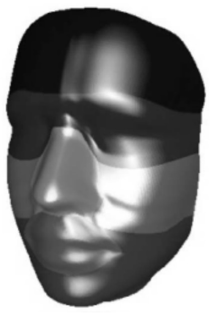


Figure 2.2: The Annotated Face Model. Each facial feature is marked as a distinct area. [3]

The Authentication phase produces the recognition rates. Using the signatures from the Enrollment phase, the algorithm computes distances from one face to another, creating a distance matrix. This matrix (often referred as similarity matrix), is processed by the algorithm and recognition results are derived in Cumulative Matching Curve (CMC) or Receiver Operating Curve (ROC) form. The URxD project was tested on the Full FRGC v2 Database containing real faces, and it scored over 97% as seen in Figure 2.4, the biggest score at the time.

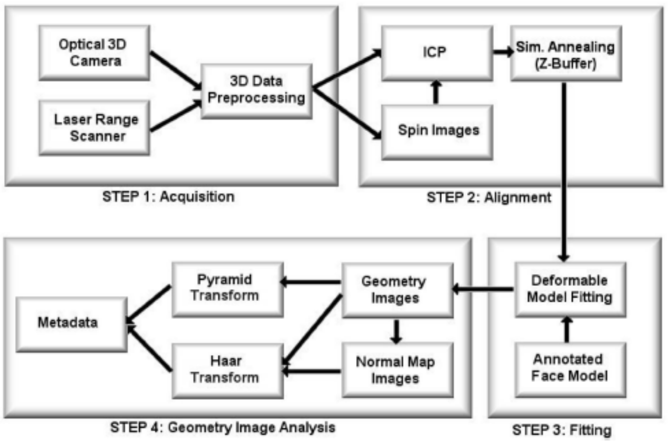


Figure 2.3: Enrollment phase of the URxD workflow[3]

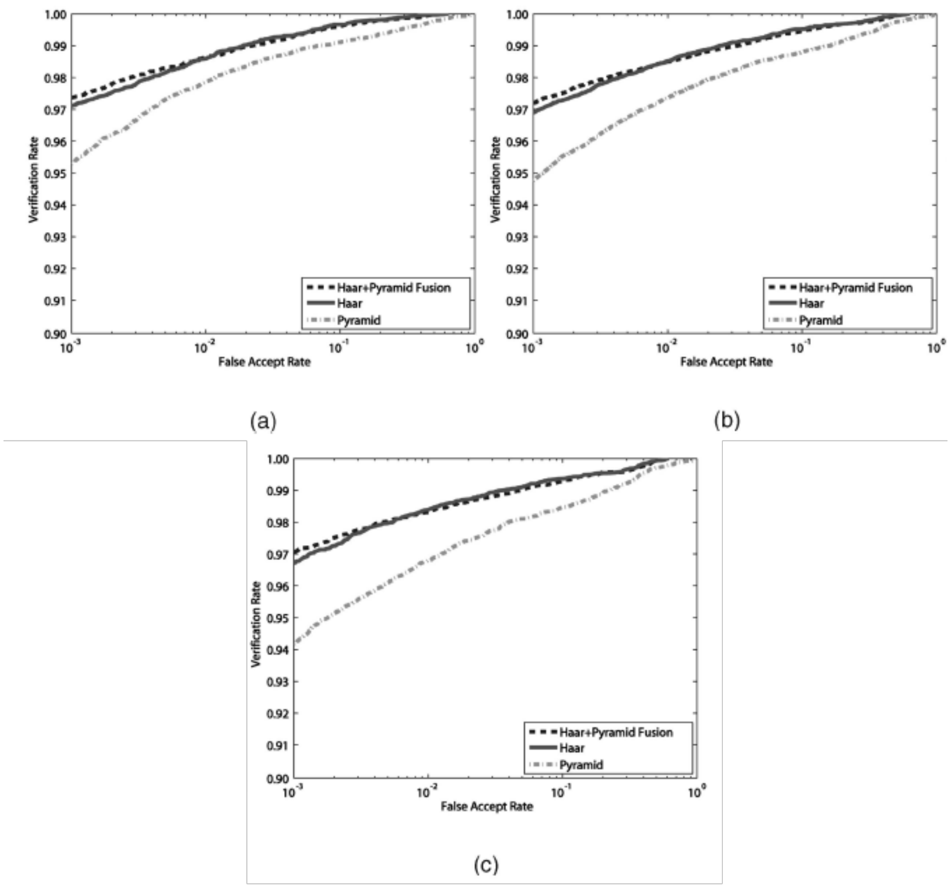


Figure 2.4: URxD performance on the FRGC v2 Database.[3]
(a) ROC I, (b) ROC II, and (c) ROC III.

3. SYNTHETIC 3D FACE GENERATION AND RECONSTRUCTION SYSTEMS

This chapter introduces two of the most prevalent face reconstruction and synthetic face generation systems : EOS and FaceGen. The differences between these two technologies are thoroughly examined, in order to choose the technology that produces the best results and is the easiest to use. The categories in which EOS and FaceGen will be compared, are the interface (how user friendly the API is), the inclusion or not of facial expressions, the number of landmarks, the quality of both the texture and the .obj , and the different input parameters. In order to understand the differences, let us study the workflow for each technology first.

3.1. EOS Worklflow

EOS[4] is a library written in C++11/14, offering basic functionalities such as face reconstruction from a 2D image using a 3DMM and camera manipulation. The usage of EOS requires additional dependencies before a face can be generated. In Windows systems, these dependencies can be installed via a package downloader called vcpkg, while Linux users can simply install the additional programs using the apt install command. The Appendix describes the complete installation and usage of EOS step by step.
Example : Suppose we have the following image in Figure 3.1.
Implementing the workflow, EOS will give the following result in Figure 3.2
A few observations about the derived results are listed below :

- EOS does *not* provide facial expressions. The program can be combined with 4dface, a software made by the same creator of EOS, in order to use blendshapes.
- Almost all test images had texture streching and incomplete texture maps (see Figure 3.3)
- The total amount of polygons for each face is 6,736. This means that EOS produces low resolution facial meshes and subsequently the facial expressions will be less accurate.

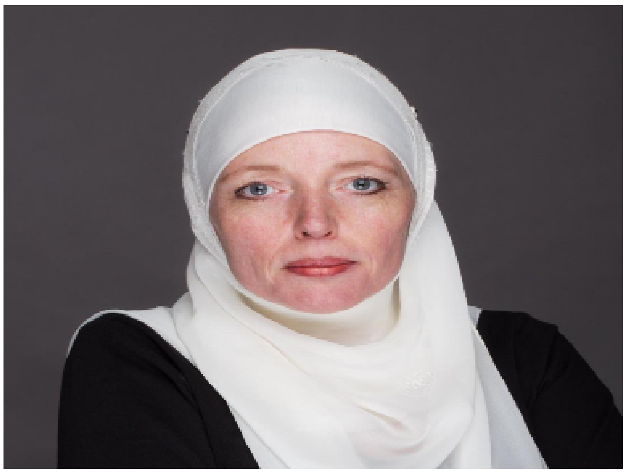


Figure 3.1: Face image reference

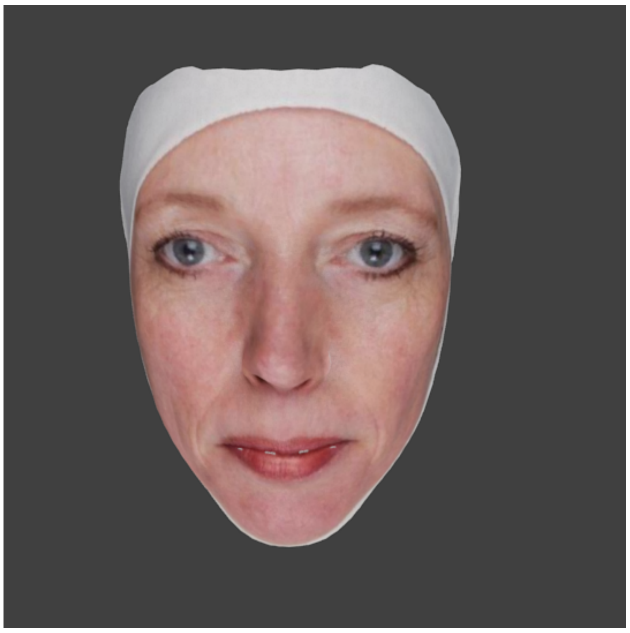


Figure 3.2: 3D Face reconstructed by EOS

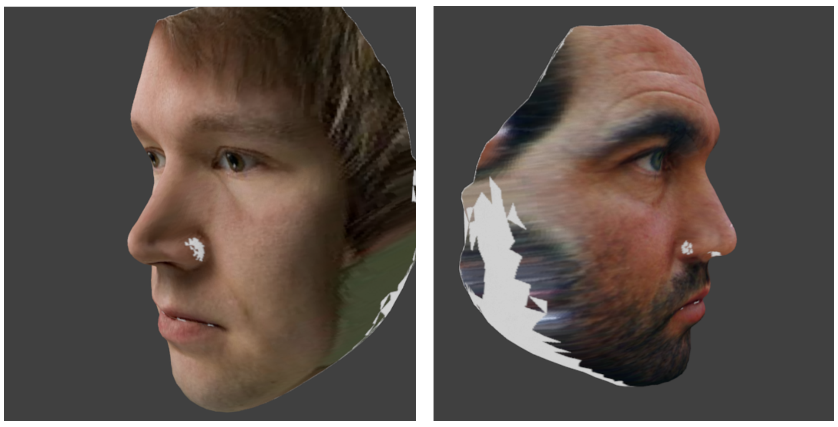


Figure 3.3: EOS produces texture stretching and incomplete texture maps.

3.2. FaceGen workflow

There are two versions of FaceGen [5], each with different functionalities. The Demo version has a friendlier interface (sliders control different parameters of the face in real time as seen in Figure 3.4), but it has less functionalities than the Full version. The Full version consists of an SDK that extends the demo version, by allowing the user to select the export format (obj,fbx,3ds,etc.) , create renders of 3D faces, batch create randomly generated faces and many more. For now, we will present the workflow for the demo version. The correspondent workflow for the full version will be examined in the next chapter. The core ideas remain the same for the two versions.

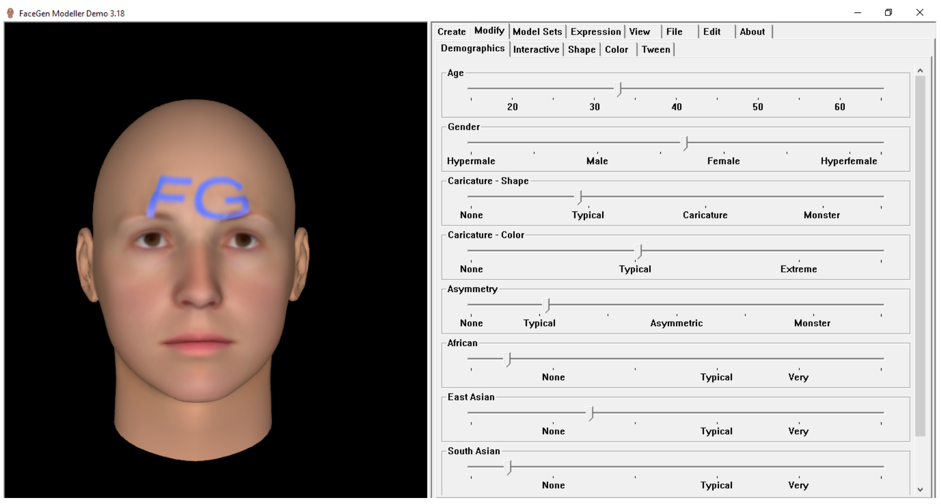


Figure 3.4: The interface of the Demo version of FaceGen.

The setup is much more simple this time. FaceGen requires no third party software or usage of VS. The demo version can be installed with a regular installer, while the Full version comes as a zipped format that can be extracted with any suitable software (WinZip for example). The user can provide up to three images to reconstruct a face (one frontal and two optional side images). The frontal image must be passport type (neutral facial expression, direct light). It is recommended that the input image has as high resolution as possible (at least 500 pixels for the height). The higher the resolution, the better the results.

Once an image (or images) is provided, the user is prompted to manually place the landmarks. The number of landmarks is 11 minimum (only one image as seen in Figure 3.5), and 29 maximum (three images). Typically, the more landmarks, the more accurate the final result will be.

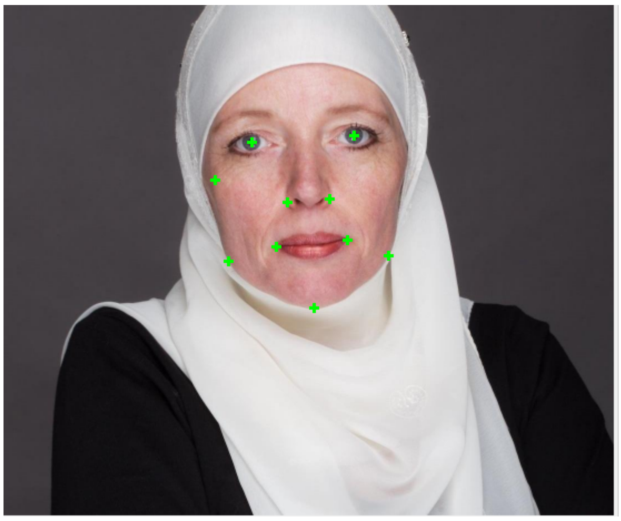


Figure 3.5: Manual placement of landmarks in a 2D face image.

FaceGen will calculate the 3D face based on the landmarks and the input image(s). The total time needed for the face in Figure 3.1 was 35.2 seconds. The result is seen in Figure 3.6.

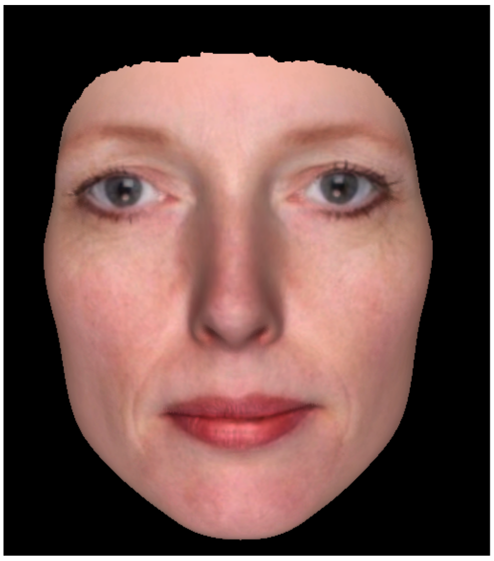


Figure 3.6: 3D face reconstruction using FaceGen.

3.3. The Winner

Both technologies were tested on the same training set of 11 images. Below is a summary of the basic differences between EOS and FaceGen, based on the training set.

EOS	FG
Non user friendly interface. Requires the installation of 3rd party libraries and compilation of the project in Visual Studio.	User friendly interface. Requires only the installation/extraction of the program.
No blendshapes. There is another program for this feauture called 4dface	A wide range of blendshapes that change the shape of the face in real time.
Requires image + a collection of 2D data points (usually a .pts file)	Requires at least one frontal face image and two optional side images. The landmarks are placed manually.
The number of landmarks are not limited (typically 68 points are enough).	11 landmarks minimum 29 landmarks maximum
The texture has the same resolution as the input image.	The texture has often low resolution (as seen in Figure 3.6). It produces better results with a high resolution image.
There is often severe texture stretching and untextured spaces (see 3.3)	No significant texture stretching No untextured space
6,736 polygons	11,438 polygons

It is only natural to choose the most efficient and user friendly software, and that is FaceGen.

4. CREATING A SYNTHETIC DATABASE FOR FACE RECOGNITION

In this chapter, a database of 100 random faces will be created with the FaceGen SDK. Other 3D databases that are worth mentioning, are :

- **Face Recognition Grand Challenge (FRGC)[6]:** The FRGC data set consists of 50,000 records, where 4,003 of them are used for validation and the rest of the subjects are used for training the 3D face recognition algorithm.
- **Bosphorus[7]:** The Bosphorus database includes 4666 3D faces, derived by 105 subjects. The database stores up to 35 expressions for each subject. Some 3D faces have a physical obstacle covering the face (hands, hair, eyeglasses), making the recognition process more challenging.
- **Binghamton University 3D Facial Expression(BU-3DFE) database[8]:** The BU-3DFE database contains 100 subjects (44 males and 56 females) from different age groups and ethnicities. Each subject has 7 expressions, and each expression has 4 levels of intensity. BU-3DFE was chosen to compete with the database created in this thesis. The expressions and their intensities were chosen in a round robin order.

The structure of the DB that will be created in the current thesis is as follows:
Each face has its own unique number from 1 to 100. The ”Data” folder contains the texture and obj file of the face , a frontal and side render. The ”Query” folder contains the texture , obj file and a frontal image of the same face with different facial expression (one of the 22 provided by FG). Each expression and its intensity will be randomly selected. An example of the DB’s structure is seen in Figure 4.1.

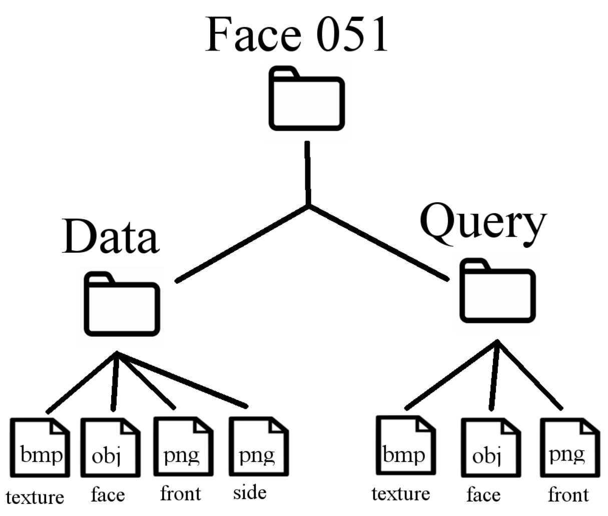


Figure 4.1: The structure of the database for an arbitrary number.

FG provides random values for the following parameters :

- **Gender**, Hypermale , Male , Female, Hyperfemale
- **Age**, 10 - 65 years
- **Ethnicity**, African, European, East Asian or South Asian
- **Asymmetry**, how asymmetrical is the shape of the face
- **Caricature Shape**, the degree of deformation
- **Caricature Color**, how ”normal” the color of the face is

Note that a person generated by FG might have intermediate values for age and gender. For example, a random person’s gender can be between Male and Female.

4.1. Random Face Generator

The database consists of 100 random faces, produced by a Windows batch script (.bat) file using commands from the FG SDK (see Appendix for more details). This chapter examines how the SDK builds a random face. FaceGen comes with 5 different model sets (csam) :

- **3DPrint**, used for 3D printing applications with a high poly model

- **Animate**, used to create morphs (facial expressions)
- **MakeHuman**, creates a face that can be easily used with MakeHuman software
- **Preview**, creates an 11K poly face without expressions
- **Real Time**, low poly cutout of a face with no facial expressions

Since we want facial expressions, we will use the Animate csam. Each csam contains the average face and texture. Every other face will be generated by adjusting the coefficients of the average face. The Statistical Shape Model (SSM), represents a polygonal model of fixed mesh topology. The SSM is stored as a file with .EGM extension, which saves information about the shape of the mean face and there is 1-1 vertex-feature correspondence (for example, the vertex at the corner of the eye will remain the same for both the average and the generated face). The Statistical Color Model (SCM), stored as a file with .EGT extension, describes the mean texture map (as a 24-bit RGB color image). In order to use a csam set, we need facial coordinates (described with a .fg file). These coordinates combined with the EGM and EGT will transform the average face and will output the base geometry corresponding to the new face (.tri file). Our new face is generated, but we want a widely recognised format (.obj) instead of .tri. FG provides tools to convert tri format to obj. The process is described briefly in Figure 4.2. The Figure 4.3 visualises two random faces created by Facegen. The two left images represent the neutral expression of the face and the right image represents the random expression for the particular face.

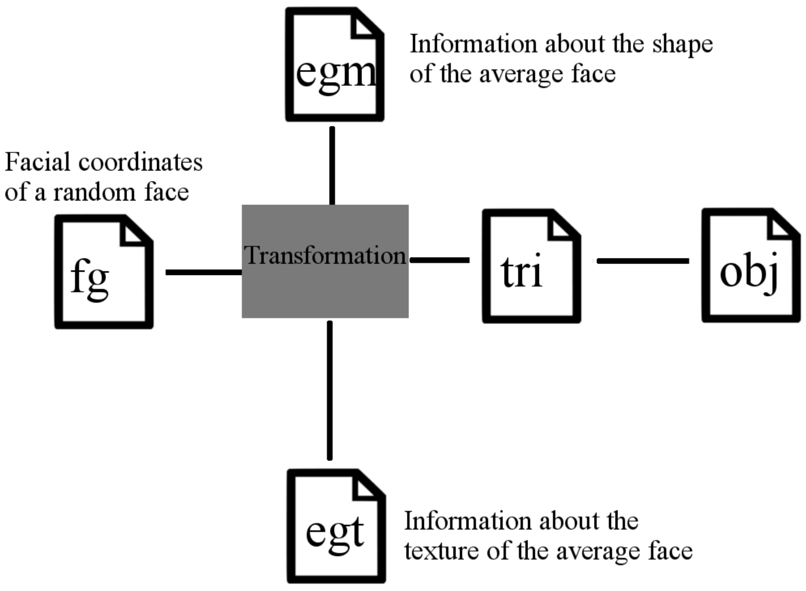

egm
Information about the shape of the average face
Facial coordinates of a random face
fg
Transformation
tri
obj
egt
Information about the texture of the average face

Figure 4.2: Creating a random face with a csam

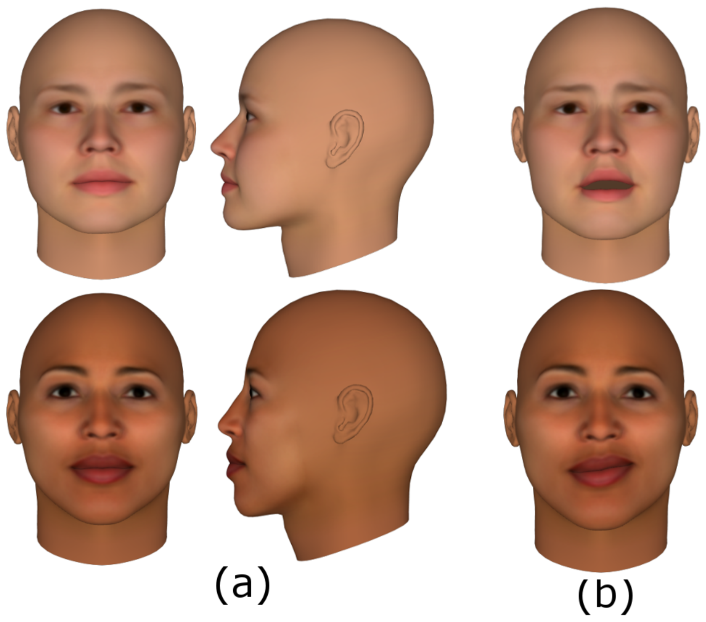


Figure 4.3: Random faces created by Facegen.
a) Neutral expressions
b) Random expression with random intensity

5. FACE RECOGNITION WITH URxD

After the executable programs are successfully produced (see Appendix), they will be used in conjunction with the FaceGen 3D database, in order to evaluate the performance of URxD in synthetic faces. For comparison purposes, URxD will also be used with the BU-3DFE database, which is a 3D DB consisting of 100 real faces. The evaluation metric that will be used for the two DBs, is CMC.

5.1. Workflow

The procedures that will be used from the URxD project, are :

- fimgen : creates the ”signature” of a 3D face
- simutil : creates similarity matrices, CMC and ROC curves

Fimgen takes as input the 3D faces of the DB, performs a registration, alignment and fitting process, and outputs a unique signature of each face with fixed size (121 KB).
As seen in Figure 5.1, the program is parametrized by a configuration file (config.ini). A typical configuration file is seen in Figure 5.2. The command ”extension” defines the format of the 3D face (wrl or obj). The commands ”alignmodel_file” and ”facemodel_file”, define the file that will be used for the alignment and fitting process. These files end with the extension .sas, and describe the Annotated Face Model that will be aligned and later wrapped around the 3D face in the alignment and fitting process respectively, in order to create the signature. Each sas file has different geometry and annotated areas. A cumulative depiction of all the available sas files is seen in Figure 5.3.

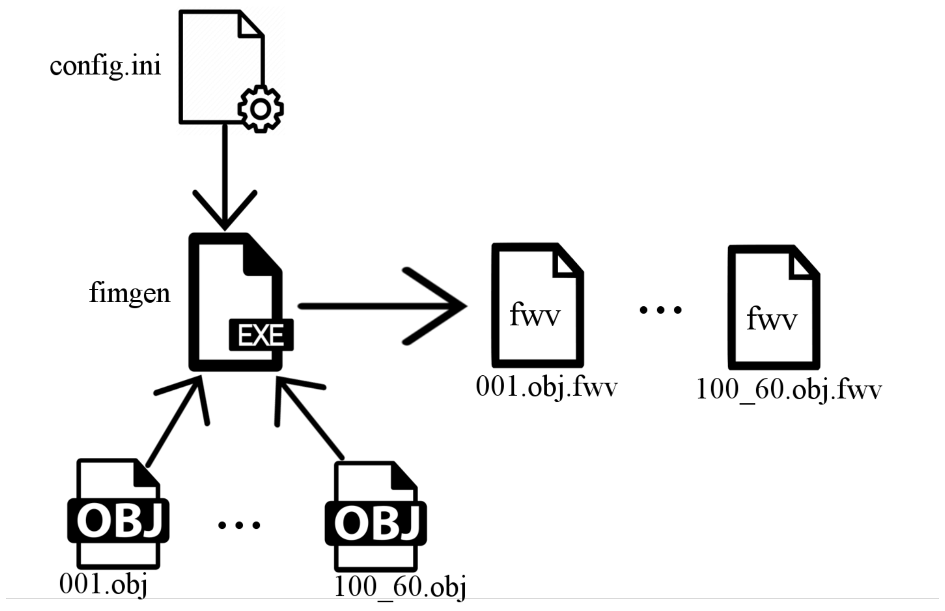
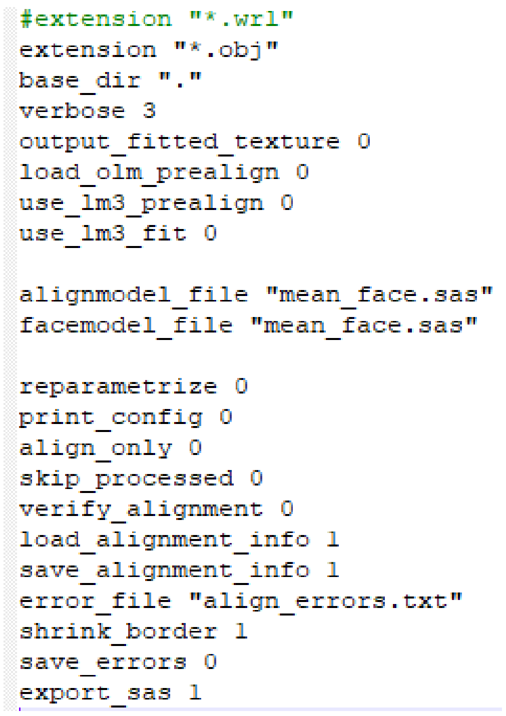


Figure 5.1: Fimgen workflow : The executable program takes as input the configuration file and the 3D database in obj or wrl format (Data + Query) and produces the signature for each face (fwv).

After the signatures (fwv files) have been created, another procedure called ***simutil***, will be used to extract CMC curves.
Simutil collects all the fwv files and creates a NxN similarity matrix , where N is the number of fwv files. In this thesis, N is 200, since 200 faces (100 query plus 100 database faces) in the FG database produce 200 signatures with fimgen. A similarity matrix contains the names of the signatures as rows and columns and each cell describes how similar two signatures are. For example, suppose we have the similarity matrix of the Figure 5.4 . The rows and columns represent the signatures (this example uses the FG DB) and subsequently, the elements of the main diagonal are zero (each face is 0 units away from itself) and the matrix is symmetrical. The cell located in the first row and third column, suggests that the distance from face 001 to face 002 is 234.296. In conclusion, the similarity matrix computes distances from each face to another.



```
#extension "*.wrl"
extension "*.obj"
base_dir "."
verbose 3
output_fitted_texture 0
load_olm_prealign 0
use_lm3_prealign 0
use_lm3_fit 0

alignmodel_file "mean_face.sas"
facemodel_file "mean_face.sas"

reparametrize 0
print_config 0
align_only 0
skip_processed 0
verify_alignment 0
load_alignment_info 1
save_alignment_info 1
error_file "align_errors.txt"
shrink_border 1
save_errors 0
export_sas 1
```

Figure 5.2: A typical configuration file for fimgen. The commands ”extension”, ”alignmodel_file” and ”facemodel_file” are the most important for the fimgen executable.

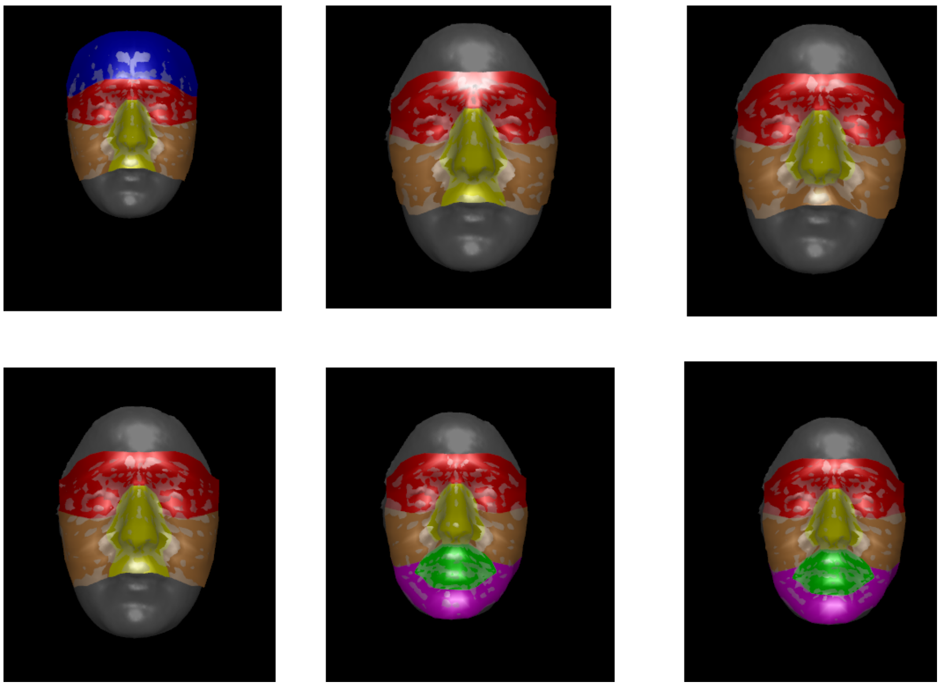
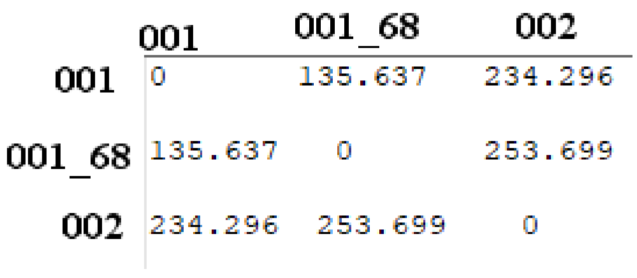


Figure 5.3: From top left to bottom right : half_face , meanface , slim_face2 , quarter_face , slim_full_face2 , slim_full_face_bones. The alignment process aligns the 3D face with the AFM, and the fitting process deforms the shape of the AFM in order to fit into the 3D face.



	001	001_68	002
001	0	135.637	234.296
001_68	135.637	0	253.699
002	234.296	253.699	0

Figure 5.4: similarity matrix

5.2. Performance Evaluation

The similarity matrix will be used to measure the performance of URxD in synthetic faces. Taking the non zero minimum element of each row, the algorithm checks if this element corresponds to the distance of the signature with the facial expression and the signature with the neutral expression that have the same index in the database. For example, examining the Figure 5.4 and taking the non zero minimum element of the first row (135.637), the result is that the face number 001 is closest to the face with number 001_68 (which is a correct result).
The algorithm counts the success rate based on the aforementioned criteria. Each sas file creates different signatures, because the AFM is different (see Figure 5.3). As a result, the success rate of the algorithm differs among the sas files. URxD was also used in a database consisting of real faces (BU-3DFE), in order to evaluate how the project performs in real and computer generated faces respectively. Both databases have 200 faces in total (100 faces with neutral expression, each with an additional random expression). The table below aggregates the success rates for each available sas file for the two databases.

Table 5.1: URxD success rate (%) for different sas files

sas file	FaceGen	BU-3DFE
half_face	67	91
mean_face	43	91
quarter_face	38	91
slim_face2	40	90
slim_full_face2	42	91
slim_full_face_bones	42	91

5.2.1. CMC Curve Definitions

In machine learning, a Cumulative Match Characteristics curve is a method of showing the performance of recognition precision for each rank, and it is defined by the True Positive Rate (TPR), False Positive Rate (FPR), True Negative Rate (TNR), and False Negative Rate (FNR).

True Positive Rate : Also known as sensitivity or recall, is used to measure the percentage of actual positives which are correctly identified.

False Positive Rate : Measures the percentage of data which are incorrectly classified as negative (false alarm).

True Negative Rate : Probability of correctly predicting the negative class.

False Negative Rate : Probability of incorrectly predicting the negative class.

The following equations hold for the above rates :

$$TPR = 1 - FNR$$

$$TNR = 1 - FPR$$

Each probe sample is compared against all gallery samples, and the resulting scores are sorted and ranked. A ROC curve can be extracted from a CMC curve and vice versa. The figures below illustrate the CMC curves for FaceGen and BU-3DFE respectively. The horizontal axis represent the ranks and the vertical axis represent the recognition rate (TPR). The faster the curve reaches the value 1.00 , the better the performance is for the particular model. For an in depth analysis of the creation of the similarity matrices and CMC curves, see Appendix.

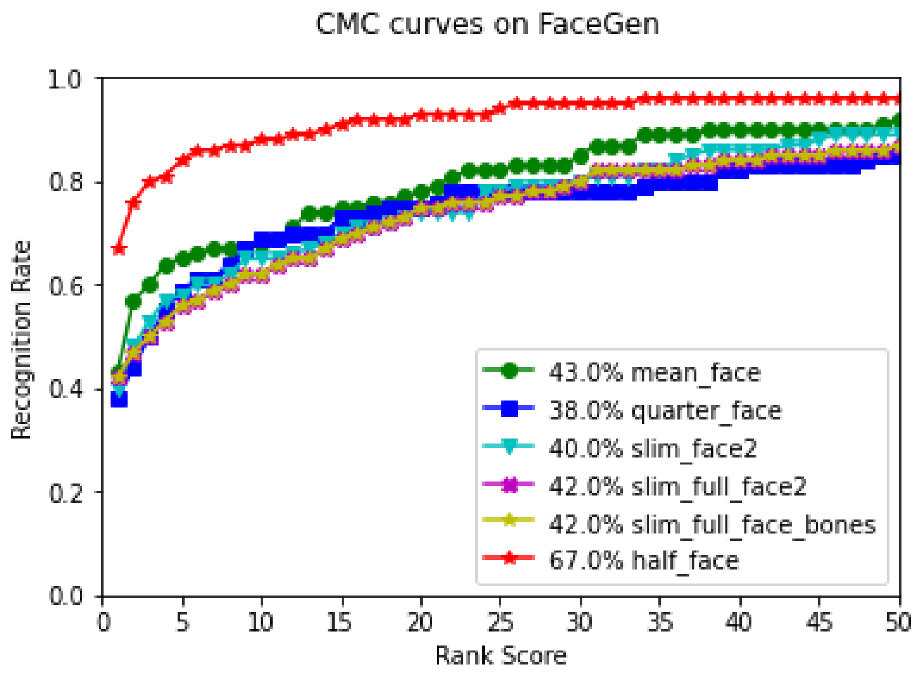


Figure 5.5: CMC curves for the FG database for each available sas file

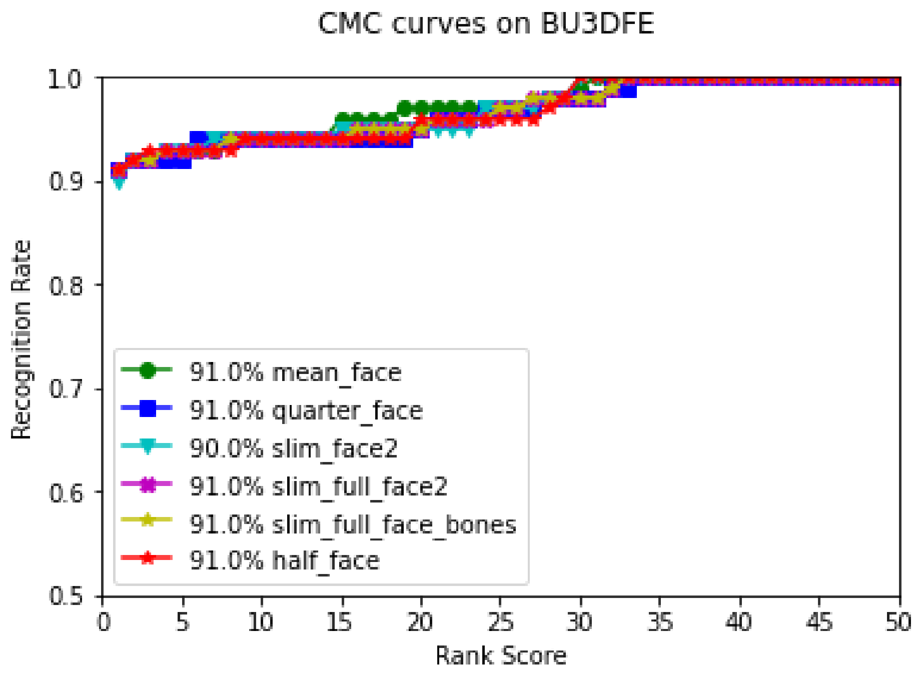


Figure 5.6: CMC curves for the BU-3DFE database for each available sas file

6. DISCUSSION AND FUTURE WORK

Examining the table 5.1, it is evident that URxD produces worse results for FaceGen than BU-3DFE, by a large margin. A potential culprit for the unsatisfying results could be the registration phase. Many faces (and that only happened with the FaceGen DB), were extremely rotated in the registration phase and as a result, the corresponding procedure tried to fit the AFM into the rotated face, and the algorithm produces a distorted fwv signature. Figure 6.1 depicts this phenomenon.
It is unclear why this happens only with FaceGen, since both databases have the same dimensions. One observation for this problem, is that if the face (obj file) is rotated by a third party program before feeding it into fimgen, then the amount and the axis of the rotation affects the registration phase. There is no automated process that can correctly rotate all the faces, so that they can be registered correctly by fimgen, because each face has its own parameters and it is unpredictable how fimgen will rotate the faces in the registration phase. If a face is slightly rotated in the wrong direction, it can have disastrous consequences in fimgen. Figure 6.2 describes this problem. The left picture depicts the initial 3D face produced by FaceGen, which is rotated in the registration phase. The middle picture depicts the same face which is now manually rotated 30 degrees on the Z axis using Blender, before using the object as input in fimgen. This transformation seems to create satisfying results for the current face. The right picture is the same face rotated 30 degrees on the Z axis and -12 degrees on the X axis. It is clear that the algorithm is very sensitive to 3D transformations.
A few methods that may help improve the overall performance of URxD are listed below :

1. Take into account the texture map. The fwv distance along with the colour of the face can improve the recognition rate of the algorithm. Apart from the shape of the face, the fwv signature can contain additional information about the texture of the face, which means slightly larger signatures. If this technique increases the running time significantly, it can only be used to clear up ambiguities regarding the identity of the face. After the initial recognition rate is derived by simutil (without texture information in the signature), the algorithm can examine the failed identifications and attempt to re-match them by comparing the texture image along with the shape of the wrongly

matched faces.

2. Higher resolution AFM. In this thesis, URxD was run in a 5 year old moderate laptop with 30-35 seconds for each face. A modern day computer can reduce the running time at least by half. Subsequently, the resolution of the AFM can be increased, so that more details can be captured. It could be a trade-off between acceptable running time and performance. This method can be combined with the technique proposed in [9]. This technique takes advantage of the symmetry of the face and the half AFM mask can have a higher resolution.

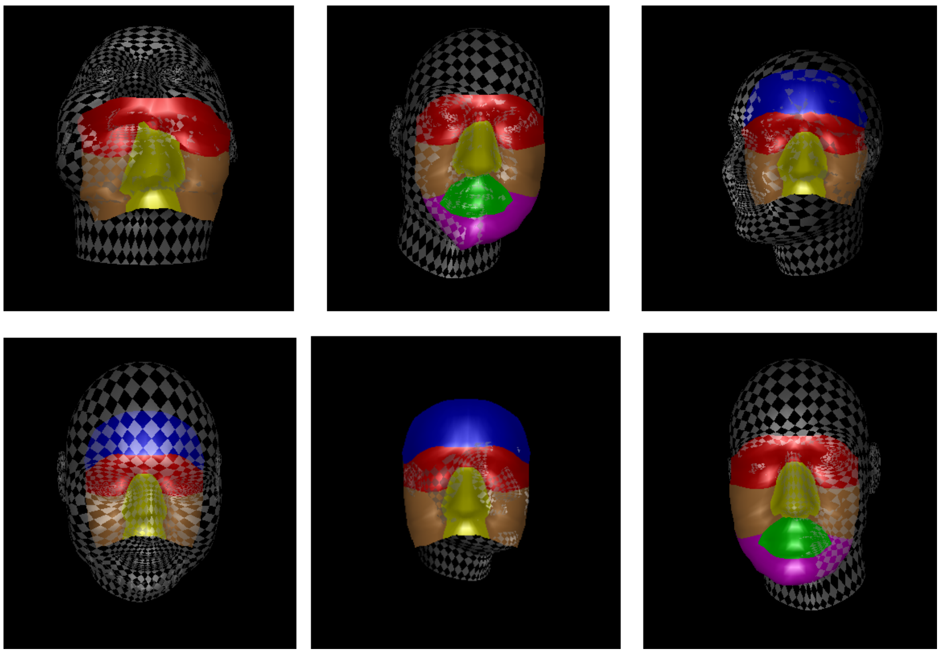


Figure 6.1: Incorrect registration phase of fimgen. The 3D object is rotated and the fitting process tries to fit the AFM into the wrongly rotated face, resulting in wrong signatures.

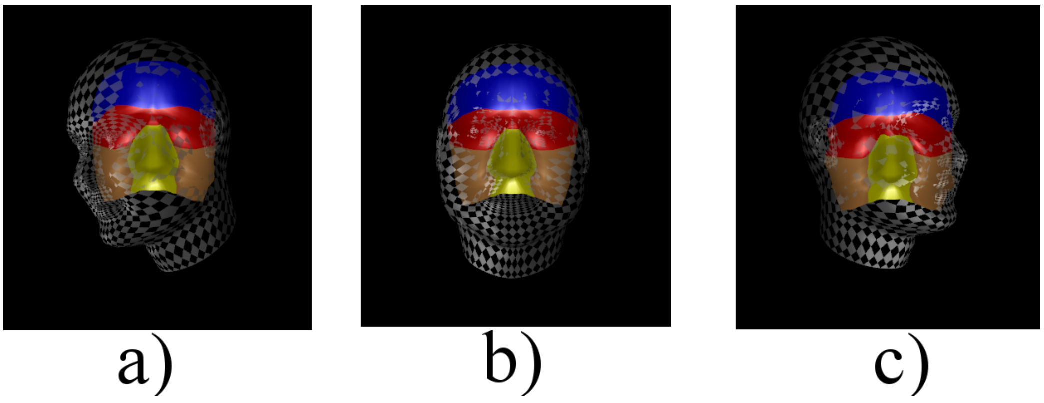


Figure 6.2: Fimgen results on the same 3D face
a) produced by FaceGen without any transformations
b) manually rotated 30 degrees on the Z axis
c) manually rotated 30 degrees on the Z axis and -12 degrees on the X axis

APPENDIX

EOS Installation

This section describes the process for the installation and usage of EOS. The program has additional dependencies that are listed below :

- Boost [10] , which provides a plethora of C++ libraries. EOS needs the following libraries from Boost :
  - filesystem , which provides facilities to manipulate files and directories
  - program options , used for declaring and managing options for a program
- OpenCV [11] , a compilation of C++ libraries used specifically in computer vision applications. The libraries needed from OpenCV are :
  - highgui , a high level GUI and Media Input/Output library
  - imgproc , an image processing library
- CMake [12] , a number of tools designed to build, test and package software

After all these dependencies are installed, the user must first clone the repository using the command :

```
git clone --recursive https://github.com/patrikhuber/eos.git
```

Then, a build directory must be created, inside of which the Visual Studio solution (.sln) files will be produced. The commands that achieve those steps are :

```
 # creates a build directory next to the 'eos' folder
mkdir build && cd build

cmake -G "<your favorite generator>" ../eos
-DCMAKE_INSTALL_PREFIX=../install/
```

The next step is to either run the eos.sln in VS on a Windows system, or execute the

```
make && make install
```

command on Linux systems. Note that Windows users must conFigure the VS compiler and linker in order to compile the project without errors.
Inside the install/bin folder are the executable files needed to create the face, specifically the fit-model-simple.exe. The install/data folder contains the 2D images and landmarks corresponding to those images. The command used to generate a 3D face from a 2D image, is :

```
fit-model-simple -m ../share/sfm_shape_3448.bin -p
../share/ibug_to_sfm.txt -i "Image File"
-l "Landmark file"
```

The Landmark file is usually a .pts file which is a collection of (typically 68) XY coordinates of an image. Each coordinate represents a facial feature. A pts file can be produced either manually or with the help of a C++ machine learning library like Dlib.

FaceGen Batch Script Decomposition

This section examines the batch script for the creation of the 3D database. Note that we want a front and side render of the face. When a render is created for the first time, FG produces an XML file that stores tags about the texture, rotation and translation of the face, the light and background colour etc. A render places the face in frontal pose by default. We will use the rotation tag of the XML file to rotate the face, and then take a new render with the new rotation value. A small batch script is needed to replace tags for an XML file. This program is called JREPL and the batch script that generates the random faces is the RandomFaces.bat .
The first lines of code declare the full paths for the SDK, JREPL and Animate csam. These values must be filled by the user.

```
set sdk=your_sdk_path
set JREP_PATH=your_jrepl_path\JREPL.BAT

set Animatecsam=%sdk%\data\csam\Animate
set Head=%Animatecsam%\Head\HeadHires
```

The next command sets the PATH environment variable to include the FG executables

```
set PATH=%PATH%;%sdk%\bin\win\x64\vs17\release\;%sdk%\bin\win\x64\
```

A directory with name ”FaceGen_DB” is created. Inside this folder, the DB’s structure will be created. Next, a loop is executed 100 times. In each loop, a structure seen in Figure 4.1 is produced, where each folder has name ”XXX”, where XXX takes values from 001 to 100. The command below creates a random face with random gender and ethnicity.

```
fg3.exe create random any any %newdir%.fg
```

The above command produces XXX.fg. For example, in the 4th loop, FG will produce 004.fg. Next, the head mesh is constructed with the Animate csam, and converetd to obj format:

```
fg3.exe construct %Head% %newdir%.fg %newdir% -d 2.0
^^I
fg3.exe mesh convert %newdir%.tri %newdir%.obj
```

We want to have random expression and intensity. Batch scripts do not support decimals, but there is a workaround, by creating a random number for the whole and one for the decimal part. The two parts are combined to a float number. FG provides 22 different expressions, and the batch script chooses one randomly. Then, we create a new .fg file (with new facial coordinates caused by the expression), and we convert it to obj.

```
set /a whole=(%RANDOM%*1/32768)+0
set /a decimal=(%RANDOM%*5/32768)+5
^^I
^^I
REM choose random intensity from 0.5 to 0.9
set random_intensity=%whole%.%decimal%

REM choose among 22 different expressions
set /a random_expression=(%RANDOM%*22/32768)+51

set expression=%newdir%_%random_expression%

REM create random expression and convert it into obj
fg3.exe morph apply %newdir%.tri %expression%.tri d
```

```
%random_expression% %random_intensity%

fg3.exe mesh convert %expression%.tri %expression%.obj
```

The final part of the script creates the renders. First, a render is created, in order to obtain the initial XML file. JREPL is used to insert the texture image to the imgFilename tag. Then, a frontal render is created. For the side render, JREPL is used one more time to change the panRadians tag. The files are moved to the appropriate folders, and any other file that will not be used is deleted.

URxD Project Restoration

URxD is a face recognition project written in C++, based on the 3DMM concept. The project offers numerous tools and procedures. It was tested on a large database consisting of real faces, and it managed to score over 97% success rate! URxD was last modified over ten years ago. This chapter examines the process of restoring the code, in order to run the project in modern x64-architecture machines.

Visual Studio Project Configuration

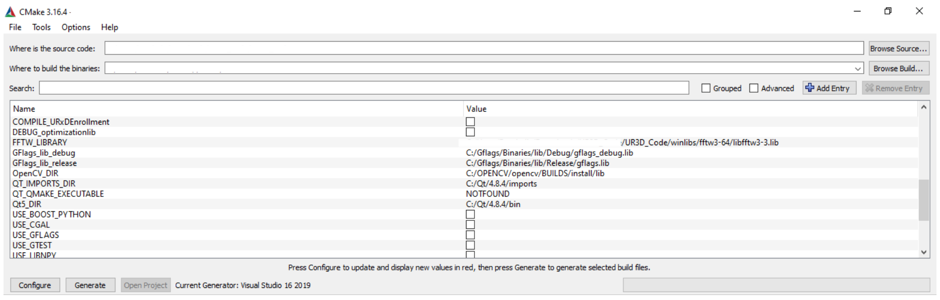
URxD was first built and compiled in VS. As a result, it comes with a VS solution (sln), that consists of 73 sub-projects. Once they are compiled, these sub-projects are either executable programs, or libraries used for the compilation of other projects. The problem is that the provided solution was produced in VS 2007, and the paths to the C++ files were hardcoded within the solution. Fortunately, URxD was packed with a CMake rule, which will be used to create a new the VS solution, this time for the VS 2019 version.

Before continuing with the creation of the solution with CMake, certain dependencies must be installed (assuming Visual Studio 2019 and CMake are already installed) :

- QtCore 4.8.4 [13]
- OpenCV [11]
- GFlags [14] (Google CommandLine Flags)
- FFTW Library [15]

- ZLIB [16]

Follow the installation instructions carefully for each dependency provided in the links.
The image below depicts the CMake GUI program. The source code is the CMakeLists.txt file found in the folder where the project is located. The second path is the desired location of the CMake output. The paths for the aforementioned dependencies should be inserted correctly, as it is depicted in the image.



Name	Value
COMPILE_URxDEnrollment	☐
DEBUG_optimizationlib	☐
FFTW_LIBRARY	/UR3D_Code/winlibs/fftw3-64/libfftw3-3.lib
GFlags_lib_debug	C:/Gflags/Binaries/lib/Debug/gflags_debug.lib
GFlags_lib_release	C:/Gflags/Binaries/lib/Release/gflags.lib
OpenCV_DIR	C:/OPENCV/opencv/BUILDS/install/lib
QT_IMPORTS_DIR	C:/Qt/4.8.4/imports
QT_QMAKE_EXECUTABLE	NOTFOUND
Qt5_DIR	C:/Qt/4.8.4/bin
USE_BOOST_PYTHON	☐
USE_CGAL	☐
USE_GFLAGS	☐
USE_GTEST	☐
USE_LIBNPY	☐

Press Configure to update and display new values in red, then press Generate to generate selected build files.

Current Generator: Visual Studio 16 2019

Figure 6.3: CMake Graphical User Interface : Project parametrization

Before the ConFigure button is clicked, the CMakeLists.txt of some programs must be edited to contain the line :

```
cmake_policy(SET CMP0079 NEW)
```

The programs that need the above command in their CMakeLists.txt files, are:

abs2wrl	process3d
fimgen	aem_train
fileconvert	lmk_analysis
simutil	get_bounding_box
megamopt	xyz2uv
pyramid_transform	afm_lmk_closest
UR2DOneToOne	indirect_align
extract_wavelet	refined_lmk_detect
svm_train	export_object3d
fwv2svm	projectivetexture
histFromALNWRL	ASM_Train
afm_projection	SI_Detect
triangle_removal	crop_face_3d
3d_feature_gen	lmk_fitting

Click ”Configure” and then ”Generate”. CMake will create a solution named URxD.sln with all the sub-projects compatible with VS2019.
Before continuing with the compiler and linker errors, Qt must be compatible with VS. To do that, open the MSVC command line (x64 Native Tools Command Prompt for VS2019). Change working directory to Qt/4.8.4 and type:

```
configure.exe -platform win32-msvc2012 -confirm-license  -debug
-opensource  -nomake tools -nomake examples -nomake tests
```

After the above command is completed, type:

```
nmake
```

Compiler Errors

Since the code is relatively old, it must be slightly modified in order to compile the project without errors. This section provides a step-by-step solution :

- Open URxD.sln with VS2019.
- Choose the Solution Configuration and Solution Platform. In this thesis, URxD was built with the options : Debug x64.
- Open the file ”general/common/include/object3d.h” and add the line ”#include <opencv2/flann.hpp>”
- In projects : filemanlib , crop_face3d , libsvm , software_z :
  Go to Properties $\rightarrow$ C/C++ $\rightarrow$ General $\rightarrow$ Additonal Include Directories
  click Edit and add 2 paths :

  1. The first path is where the file cv.h is located
     yourOpenCVpath/build/include/opencv
  2. The second path is
     yourOpenCVpath/build/include

- Add the lines:
  #include <GL/gl.h>
  #include <GL/glut.h>

in files:
general/meshlib/GLMesh.cpp
general/meshlib/glviewer.cpp
programs/PDMVisualizer/glUtils.cpp

- Add the line:
  #include <opencv2/opencv.hpp>
  in file : /sdk/mkdetectlib/LandmarkDetectorL2.cpp

- In file : /general/auxlib/src/applyaln.cpp
  Change : return Mat(mat).clone();
  to : return cv::cvarrToMat(mat).clone();

- Add the line:
  #include <algorithm>
  in files:
  /general/meshlib/TPS.cpp
  /general/meshlib/linearAlgebra.cpp

- In files:
  viewfimseq.cpp
  alignfeaturestats.cpp
  histfromanwrl/main.cpp
  /programs/view3d/main.cpp

  Find all instances of ”Object3D* data” and change it to ”Object3D* data1”.

- In files :
  InputByteMatrix.cpp
  OutputByteMatrix.cpp
  InputFloatMatrix.cpp

  Change : if (simStream == NULL)
  to : if (!simStream)

- In directory : /sdk/lmkdetectlib :
  Open all header files and delete the line : using namespace cv;
  Add cv:: before the identifiers : Mat , Point3f , Scalar

- In project view3dvideo , right click in solution explorer and go to :
  Properties $\rightarrow$ C/C++ $\rightarrow$ General $\rightarrow$ Additional Include Directories (as

seen in Figure 6.4)
Click Edit and add : yourGflagsPath/Binaries/include

- In file /sdk/facedetectlib/src/facedetection.cpp :
  In lines 21-28 delete the extra brackets.
  For example :
  {{0,0,255}}
  will be changed to :
  {0,0,255}

Linker Errors

URxD needs libraries to produce the executables. Many of these libraries are part of the solution, and they must be linked with the appropriate project. If the desired configuration is Release, change the directories below to ”/Release” instead of ”/Debug”

- In projects :

  1. abs2wrl
  2. viewfimseq
  3. PDMVizualizer
  4. view3d
  5. view3dvideo
  6. Spin_Train
  7. test_curvature
  8. gmmr
  9. surfacefittinglib
  10. spin_image_gen
  11. SI_Detect
  12. spin_gen_takis
  13. MeshViewer
  14. pca
  15. object3d_trimming
  16. projectivetexture
  17. 3dviewer_projection

18. megamopt
19. lmk_analysis
20. landmark_dist
21. Spin_Train_lm2e
22. fimgen
23. histFromALNWRL
24. fileconvert
25. ffs
26. failurestats
27. extract_wavelet
28. face_profile_line
29. exgen
30. afm_projection
31. afm_edit
32. assign_color
33. simutil
34. lda
35. lmk_verify

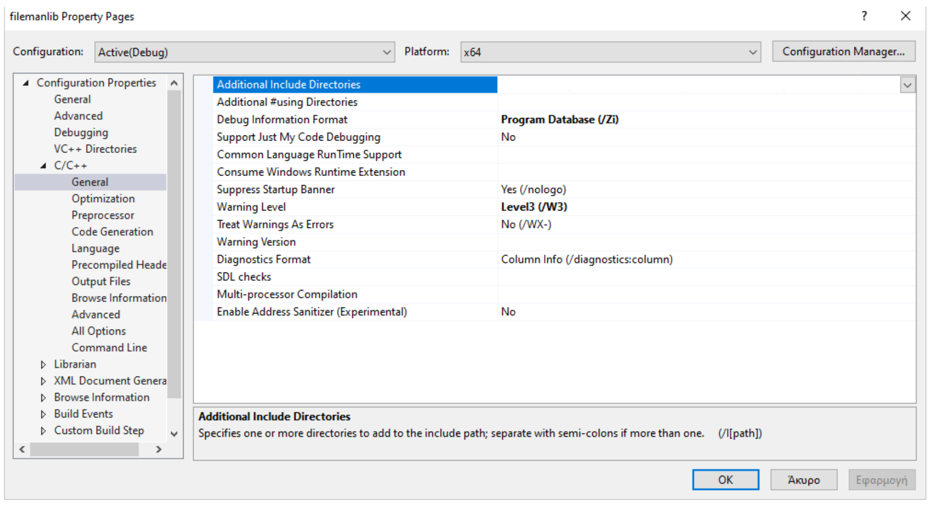
Go to Linker→General→Additional Library Directories
Add your Opencv path/BUILDS/lib/Debug
Go to Linker→Input→Additional Dependencies
Add opencv_world343d.lib

- In project crop_face_3d :

1. Go to Linker→General→Additional Library Directories (as seen in Figure 6.5)
   Add the following paths :
   your BUILD PATH/general/ndiolib/Debug
   your BUILD PATH/general/common/Debug
   your BUILD PATH/general/auxlib/Debug
   your BUILD PATH/general/deformlib/Debug
   your BUILD PATH/general/filemanlib/Debug

2. Go to Linker→Input→Additional Dependencies
   Add the libraries :
   ndiolib.lib
   common.lib
   auxlib.lib
   deformlib.lib
   filemanlib.lib

Right click the solution in the explorer and choose ”Build Solution”. All the executables should now be located in the desired build path, inside the folder ”programs”.
The full restoration of the code took 1 month of total work.


filemanlib Property Pages

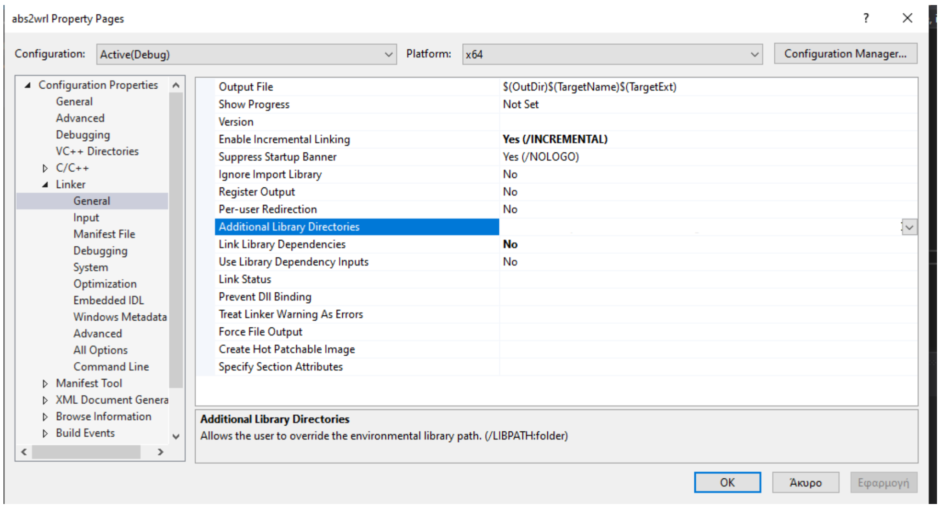
Configuration: Active(Debug) Platform: x64

- Configuration Properties
  - General
  - Advanced
  - Debugging
  - VC++ Directories
  - C/C++
    - General
    - Optimization
    - Preprocessor
    - Code Generation
    - Language
    - Precompiled Heade
    - Output Files
    - Browse Information
    - Advanced
    - All Options
    - Command Line
  - Librarian
  - XML Document Genera
  - Browse Information
  - Build Events
  - Custom Build Step

Additional Include Directories	
Additional #using Directories	
Debug Information Format	**Program Database (/Zi)**
Support Just My Code Debugging	No
Common Language RunTime Support	
Consume Windows Runtime Extension	
Suppress Startup Banner	Yes (/nologo)
Warning Level	**Level3 (/W3)**
Treat Warnings As Errors	No (/WX-)
Warning Version	
Diagnostics Format	Column Info (/diagnostics:column)
SDL checks	
Multi-processor Compilation	
Enable Address Sanitizer (Experimental)	No

Additional Include Directories
Specifies one or more directories to add to the include path; separate with semi-colons if more than one. (/I[path])

Figure 6.4: VS 2019 Compiler Options



abs2wrl Property Pages

Configuration: Active(Debug) Platform: x64 Configuration Manager...

- Configuration Properties
  - General
  - Advanced
  - Debugging
  - VC++ Directories
  - C/C++
  - Linker
    - General
    - Input
    - Manifest File
    - Debugging
    - System
    - Optimization
    - Embedded IDL
    - Windows Metadata
    - Advanced
    - All Options
    - Command Line
  - Manifest Tool
  - XML Document Genera
  - Browse Information
  - Build Events

Output File	$(OutDir)$(TargetName)$(TargetExt)
Show Progress	Not Set
Version	
Enable Incremental Linking	**Yes (/INCREMENTAL)**
Suppress Startup Banner	Yes (/NOLOGO)
Ignore Import Library	No
Register Output	No
Per-user Redirection	No
Additional Library Directories	
Link Library Dependencies	**No**
Use Library Dependency Inputs	No
Link Status	
Prevent Dll Binding	
Treat Linker Warning As Errors	
Force File Output	
Create Hot Patchable Image	
Specify Section Attributes	

Additional Library Directories
Allows the user to override the environmental library path. (/LIBPATH:folder)

Figure 6.5: VS 2019 Linker Options

Creating URxD Experiments

In order to run fimgen, open a command line prompt and type:

```
fimgen.exe -your working directory-
```

where ”-your working directory-” is a directory with the following files :

1. All the 3D data files (faces) in obj or wrl format
2. The ”matrix” directory, which can be found in ”cluster/req” in the URxD directory (the non compiled version)
3. All the available .sas files located in ”cluster/req”
4. ”area_map.bmp” found in ”cluster/req”
5. A configuration file named config.ini containing the desired commands for fimgen
6. freeglut.dll
7. libfftw3-3.dll
8. Qt 4.8.4, OpenCV and VS2019 installed. Alternatively, the files:
   - QtCored4.dll
   - opencv_world343.dll

- opencv_world343d.dll

must be present in the directory. Since the project needs many dll files from VS, it is recommended that VS2019 is fully installed in the target machine.

Fimgen was executed in a mid/low end laptop with specs :

- Intel i5 5200U
- Nvidia GT 920M 2GB
- 8GB RAM DDR3

Each face took roughly 30 seconds for the registration and fitting process. The table below describes the total elapsed time for each sas file for the two databases:

Table 6.1: Fimgen running time in seconds for different sas files

sas file	FaceGen	BU-3DFE
half_face	5471	6588
mean_face	3915	4838
quarter_face	4358	5683
slim_face2	6263	6639
slim_full_face2	6246	5473
slim_full_face_bones	5997	6930

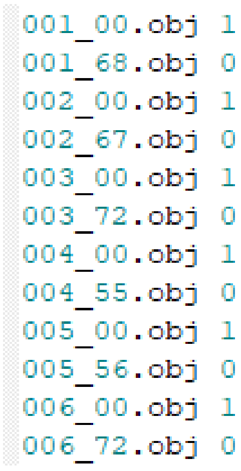
After fimgen is completed, a new folder with the name ”fwv” will be created, in which all the .fwv files are located. Inside this folder, the following files must be present in order to run simutil :

1. uv_wavelet.bmp found in ”cluster/req”
2. The compiled ”simutil.exe”

The simutil program has several commands for the creation of full and sub similarity matrices and operations on them (conversion from full to sub, combining two similarity matrices together, finding the minimum of two similarity matrices etc.), as well as the creation of ROC and CMC curves. Only two commands will be needed to create the similarity matrix and the CMC curve. The first command is :

```
simutil.exe c newmatrix
```

The above command produces a similarity matrix with the name newmatrix, and outputs ”newmatrix.sim”. Before producing the CMC curve, simutil needs a tab file that differentiates the galleries from the probes. Along with ”newmatrix.sim” , simutil produces a header file that contains the names of all the fwv files present in the directory. The header file can be modified in order to create the tab file. At each row of the header file, if the fwv name represents a gallery name, the number 1 will be inserted after a space. Else, the number 0 will be inserted. Then, the file must be renamed to contain the .tab extension. A brief example is shown below.



```
001_00.obj 1
001_68.obj 0
002_00.obj 1
002_67.obj 0
003_00.obj 1
003_72.obj 0
004_00.obj 1
004_55.obj 0
005_00.obj 1
005_56.obj 0
006_00.obj 1
006_72.obj 0
```

Figure 6.6: A typical tab file for simutil. The number 1 represents a gallery name (a face without expression) and the number 0 represents a probe name (a face with a random expression).

After the tab file is created, simutil will be used to create the CMC curve. The command is :

```
simutil.exe gp newmatrix tabfile.tab
```

The program creates a text file containing 50 ranks and the recognition score for each rank, along with a file that lists the failures (if any).

ABBREVIATIONS

3DMM	3D Morphable Model
DB	Database
FG	FaceGen
SDK	Software Development Kit
VS	Visual Studio
GUI	Graphical User Interface
CMC	Cumulative Match Curve
AFM	Annotated Face Model
API	Application Programming Interface
FRGC	Face Recognition Grand Challenge
ROC	Receiver Operating Curve
SSM	Statistical Shape Models
SCM	Statistical Color Models
TPR	True Positive Rate
FPR	False Positive Rate
TNR	True Negative Rate
FNR	False Negative Rate
TNR	True Negative Rate
TNR	True Negative Rate
BU-3DFE	Binghamton University 3D Facial Expression

REFERENCES

[1] Volker Blanz and Thomas Vetter. A morphable model for the synthesis of 3d faces. *SIGGRAPH'99 Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, 09 2002.

[2] V. Blanz and T. Vetter. Face recognition based on fitting a 3d morphable model. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(9):1063–1074, 2003.

[3] Ioannis Kakadiaris, Georgios Passalis, George Toderici, Mohammed Murtuza, Yunliang Lu, Nikos Karampatziakis, and Theoharis Theoharis. Three-dimensional face recognition in the presence of facial expressions: An annotated deformable model approach. *IEEE transactions on pattern analysis and machine intelligence*, 29:640–9, 05 2007.

[4] EOS GitHub Repository. *https://github.com/patrikhuber/eos*.

[5] FaceGen Official Website. *https://facegen.com/*.

[6] FRGC 3D databse. *https://www.nist.gov/programs-projects/face-recognition-grand-challenge-frgc*.

[7] Bosphorus 3D databse. *http://bosphorus.ee.boun.edu.tr/Home.aspx*.

[8] BU-3DFE 3D database. *https://www.cs.binghamton.edu/~lijun/Research/3DFE/3DFE_Analysis.html*.

[9] T. Theoharis, P. Perakis, G. Passalis, and I. A. Kakadiaris. Using facial symmetry to handle pose variations in real-world 3d face recognition. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 33(10):1938–1951, oct 2011.

[10] Boost C++ Libraries. *https://www.boost.org/*.

[11] OpenCV Version 3.4.3. *https://opencv.org/releases/page/3/*.

[12] CMake latest official release. *https://cmake.org/download/*.

[13] Qt 4.8.4 Open Source Library for Visual Studio. *https://download.qt.io/archive/qt/4.8/4.8.4/*.

[14] Google Commandline Flags GitHub Repository. *https://github.com/gflags/gflags*.

[15] FFTW Library for Windows OS. *http://www.fftw.org/install/windows.html*.

[16] ZLIB Official Website. *https://zlib.net/*.