AUTOMATING A 3D POINT MATCHING SYSTEM FOR HUMAN FACES

by

Priya Vashistha

A Thesis Submitted in

Partial Fulfillment of the

Requirements of the Degree of

Master of Science

in Computer Science

at

The University of Wisconsin Milwaukee

May 2017

ABSTRACT

AUTOMATING A 3D POINT MATCHING SYSTEM FOR HUMAN FACES

by

Priya Vashistha

The University of Wisconsin-Milwaukee, 2017
Under the Supervision of Professor Zeyun Yu

3D point matching for human faces is opening new possibilities in the fields of face matching, face recognition, face retrieval, biomedical, virtual reality, etc. and is overcoming the limitations of 2D face matching. The purpose of this study is to research and implement an automated 3D point matching system for human faces. This will be added to an existing system implemented for 3D point matching on face models. The current implementation is a manual procedure to find matching between the faces, where a set of landmarks are selected on both sources and target meshes and the faces are registered using ICP and TPS techniques. The study aims to eliminate the manual process by automating the initial landmark selections.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

Firstly, I would like to thank Dr. Zeyun Yu for recognizing my capabilities and giving me this amazing opportunity to work with him on this project. His constant motivation and expert guidance enabled me to successfully complete this study. I am especially indebted to Reihaneh Rostami, who has enlightened me with her guidance and knowledge and was a strong support system for this endeavor. This study wouldn't have been possible without her continuous involvement and considerable personal and professional counsel. I have always highly appreciated her professionalism and knowledge of the topic of my study and have learned many valuable skills during the process. This study has empowered me with confidence and has shaped me into a better professional.

I am also very grateful to my friends and family who have always showered their love and blessings in all my ventures.

Chapter 1- Introduction

1.1 Background and Literature

2D Facial recognition has been an active research area for many years and a significant progress has been made to automate face recognition for 2D images. However, there are certain limitations with 2D images, for the facial recognition to be applicable, it needs to deal with variations in pose, color, lighting, background, resolution etc. With the recent development in 3D scanning, it has become more affordable and a complete new dimension for facial matching has been uncovered. Unlike the 2 dimensional facial recognition, 3 dimensional format overcomes the limitations of lighting, resolution, background, etc. 3D facial data is invariant to setting conditions and has closer real life applications. 3D measurements are of same size as the actual object which gives more accurate information about depth, foreground and background. It also has numerous applications in the fields of facial recognition, face retrieval, face reconstruction, biometrics and virtual reality.

The inspiration of this project came from an existing system of 3D point matching. Two 3 dimensional human face models are taken where one is the source mesh and the other one is the target mesh. The aim is to deform the source mesh to make it look like the target mesh while maintaining the source mesh topology. This is obtained by manually selecting landmarks which are spread out throughout the face on both source and target meshes on the same location and in the same order. The selected points are then used to deform the source mesh and make it look more like the target mesh. This process would be a lot more user friendly if

the landmark detection was automatic rather than manual, this problem led to the current study where an automatic system to generate landmarks have been implemented.

In this study, we approached to solve this problem of automatic point detection on both source and target faces with different algorithms such as constraint kmeans, modified kmeans, meshSIFT, etc, out of which meshSIFT has generated the most promising results so far. We will discuss meshSIFT in details in Chapter 3. In the current implementation, the source and target meshes are converted to OFF format (discussed in Chapter 2) and fed as input to meshSIFT. MeshSIFT returns matching points for the meshes, these points are not very accurate and are filtered before the source mesh is deformed.

In the current system implementation, we take two 3d face models, a source mesh and a target mesh and deform the source mesh to the target. This is achieved by first detecting the tip of the nose, top of the forehead, left ear and right ear which is implemented based on the basic human face structure. Then, the points obtained from meshSIFT are filtered using a threshold value to get rid of the wrong matchings. The points for nose, forehead, ears and those from meshSIFT are then combined and used to transform the source face geometry and make it look like the target.

1.2 System Setup

Various open source software and libraries have been used to implement this project. The base code is written in C++ programming language and built over Microsoft Visual Studio. Additional external libraries that have been used are VTK, OpenCV and Boost. Cmake has been used to

build binaries for external libraries. All the software being used is Open Source and freely available online for download. Table 1.1 gives a detailed information on the software used.

S.No	Software Name	Purpose	Version	Download Source	Install Guide Resources
1	Microsoft Visual Studio	Coding Interface	Express 2015	https://www.visualstudio.com/downloads/	Executable file available
2	Visualization Toolkit-VTK	Additional External Library – for Visualization of 3D data	VTK 7.0	http://www.vtk.org/download/	http://www.vtk.org/Wiki/VTK/Building/Windows
3	Open Source Computer Vision-OpenCV	Additional External Library	OpenCV 2.4.13	http://opencv.org/downloads.html	Executable file available
4	Boost C++ Libraries	Additional External Library	Boost 1.0	https://sourceforge.net/projects/boost/files/boost/1.62.0/	http://www.boost.org/doc/libs/1_55_0/more/getting_started/windows.html
5	CMake	Build external Libraries	CMake 3.7.0	https://cmake.org/download/	Executable file available

Table 1.1: Software Installations

After each of the above mentioned software is installed:

1. Create a folder for the project and create two sub folders “src”, for source files and “bin”, for the binaries.
2. Add the .cpp and .h files in the src folder and rename change the extension of .cpp files to .cxx.
3. Create a new text file in the src folder and name it as “CMakeLists.txt”. Add the following to the file:

```
cmake_minimum_required(VERSION 2.6)

project(Clustering)

file( GLOB SRCS *.cxx *.h)

find_package(VTK REQUIRED)

include(${VTK_USE_FILE})

add_executable(Clustering Clustering.cxx Functions.cxx)

target_link_libraries(Clustering ${VTK_LIBRARIES})
```

4. Open Cmake, add src’s path in source and bin’s path in destination. Configure and generate.
5. After this is done, open the bin folder, select an dopen the .sln file to open the project in visual studio.
6. After the project is open in visual studio, in solution explorer, right click on the name of the main project and set it as the first project. Build and Run.

1.3 Chapter Reviews

This thesis document has been divided into five different chapters, the first chapter being the introduction. Following is a short summary of the remaining chapters:

Chapter 2 is Dataset and Preprocessing. In this chapter, the dataset used for the 3D point matching has been discussed. We will talk about the different facial meshes we have, the data organization and the challenges faced while dealing with the data. We also discuss the preprocessing that is done before the mesh is worked on. For the preprocessing, we smoothen the mesh in order to improve the mesh quality and make it ready to be worked on.

Chapter 3 is the Point matching system for 3D faces. In this chapter, the actual pipeline of the current system has been described. This chapter has six sub sections, Tip of nose, Top of forehead, Left and Right Ear, MeshSIFT, Thin Plate Spline and Iterative Closest Point, each section being a vital component and are arranged in the order they are being used in the actual implementation of the system. MeshSIFT, Thin Plate Spline and Iterative Closest Point are open source algorithms which have been used to implement this system and are described in detail in this chapter.

Chapter 4 is Results and Discussion. In this Chapter we show the results of all the approaches that were experimented to implement this system. Subsection 4.1 shows the results of the initial approaches which were considered and implemented. Subsection 4.2 shows the results of the current system step by step, following the pipeline in the same order as in Chapter 3. In subsection 4.3, we analyze the errors and discuss the outcomes of the current approach. In subsection 4.4, we experiment the change in processing time and the number of point

matchings obtained by MeshSIFT before and after mesh simplification. The meshes are simplified to one eighth of their original size using an open source software called MeshLab [22].

Chapter 5 is the conclusion. In this chapter, the entire study has been summed up and the outcomes have been discussed. We also discuss the future work and improvements which can possibly eliminate the shortcomings of the current approach and make the system even better.

Chapter 2- Dataset and Preprocessing

2.1 Dataset

The BJUT-3D Large-Scale Chinese Face Database from Multimedia and Intelligent Software Technology Beijing Municipal Key Laboratory, Beijing University of Technology [5] has been used for this study, this face database is made available for research purpose only. A CyberWare Laser Scanner has been used to obtain the 3D models. A lot of efforts have been put into the scanning process so that the 3D face captured is of the best quality. The scanner is positioned strategically so that it does not cast any shadows on the human faces. Illumination has been taken into consideration, a consistent ambient illumination is maintained throughout the scanning process.

The database consists of 500 3D face models of Chinese people, out of which 250 are males and 250 females. The people whose faces are scanned are between the age group of 16 and 49. All faces pose a natural expression and without any glasses, dressing or accessories. All the subjects were made to wear a hat to avoid having hair on the forehead, eyes and ears and capture maximum amount of face information. Figure 2.1 shows a male and a female 3D face from the database.

Figure 2.1 Male and Female face from the database

The database also follows a naming convention which is explained as follows:

x_xxxx_Ax_Ex_Cxxxx_Rx.ext

- Here the first x is of 1 bit and corresponds to the sex field where “M” is male and “F” is female.
- xxxx corresponds to a 4 bit ID which is a number sequence unique to each person scanned.
- Ax is of 2 bits and corresponds to age. Here, “A” represents age variation. Symbol 1 represents age group 10 to 19 year, 2 is for the age group of 20 to 29 years, 3 if for 30 to 39 years and 4 is for the ages of 40 to 49 years.

- Ex is of 3 bits and represents extression. “E” represents expression variation and the seond bit id for different expressions. Symbol “n” stands for natural, “h” for happy, “p” for surprise and “a” for anger. In current database, all the faces scanned have a natural expression.
- Cxxxx is of 5 bit and for content. “C” represents content. The remaining 4 bits “xxxx” are “trim” which indicated that the face is presented after removing the redundant parts of the data.
- Rx is of 2 bits and represents publicity. “R” is for publicity and “x” is “0” if data is unpublished and “1” if published.
- ext is a 3 bit extension name which indicates the file format.

Examples of this naming convention are “F0001A1EnCtrimR0”, “F0111A1EnCtrimR0”, and “M0032A2EnCtrimR1” etc.

Each 3D face file consists of vertex information, i.e. the x, y and z coordinates, texture information which is the R, G, B values and the Triangle or Face information which is the indices of the 3 points with constitute a triangle for mesh formation. Unfortunately, the representation of this data is not standard. Due to this we convert the data obtained by the database into OFF format. OFF is an object file format used to represent the geometry of a model. It specifies the polygons on the modal’s surface and the polygons can have any number of vertices. An OFF file is ideal for storing a 2D or 3D object description for the object constructed from polygons and a simple extension to it can handle objects in 4D as well. Following is an example of the OFF file format:

OFF

No. of Vertices No of Faces No. of Edges

vx1 vy1 vz1

vx2 vy2 vz2

……..

vxn vyn vzn

//Triangles

3 v11 v12 v13

3 v21 v22 v23

……

3 vm1 vm2 vm3

In the above format “n” stands for the total number of vertices and “m” is for the total number of faces in the mesh. Providing the “No. of Edges” is optional and the texture information can be provided as RGB values along with each vertex, eg: vx1 vy1 vz1 r g b.

Figure 2.2 represents a zoomed in mesh where the mesh triangulation/faces of the mesh can be seen.

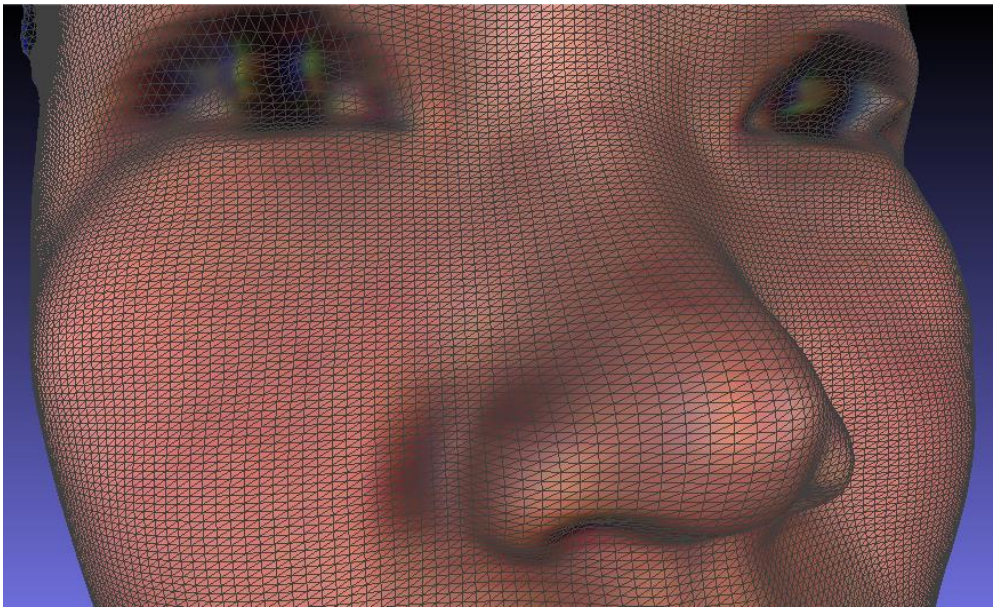


Figure 2.2 Mesh Triangulation

There have also been a lot challenges while dealing with the data. In this database, the total number of vertices are different for each face and there is no correspondence between the modals which makes it difficult to work with. There is also a lot of error in in some faces. Some faces have neck portion and some don't and some faces have missing features and some have very irregular surfaces. Figure 2.2 shows a few noisy faces.

Figure 2.3 Noisy Faces

Another issue with this database is that the scale of 3D faces is very limited. This database features faces only of oriental origin. There is a considerable difference between the face structure and features of western and oriental and may not provide all practical requirements and enough 3D faces for a global application. However, there are other 3D face databases which are based on western face structures.

2.2 Preprocessing Data

In order to obtain better results, all the meshes have been smoothened before any other operation. Taubin mesh smoothening has been used to accomplish this operation. In this system implementation, we use 10 iterations to smooth the surface of the mesh. We shrink the mesh first using lambda and then inflate it using mu, where both lambda and mu are scaling factors. This is done to avoid mesh shrinkage and loss of features during smoothing. The pseudocode for mesh smoothing used in the system is as follows:

1) For each iteration (We are using 10 iterations).
2) For each point P in the mesh
3) $P_i = P_i + \lambda \Delta P_i$ // for Shrinking
4) $P_i = P_i + \mu \Delta P_i$ // for inflation
5) Repeat Steps 2 to 4 for each point in the mesh.

For this system implementation, we have set lambda value to be 0.5 and mu value to be -0.51.

Figure 2.4A and 2.4B show the result of smoothing on the given mesh. Figure 2.5 represents the difference between the results of point matching with and without smoothing. Clearly, the results with smoothing are better.

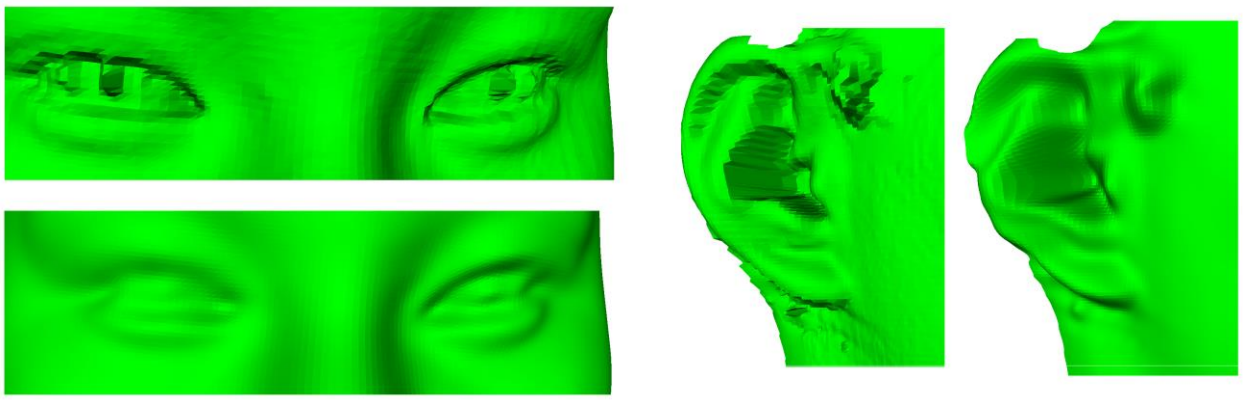


Figure 2.4A: Mesh Smoothing

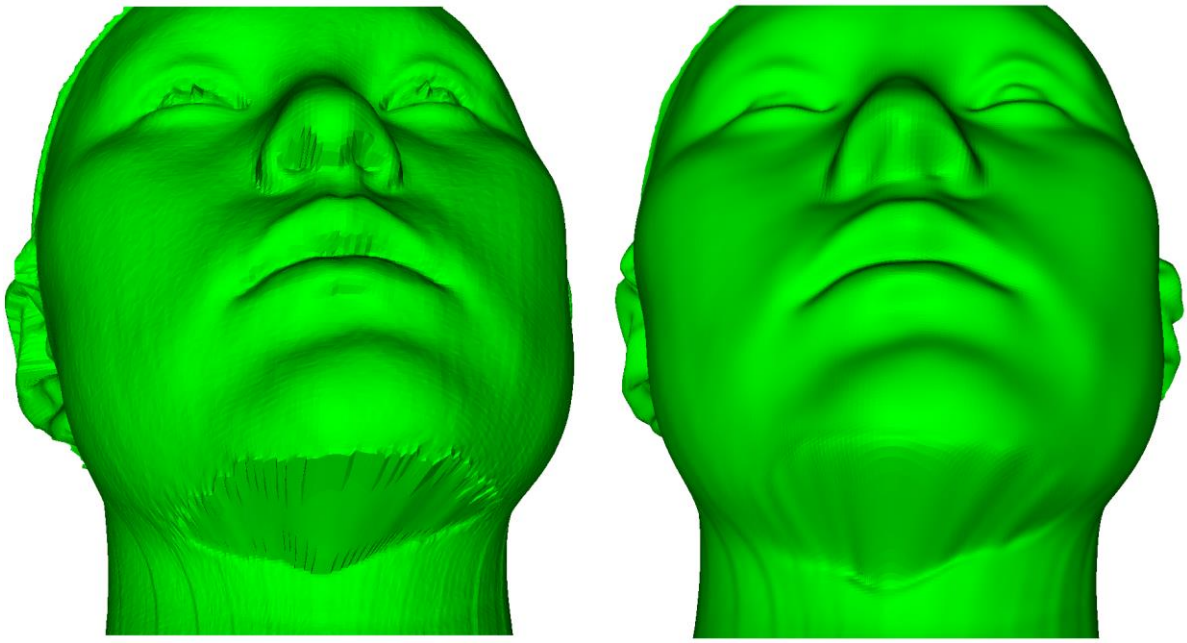


Figure 2.4B: Mesh Smoothing

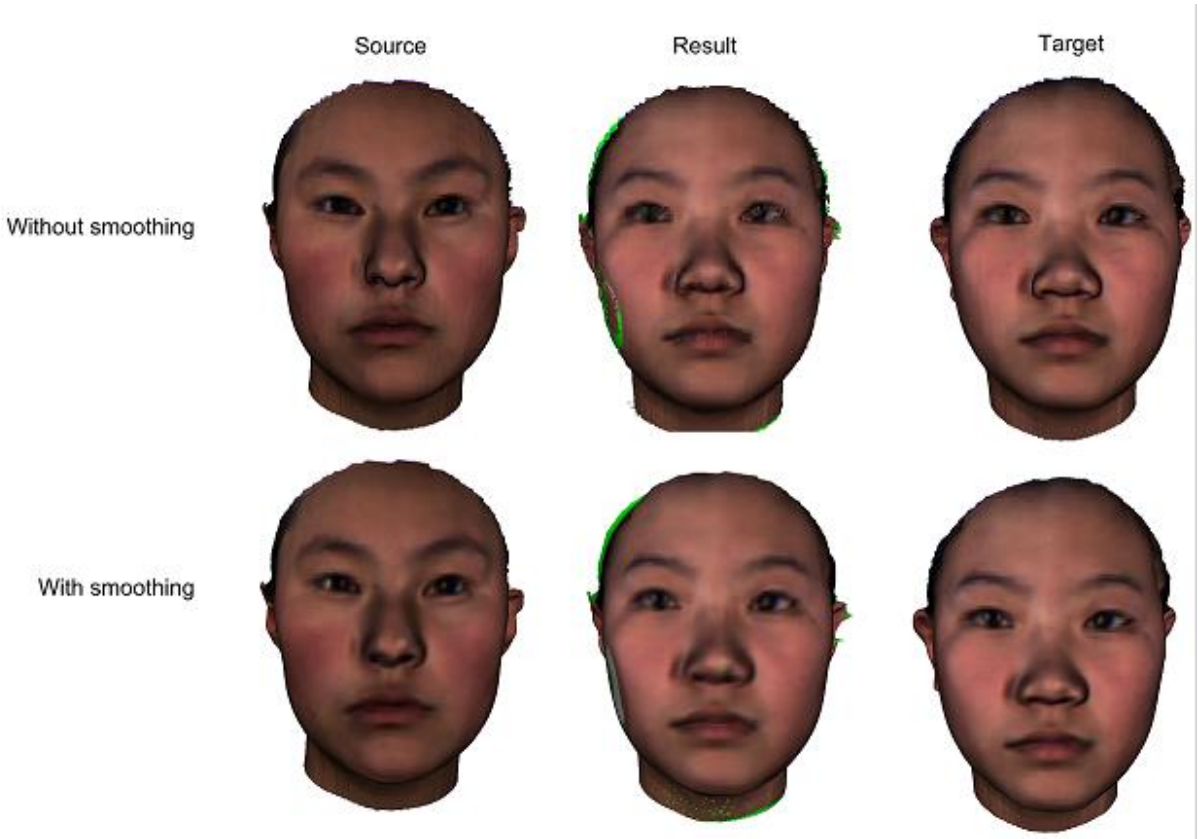
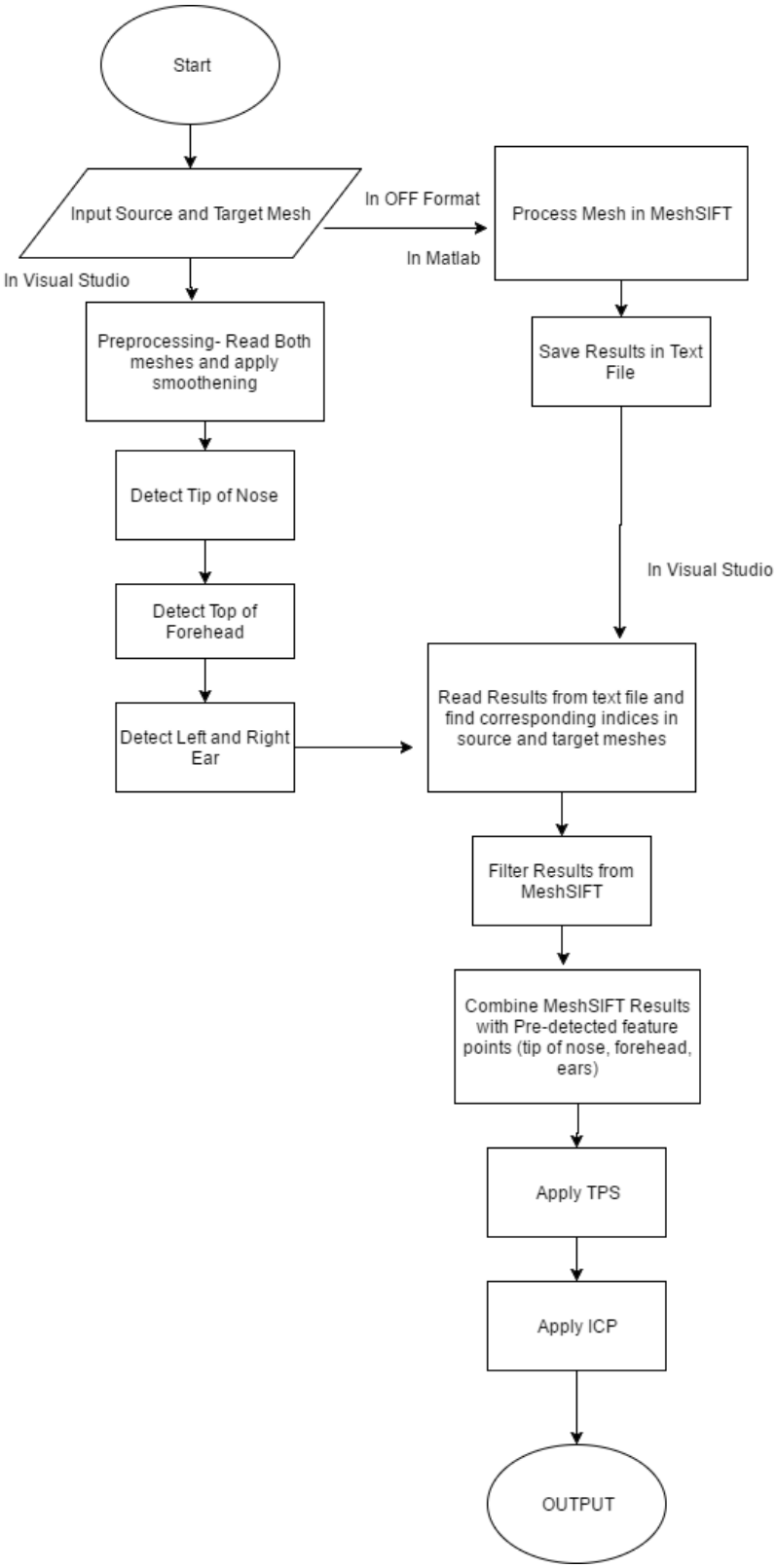


Figure 2.5: Point Matching Results with and without Mesh Smoothing

Chapter 3- Point Matching System for 3D Faces

This chapter is the actual pipeline of the implementation of the point matching system in the order of processing. Before proceeding to using meshSIFT, which is an open source algorithm used to obtain the point matchings of the 3D faces, we detect some facial features including the tip of the nose, the top of the forehead, the left ear and the right ear beforehand. This has made the point matchings significantly better and have been a tremendous help in getting good results.

Here is a flowchart of the Algorithm pipeline:


Start
Input Source and Target Mesh
In OFF Format
In Matlab
Process Mesh in MeshSIFT
In Visual Studio
Preprocessing- Read Both meshes and apply smoothening
Detect Tip of Nose
Detect Top of Forehead
Detect Left and Right Ear
Save Results in Text File
In Visual Studio
Read Results from text file and find corresponding indices in source and target meshes
Filter Results from MeshSIFT
Combine MeshSIFT Results with Pre-detected feature points (tip of nose, forehead, ears)
Apply TPS
Apply ICP
OUTPUT

3.1 Tip of Nose

The first step to automatic point matching was to figure out a way to detect high feature points, which could be eyes, nose, chin, forehead, ears, center of eyebrows, lips etc. Out of all these, nose was the most straightforward detection. In general terms, nose is the most protruding part of the human face. To leverage this fact, the center of the face is aligned on the origin for x and y coordinates and the z coordinate is set to 50, this is done to orient the face modal towards the viewer and set the viewing position such that the face faces towards positive z direction. Following is the pseudocode used for the detection of the tip of the nose.

1. let maxZ be -1000;
2. let index be -1
3. for each point in the mesh.
4. If the z coordinate of the point on mesh is greater than maxZ, update maxZ to z and set index to be the index of the point on the mesh.
5. Repeat step 3 an 4 until all the points on mesh have been traversed.
6. Set the final value of index to be the tip of the nose.

This algorithm selects the point in maximum z direction to be the tip of the nose. Figure 3.1 shows the detected tip of the nose.

Figure 3.1: Detected tip of Nose

3.2 Top of Forehead

To get better matchings, it is critical that there are points on the boundaries of the source mesh and target mesh which can be matched to overlap the meshes better. The top of the forehead is one of those critical points. Detecting the top of forehead was challenging. The approach used to detect the forehead was to start from the tip of the nose and continue going in the positive y direction until we reach the edge of the mesh, this will result in a point which is collinear to the tip of the nose and lies on the topmost part of the forehead. This working of this approach mainly depends on the mesh triangulation. If we start from nose and keep selecting the point with minimum distance from the x coordinate of the tip of the nose and maximum distance from the y coordinate of the tip of the nose, the result could be a point

which is too far from being at the central tip of the forehead. To overcome this issue, the search area to choose for the best neighbor had to be minimized so that the search direction is accurate, to accomplish this, an array of linked list of point neighbors is maintained and only the points in the linked list are considered for the search and the best qualified neighbor is selected Below is the pseudocode for the detection of the top of forehead.

1) Check if the tip of nose has been detected, it is critical to detect the tip nose before proceeding to detect the forehead as it's the starting point for the search. If not, return with message explanation.
2) select the next neighbor of the tip of the nose
3) If the difference between selected neighbor's x coordinate and tip of the nose's x coordinate is less than a threshold value (which is 1 in our case), and y coordinate is greater than the y of the last qualified neighbor, set this point as the last qualified neighbor.
4) Select the next neighbor of the last qualified neighbor and repeat step 3 until the highest neighbor is reached.

Figure 3.2 white color represents the neighbors selected in the process of finding the top of forehead, and yellow is final point set to be the top of forehead.

Figure 3.2: Top of Forehead

3.3 Left and Right Ear

Another set of critical points that needed to be detected for better boundary overlap of both meshes were the points on left and right ear. Detecting these points was also a challenging task because of some errors in the face data models. The initial approach to detect points on the left and right ear was to find the bottom most points of the left and the right earlobes. This was accomplished by detecting the leftmost and the rightmost point on the face with maximum z value and a negative normal direction. Hence, the target was to detect points on earlobes facing the earth. This approach didn't work very successfully because of some nose in the dataset. Some faces had a bulge right under an ear due to scanning, since we are targeting the

leftmost and rightmost points with highest z and negative normal direction, the point ended up being detected on the bulge.

To avoid this, another approach was implemented. The tip of the nose needs to be detected beforehand for this this approach to work. In this approach, we look for only those points whose y coordinates are under a certain threshold distance from the origin, threshold range considered here is 0.5 to -0.5, and this is because we set the center of the face at origin. We set the initial left and right ear points at the top of the nose and then keep moving right and left until the mesh boundary is encountered. Following is the pseudocode for this approach.

1. Let the leftEarIndex and rightEarIndex equal to the index of the tip of the nose.
2. For each point in the mesh
3. If y coordinate of the selected point is between the threshold range of 0.5 and -0.5, continue to steps 4 and 5.
4. If x coordinate of selected point is less than x coordinate of leftEarIndex, set current point as leftEarIndex.
5. If x coordinate of the selected pointed greater than the x coordinate of rightEarIndex, set the current point as rightEarIndex.
6. Repeat steps 3 to 5 until all points on the face have been traversed.

Figure 3.3 represent the points considered in order to find the correct points on the left and right ear. The final selected points are colored yellow.

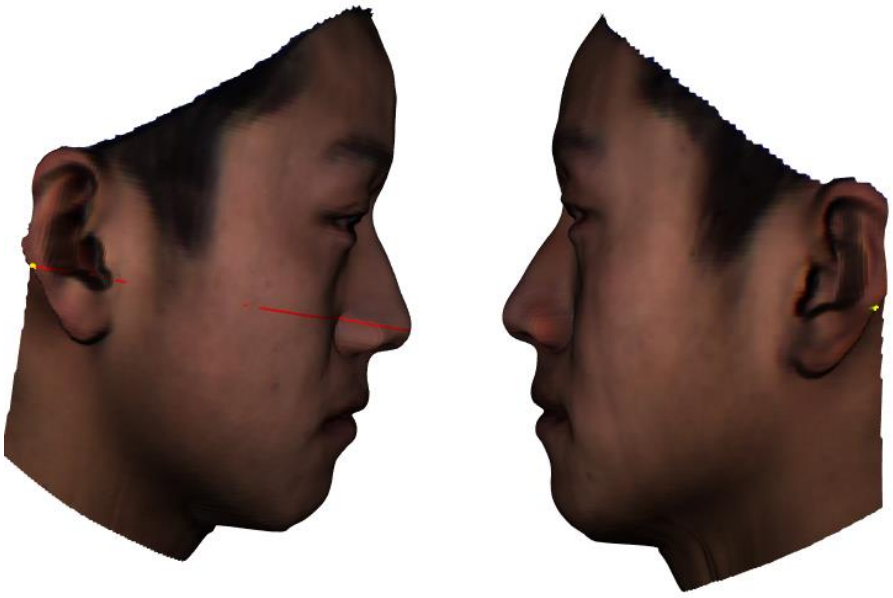


Figure 3.3 Left and Right Ear

3.4 MeshSIFT

MeshSIFT is an open source algorithm [6] which detects salient points on 3 dimensional face data based on the geometry of the face. This algorithm is an extension of the Scale Invariant Feature Transform (SIFT) algorithm which works for 2D data, meshSIFT has the added ability to deal with 3D data. MeshSIFT has four major components:

1. **Key-point detection**: It detects the salient points on the mesh. It constructs a scale space which consists of the smoothed input Mesh M_S. It is expressed as:

$$M_s = \begin{cases} M, & s = 0 \\ G_{\sigma_s} \otimes M, & else \end{cases}$$

Here, M refers to the original mesh and G_{σ_s} refers to the approximated Gaussian filter with the scale σ_s. Then mean curvature is computed for each vertex at each scale of the scale space to detect the salient points.

2. **Orientation assignment**: Each key-point is assigned a canonical orientation to obtain an orientation invariant descriptor. This makes the point detection independent of the facial pose. To ensure a scale invariant descriptor, neighborhood size is expressed as a function of scale σ_s.This limits the consideration of vertices around a key points within a spherical region of radius $9\sigma_s$.

   The normal vector for each vertex in consideration is computed and based on the fast marching algorithm, a geodisc distance to the respective keypoint is determined. After this, the normal vectors projected are then collected in 360 bin weighted histogram. Gaussian weight with its geodisc distance to key point with a proportional bandwidth to the scale assigned for each histogram entry is calculated. At last the highest peak and every peak above 80% of the highest peak value in the histogram is selected to be the canonical orientation. This canonical orientation is then calculated to sub-bin precision by fitting a quadratic function to the histogram using the neighbor bins of a peak. This leads to having a canonical orientation for multiple key points.

3. **Local feature description:** is basically the local neighborhood around a key point. Each key point's geodisc radius contains 9 small circular regions with respect to the canonical orientation. Feature descriptor is created based on local neighborhood which provides a feature vector around each key point.

   Feature vector consists of concatenated histograms which are calculated over 9 neighborhood regions. For each of the 9 regions, two histograms $\hat{p}_s$ and $\hat{p}_\theta$ are computed. Here, $\hat{p}_s$ contains shape index which is calculated based on the curvature and $\hat{p}_\theta$ consists of slant angles which are the angles between every projected normal

and canonical orientations. The resultant histograms are weighted based on distance to the center of the region. Additionally, shape index histograms are weighted on curvature and slant angle histogram based on tilted angle, which is the angle between the considered vertex and the normal in the keypoint. Every histogram is then normalized to 1/sqrt(8) to reduce the influence of large histogram values. The resultant histograms are concatenated to a single feature vector shown in the following equation.

$$f_i = [\hat{p}_{s,1}, \hat{p}_{\theta,1}, \ldots., \hat{p}_{s,9}, \hat{p}_{\theta,9}]^T$$

This way a feature descriptor is computed for each keypoint, resulting in a set of feature vectors per face. The following equation describes a feature descriptor F:

$$F = \{f_1, f_{2,} \ldots. f_n\}$$

Feature matching, the selected features of the source mesh and the target are then compared using angle. This angle is defined as:

$$\alpha = \cos^{-1}\left(\frac{< f_i, f_j >}{||f_i||\ \left|\left|f_j\right|\right|}\right)$$

Angles for all candidates for each feature are ranked in the ascending order. If the ratio between the angles is smaller than a threshold, the match is accepted, this is done for both the meshes.

MeshSIFT code has been taken from online, it is open source and free to use. This code runs on Matlab. In the current system implementation, the vertex and face information of both the source and target mesh is loaded. MeshSIFT returns a matrix of matching points in both the meshes. The number of matched points obtained can be different for different meshes.

Figure 3.4 represents an example result of the point matchings from meshSIFT.

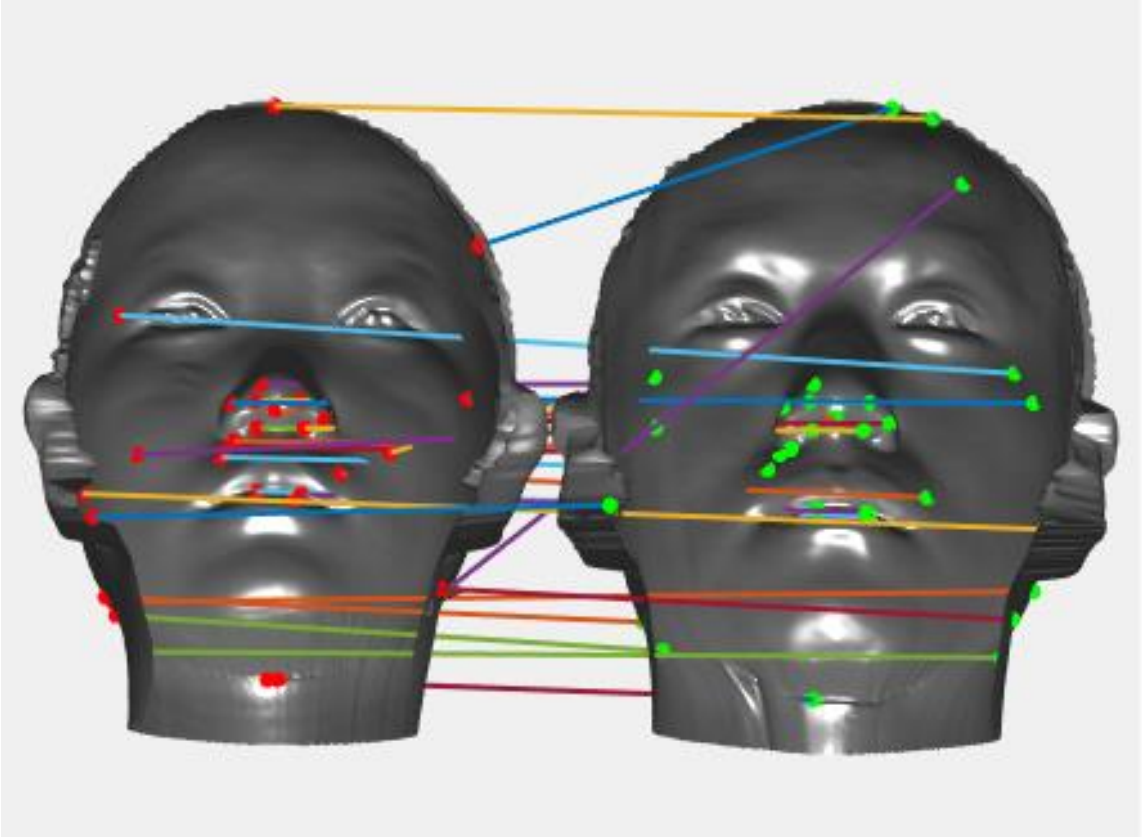


Figure 3.4 Result from MeshSIFT

However, the points obtained by meshSIFT are not very accurate and not spread evenly throughout the face. The matchings obtained are then filtered using a certain threshold value which is different for different meshes as the number of point matching obtained are also different. For this system, a threshold value of (10, 15 and 12 have been considered for different meshes). Filtering using a threshold value gets rid of the incorrect point matchings. The result from this filtering is then combined with the point matchings obtained from detecting the tip of the nose, the top of the forehead and the left and right ear and fed as input to TPS.

3.5 TPS

Thin Plate Spline is an open source interpolation method [7]. It results in a minimal bent surface that passes through all the given control points. The actual algorithm is made for 2D data, but it has been modified to work with three dimensional data. TPS can capture two affine and non-affine transformations.

Formula used by TPS:

$$\mathrm{f(P)} = \mathrm{Pd} + \mathrm{Kw}$$

Here, P is the point coordinates on source mesh, d represents affine transformations and w represents non-affine transformations. d and $w$are obtained by the following equation:

$$\begin{pmatrix} K & M_1 \\ M_0^T & 0 \end{pmatrix} \begin{pmatrix} w \\ d \end{pmatrix} = \begin{pmatrix} M_2 \\ 0 \end{pmatrix}$$

Here, K is a $M \times M$ matrix where M is the number of selected control points and $M1$ and $M2$ represent the coefficients of control points on source and target meshes, respectively.

Thin Plate Spline deforms and smoothen the source mesh to take the shape of target mesh. This part of the pipeline has been taken from the pre-existing implementation of the system. Running TPS on source mesh leads to the change in geometry of the source face so that it looks like target while the topology is still the same.

For TPS to work correctly, the points selected to be control points need to be very accurate, otherwise, the result can be a disaster. Figure 3.5 is an example to demonstrate how TPS is applied to source mesh to change its geometry to the target.

Figure 3.5: Example of TPS

3.6 ICP

Iterative closest point also known as ICP is an open source algorithm [8] which is used to bring two point clouds closer to each other and minimize the distance between them. In our system implementation, the target is fixed and the source mesh is transformed so that it matches the target mesh. We have deployed five iterations of ICP for each mapping. With each iteration, the distance between source and target is reduced, hence minimizing the error. Following is the equation used for ICP, our main goal here is to minimize the value of E_g as much as possible:

$$E_g = \sum_{i=1}^{n} \omega_i |\tilde{v}_i - g_i|^2$$

Where, ω_i is a weight for this system implementation, we have set its value to be 1/5. g_i is the intersection with the target mesh along point $\tilde{v}_i'$s normal. $\tilde{v}_i$ represents points of source mesh.

Figure 3.6 represents three iterations of Iterative closest point on source mesh. The final result along with the target mesh is also displayed.

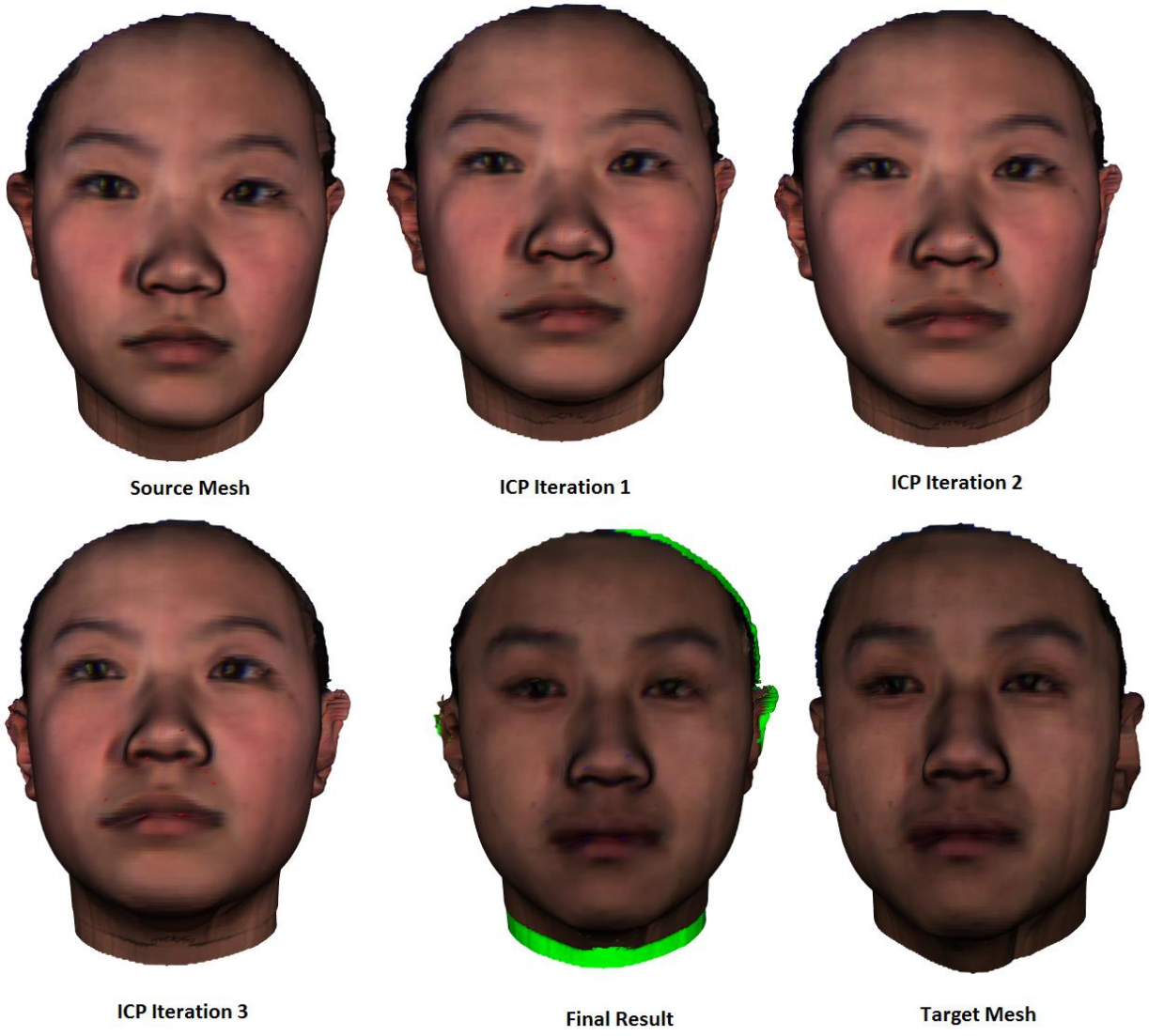


Figure 3.6: Iterative Closest Point Matching

Chapter 4 - Results and Discussion

4.1 Initial Approaches

4.1.1 Constraint K-Means Algorithm

For this approach, 122 best faces were selected from the entire database out of which, 85 faces were categorized into training set and 37 as testing set.

For the faces in training set, landmarks of the top of forehead, center of eyebrows, chin, left ear and right ear were selected manually and the position of each landmark selected was recorded. The same landmarks were selected for all the faces. After that, the mean value of the positions of all the landmarks were set as cluster seeds for the testing set.

After implementing this algorithm, it was found that the results were not as good as expected. Figures 4.1A to 4.1C show the results obtained from the constraint K-Means. The center of eye, ear and chin detection are not very accurate.

Figure 4.1A: Results- Constrained K-Means

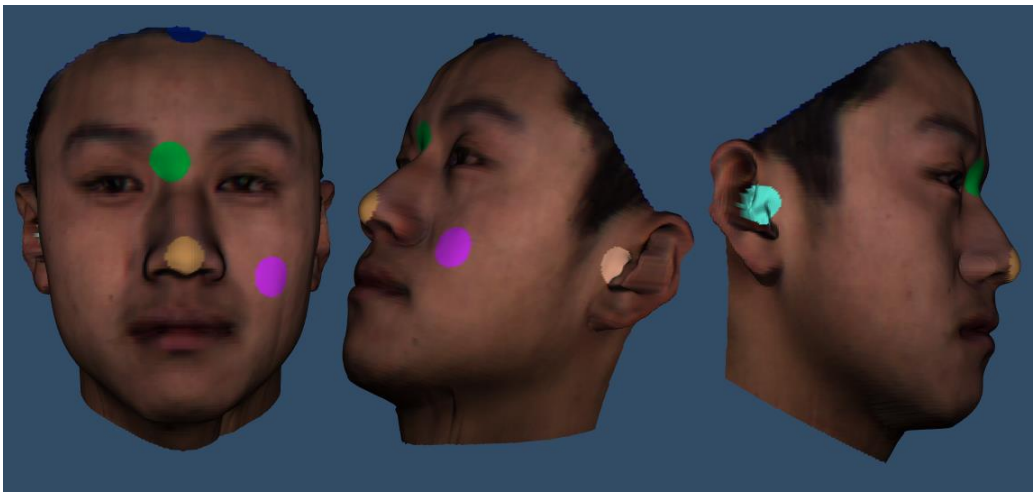


Figure 4.1B: Results- Constraint KMeans Algorithm

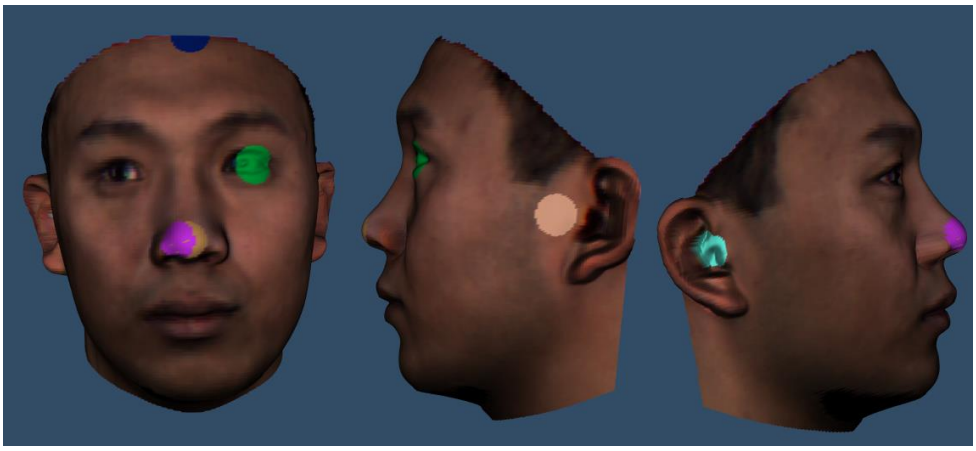


Figure 4.1C: Results- Constraint KMeans Algorithm

4.1.2 KMeans

For this algorithm, the entire mesh was clustered totally unsupervised. The number of clusters to be formed were input by the users and the cluster seeds were initially chosen at random. For each point, its Euclidean distance with respect to each cluster seeds was calculated and the point was assigned to the closest cluster. After the point had been assigned to the cluster, a new mean of the cluster was calculated and the cluster seed value was updated. These steps were repeated until all the points in the mesh had been traversed.

After implementing this algorithm and considering different number of clusters, the following results were obtained. Again, the results were not good enough, we can see that the clusters are not well defined. Figure 4.2A and 4.2B depict the results with different number of clusters.

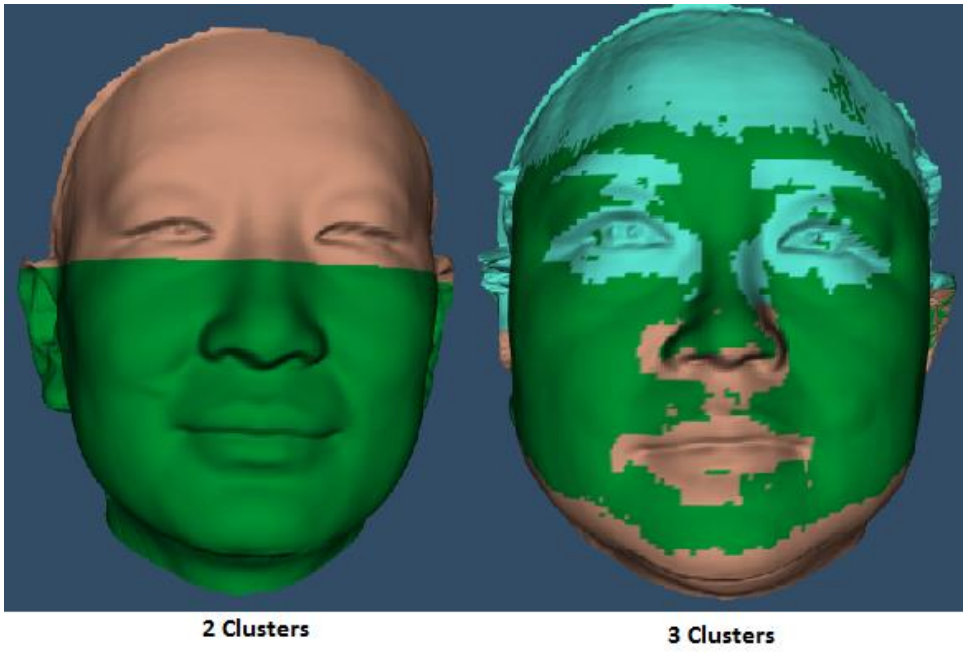


Figure 4.2A: Results- KMeans Clustering with 2 and 3 Clusters

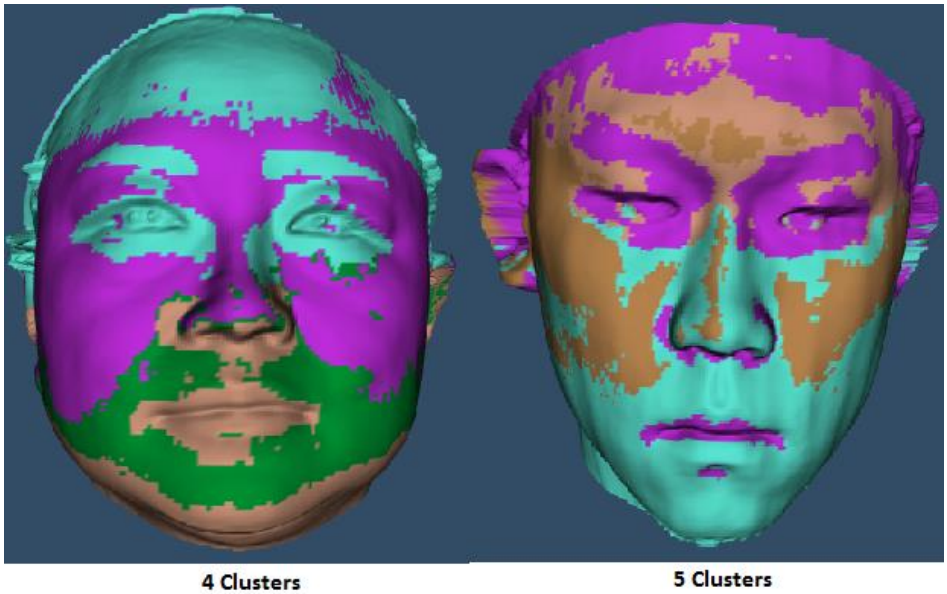


Figure 4.2B: Results- KMeans Clustering with 4 and 5 Clusters

4.1.3 Modified KMeans

The initial steps of this algorithm are the same as that of kmeans, the algorithm discussed in section 4.1.3, the difference is that while determining the cluster of a point, additional weight is given to certain features. The first experiment was done by increasing the weight of the curvature to twice its value and RGB values were multiplied by 1.5. This was done in order to cluster features with high curvature value. The second experiment was to give additional weights to normals and RGB values, both were multiplied by 1.5. Hence, the Euclidean distance of a point from the center of each cluster was calculated based on these modified values to determine the cluster for the point. After the point is assigned to the cluster, the cluster's mean is calculated and assigned as the seed of that cluster.

The results obtained by this algorithm weren't good enough either as there was not much uniformity in the clusters. Figure 4.3A and 4.3B depict the results of this approach.

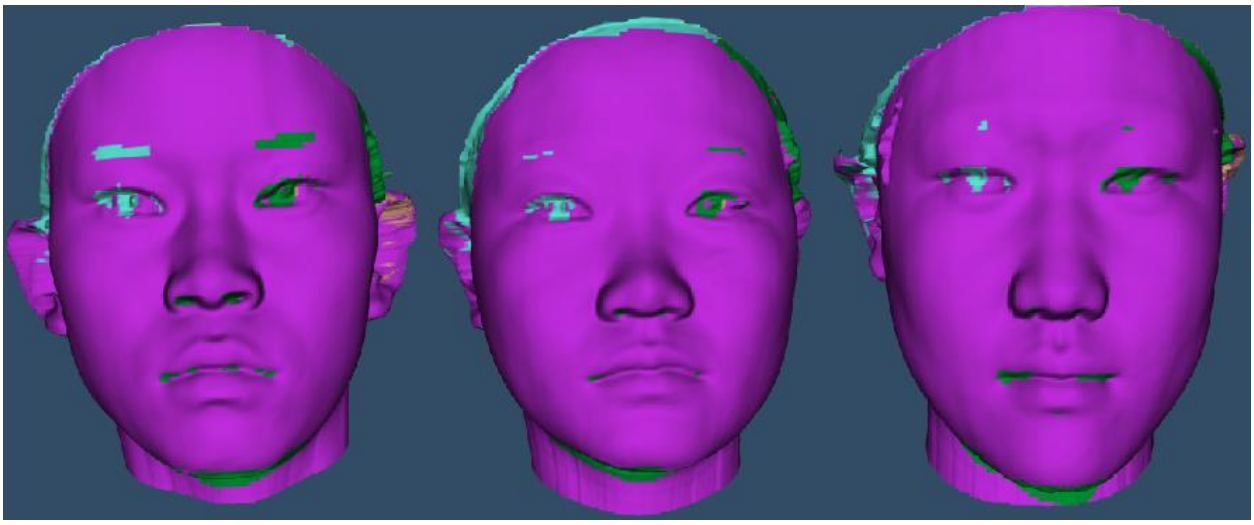


Figure 4.3A: Results- KMeans with higher weights assigned to Curvature and RGB

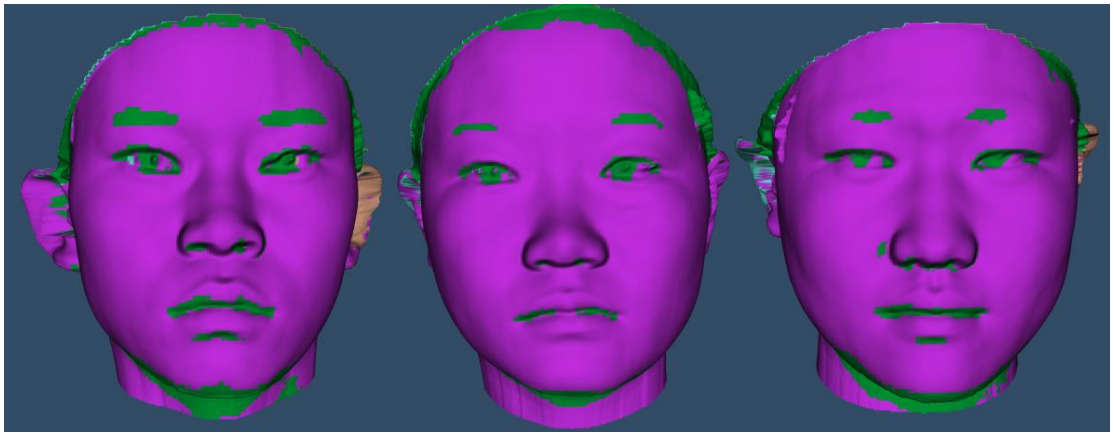


Figure 4.3B: Results- KMeans with higher weights assigned to Normals and RGB

4.2 Current Approach

The results obtained by previously discussed approaches weren't acceptable because the clusters obtained were not uniform. In order to obtain good results, accurate point matchings need to be input for TPS, otherwise, the face geometry is not modified correctly. Also, the final result from ICP depends on the deformed mesh obtained from TPS, hence, very accurate point matchings are critical for good point matchings. To solve this problem, the current approach is deployed.

Current approach is a pipeline of such algorithms as nose, forehead and ear detection, meshSIFT, TPS and ICP.

4.2.1 Tip of Nose

Results obtained by detecting the tip of the nose are very accurate and the tip of the nose has been detected correctly for all the faces. In order to detect the nose tip, the face needs to be

aligned, here, the mesh is centered at origin of the x and y plane and z plane is set as 50. Figure 4.4 represents the tip of the nose detected, highlighted in white color.

Figure 4.4: Tip of nose detection- represented by the white point

4.2.2 Top of Forehead

The top of forehead detection has also been accurate for all the faces. For this approach to work, the tip of the nose should be detected and the neighbors for all the points need to be calculated and stored in an array of linked list beforehand. During the search process, the difference between the x coordinate of the candidate point and the x coordinate of the tip of the nose should be under a threshold value of 1, so that the search direction remains vertical from the tip of the nose. Figure 4.5 highlights the route from the tip of the nose to the top of the forehead, the green line is formed by the best matched neighbor points.

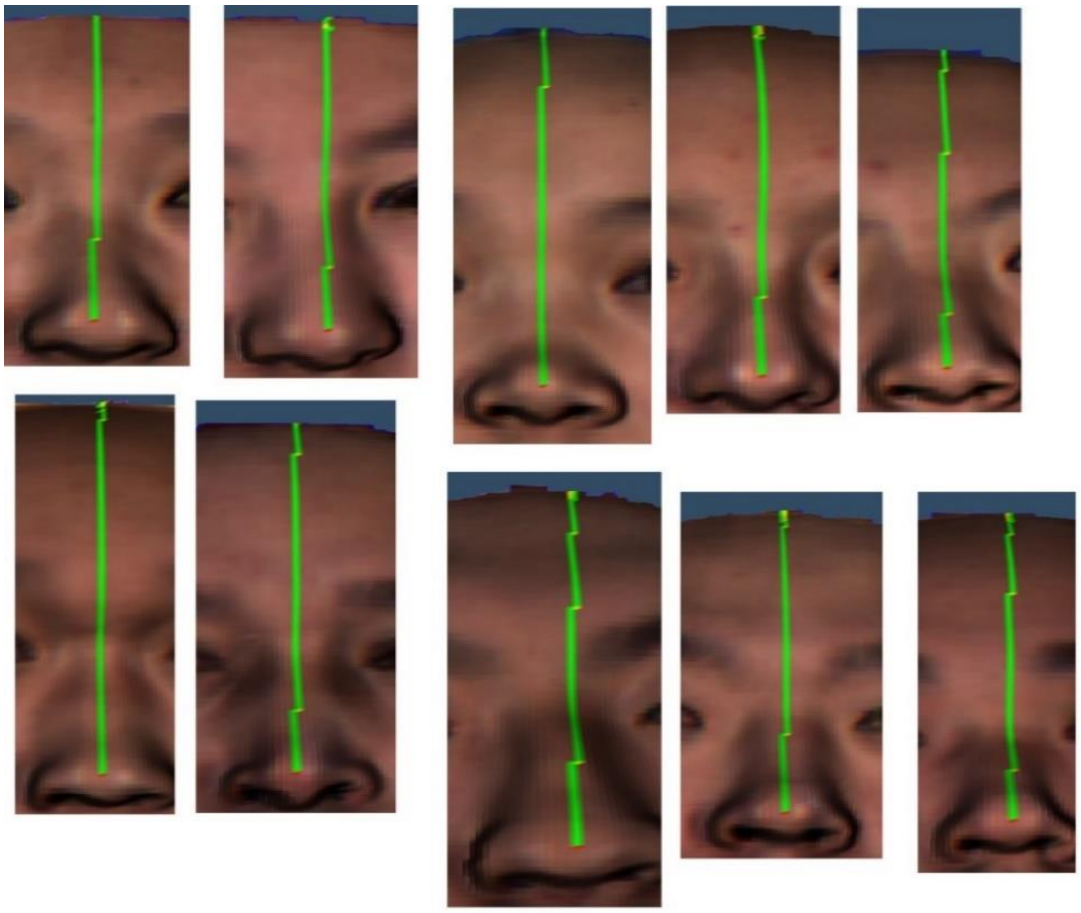


Figure 4.5: Best Matched Neighbors from tip of nose to top of forehead

4.2.3 Left and Right Ear

In order to detect the left and right ear, the tip of the nose should be detected beforehand. During the search process, the points have been filtered based of max and min x coordinated for right and left ear respectively because the face is centered at the origin and the left most point on left ear will have the lowest x value and right most point on the right ear will have the highest x coordinate. Also, to make sure that the search remains parallel to the x axis, the points have been filtered based on a threshold range of -0.5 to 0.5 for the y coordinate, i.e. the point is considered to be a candidate only if its y coordinate value is between 0.5 and -0.5. The

current approach gives good results, but sometimes, if the face geometry of two faces is too different, the points detected on the left and right ear on source can be a bit too far from that of the target. Hence, a better approach can be devised. Figure 4.6 represents the points detected on left and right ear, highlighted in yellow color.

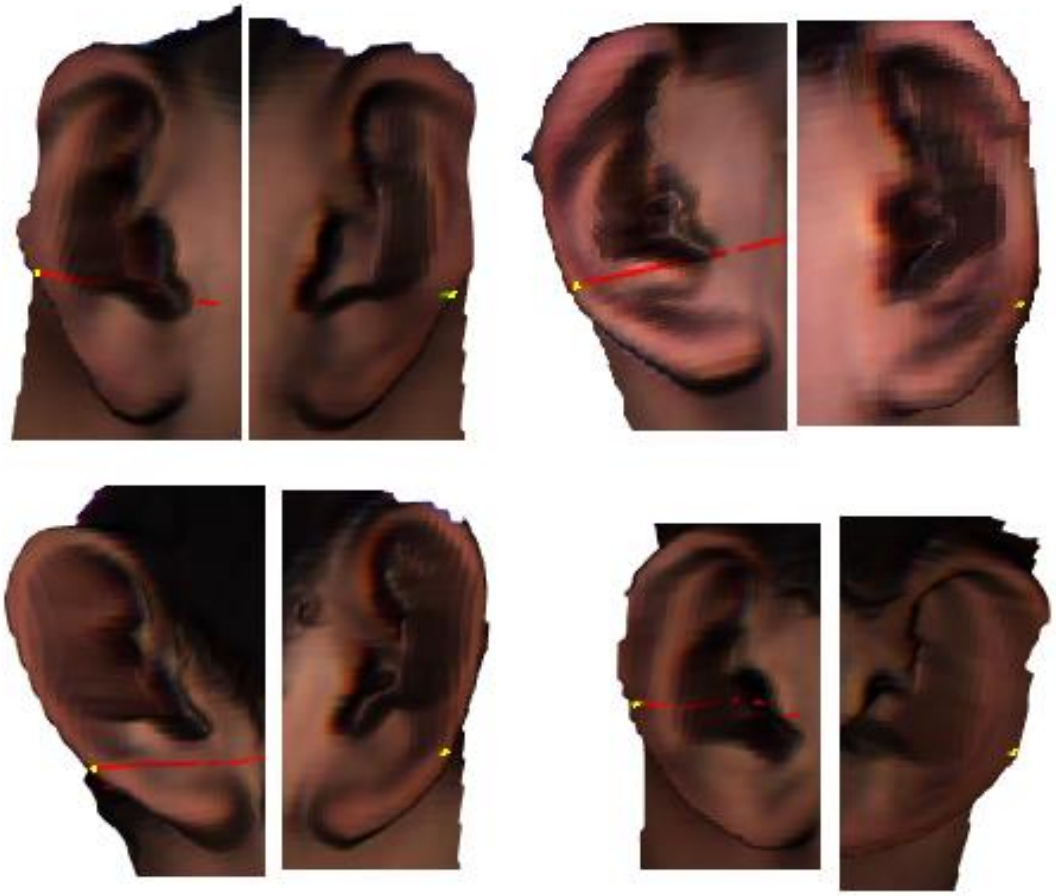


Figure 4.6: Points detected on left and right ear

4.2.4 MeshSIFT

Figure 4.2.4.1 to 4.2.4.3 represent the points matched by meshSIFT. Figure 4.7A and 4.7B are rear views of the faces and 4.7C is the front view. In these figures, all the point matchings obtained are not correct, some matchings are mirror images and some are simply incorrect. Due to this reason, the matchings obtained from meshSIFT are filtered based on a certain threshold of Euclidian distance between the matched points. The threshold varies from mesh to mesh because of difference in the number of points. For the current system implementation, the thresholds taken are either 10, 15 or 12.

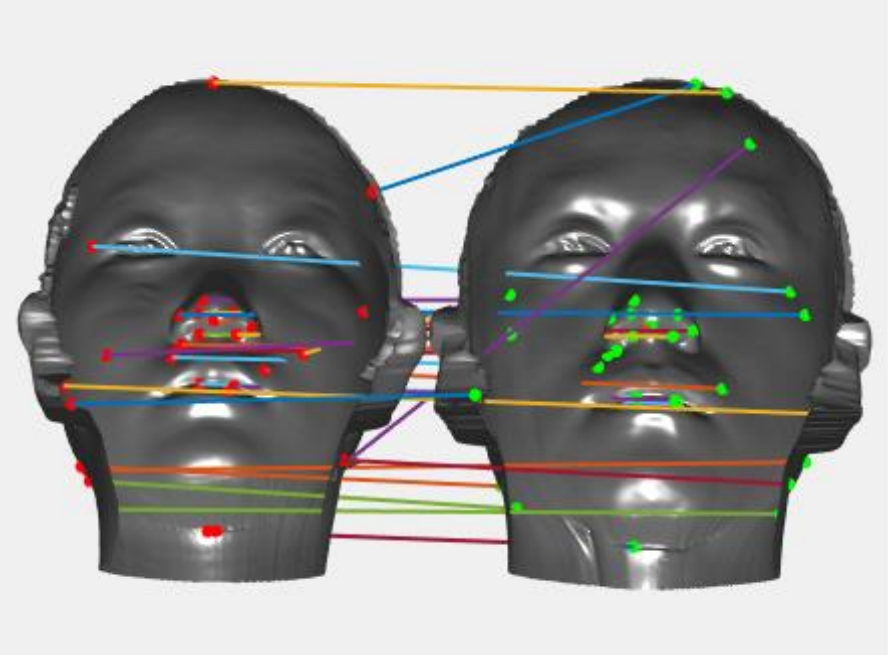


Figure 4.7A: Points matched by MeshSIFT

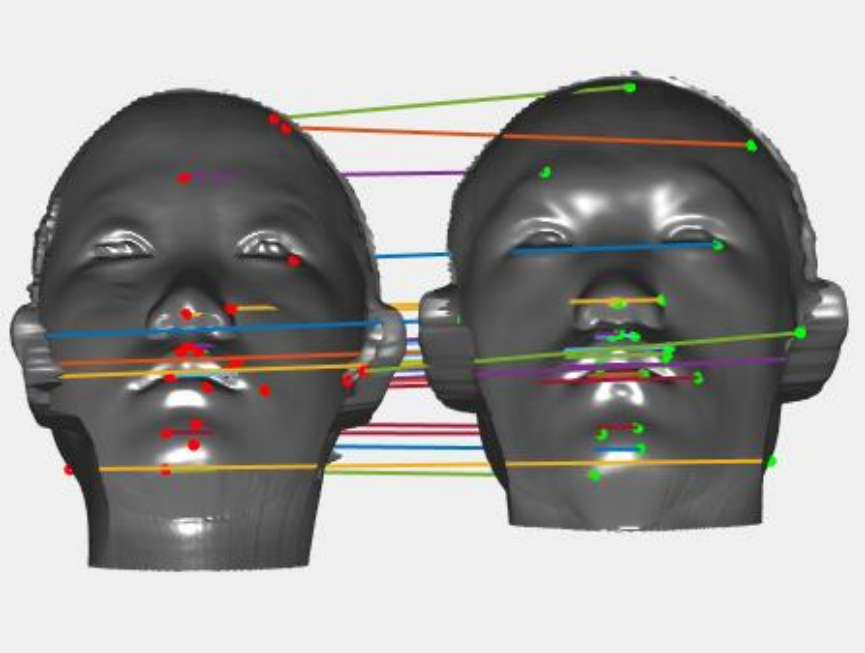


Figure 4.7B: Points matched by MeshSIFT

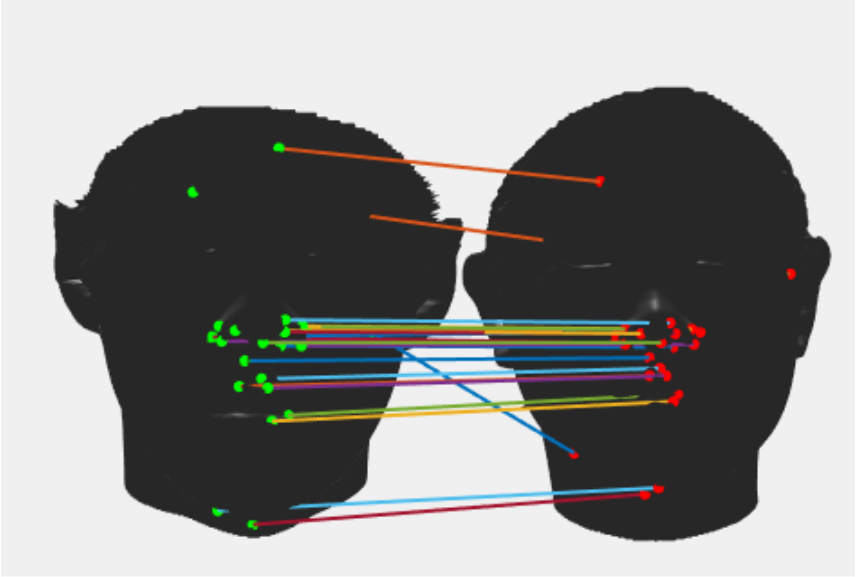


Figure 4.7C: Points matched by MeshSIFT

4.2.5 TPS

The accuracy of results generated from TPS depend on the point matchings, good results are obtained from TPS only if the point matchings are absolutely correct. In the current system implementations, filtering out the point matchings from meshSIFT and the points obtained from the tip of the nose, the top of the forehead and the left and right ear ensure that the resultant point matchings are accurate, hence, TPS results are accurate too. Figure 4.8 represents the TPS result generated from a single source face in order to deform to different target faces.

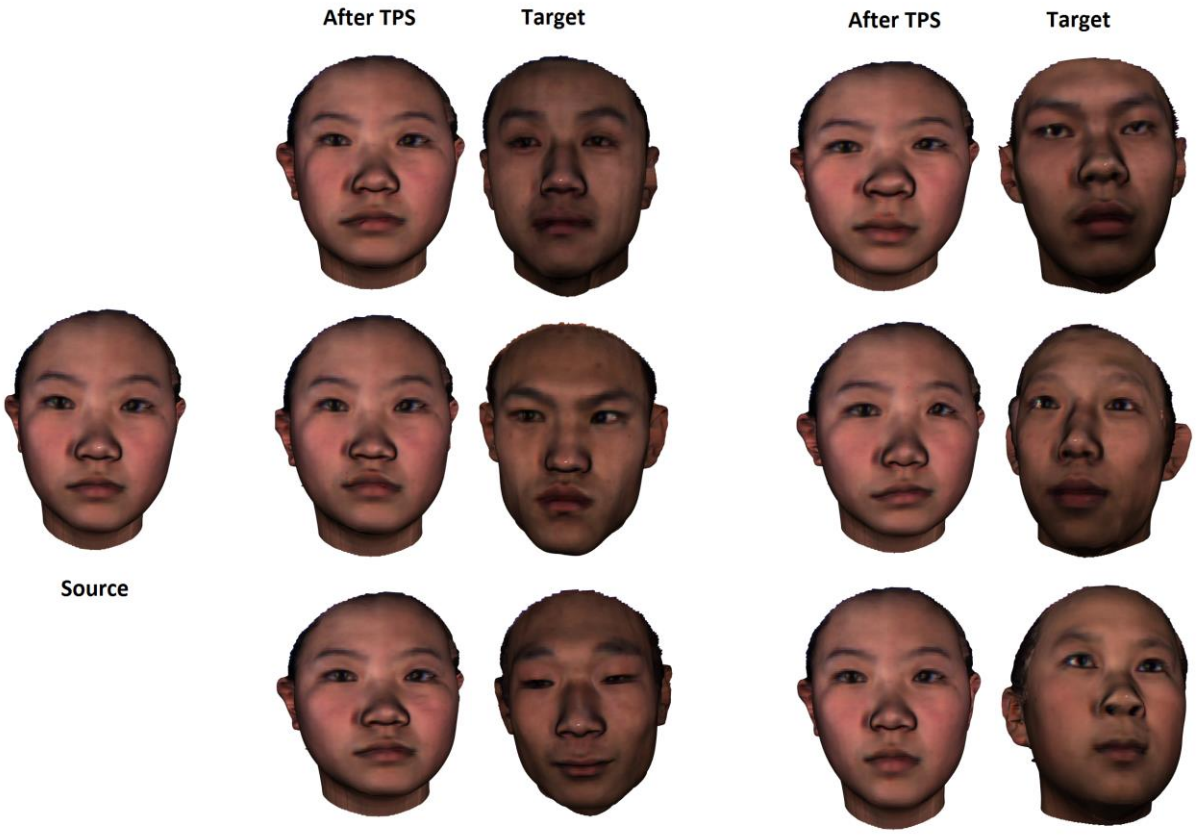


Figure 4.8: TPS Results

4.2.6 ICP

ICP results are based on the accuracy of point matchings and TPS results. Since the system pipeline is designed in a way such that it ensures that the point matchings are very accurate, TPS results are good, and hence, ICP also gives good results. However, if the point matchings are spread over the entire face evenly, the results would be much better. For example, in Figure 4.9B, ICP Iteration 3, the left eye of the mesh is distorted and deformed, this is because the number of point matching in that area are either not sufficient or not there at all. Hence, evenly spread point matching on the face mesh will yield better results. Figure 4.9A to 4.9E represent the results of 3 Iterations of ICP and the final results for different faces.

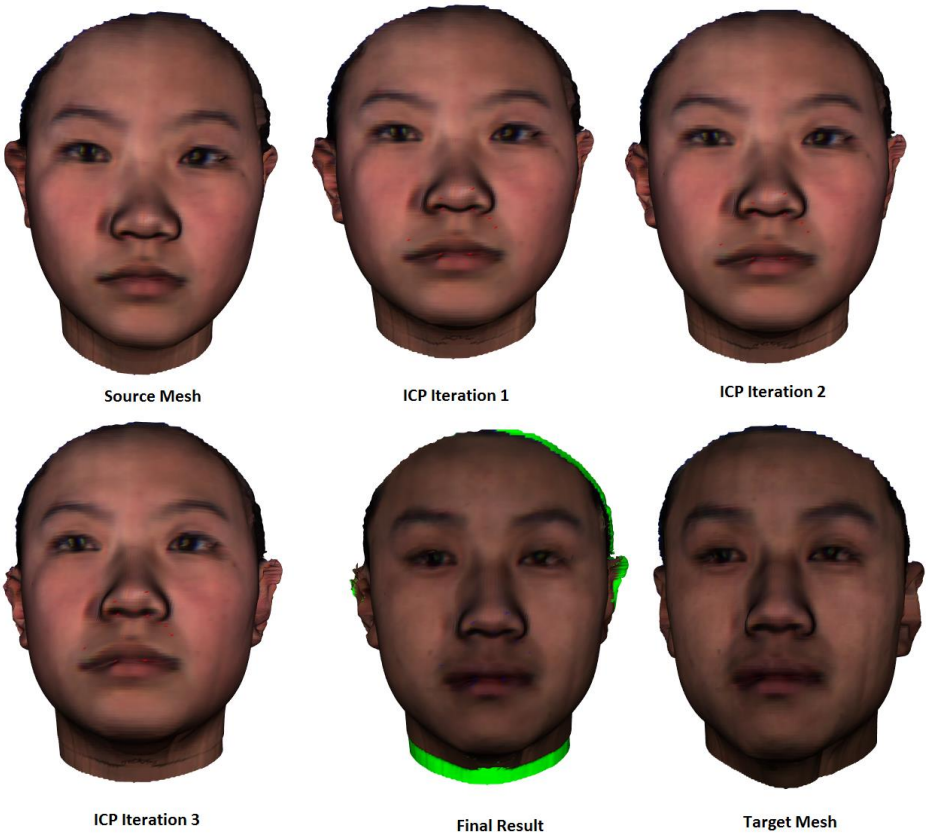


Figure 4.9A ICP Iterations with final result

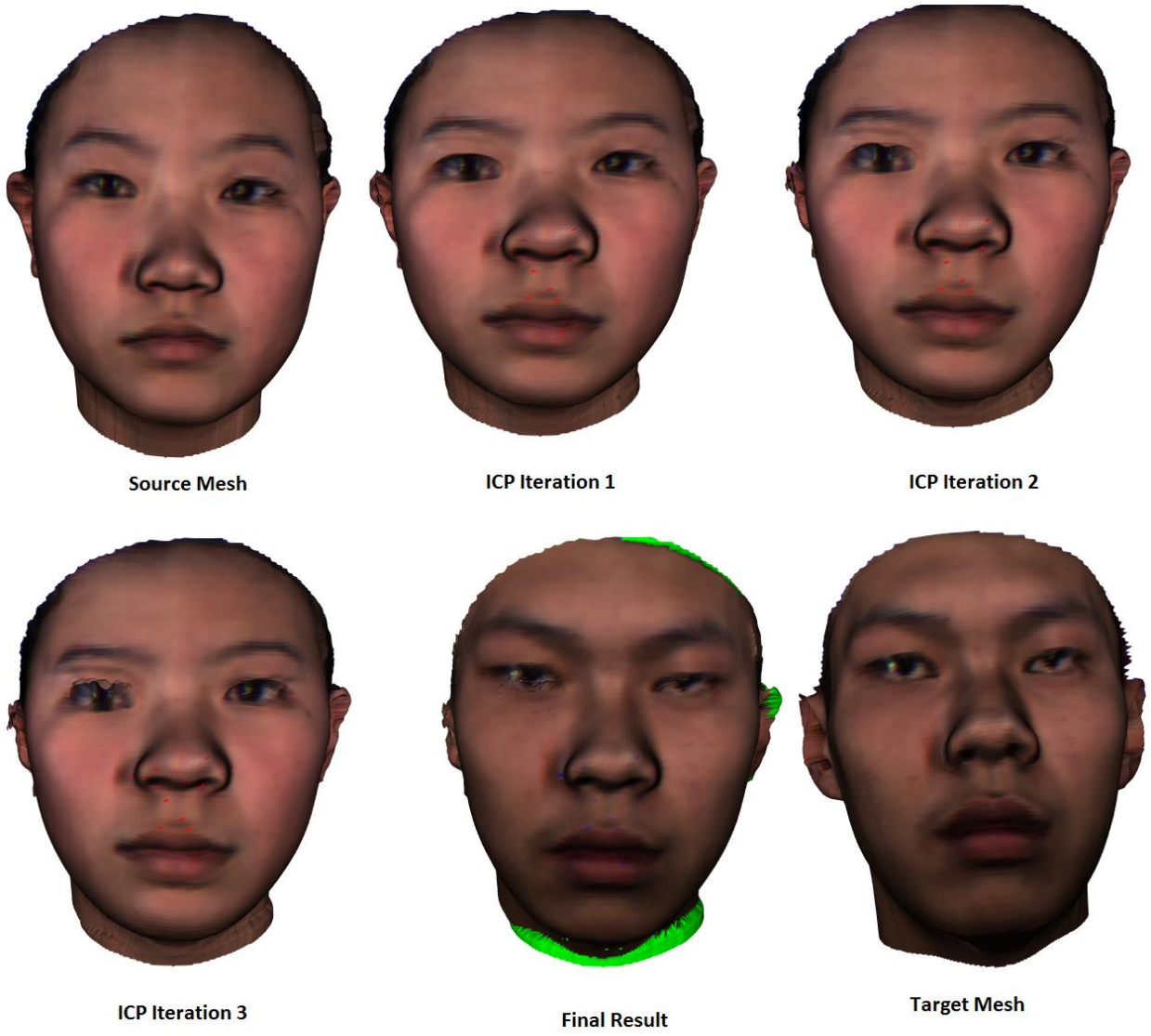


Figure 4.9B ICP Iterations with final result

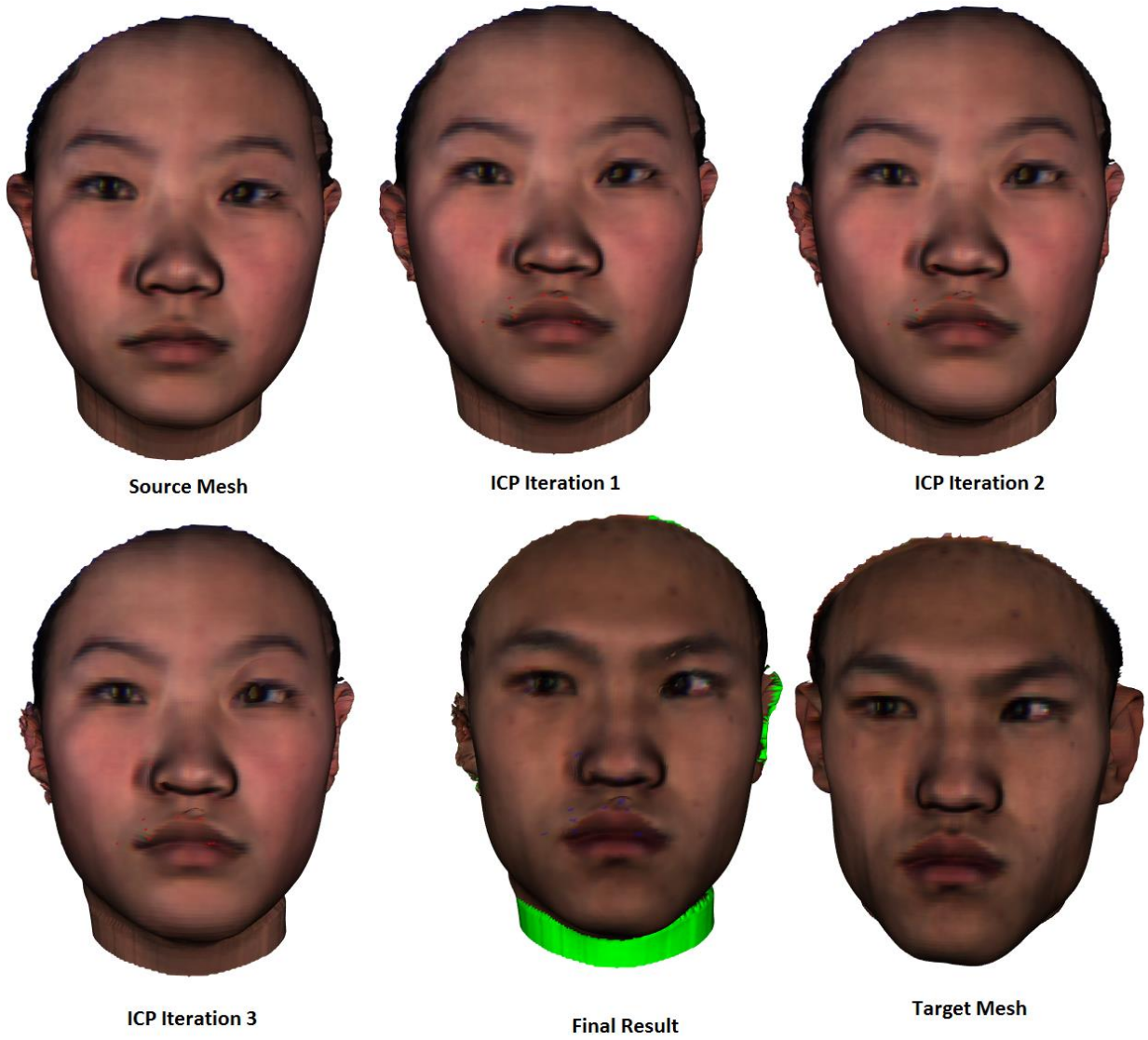


Figure 4.9C ICP Iterations with final result

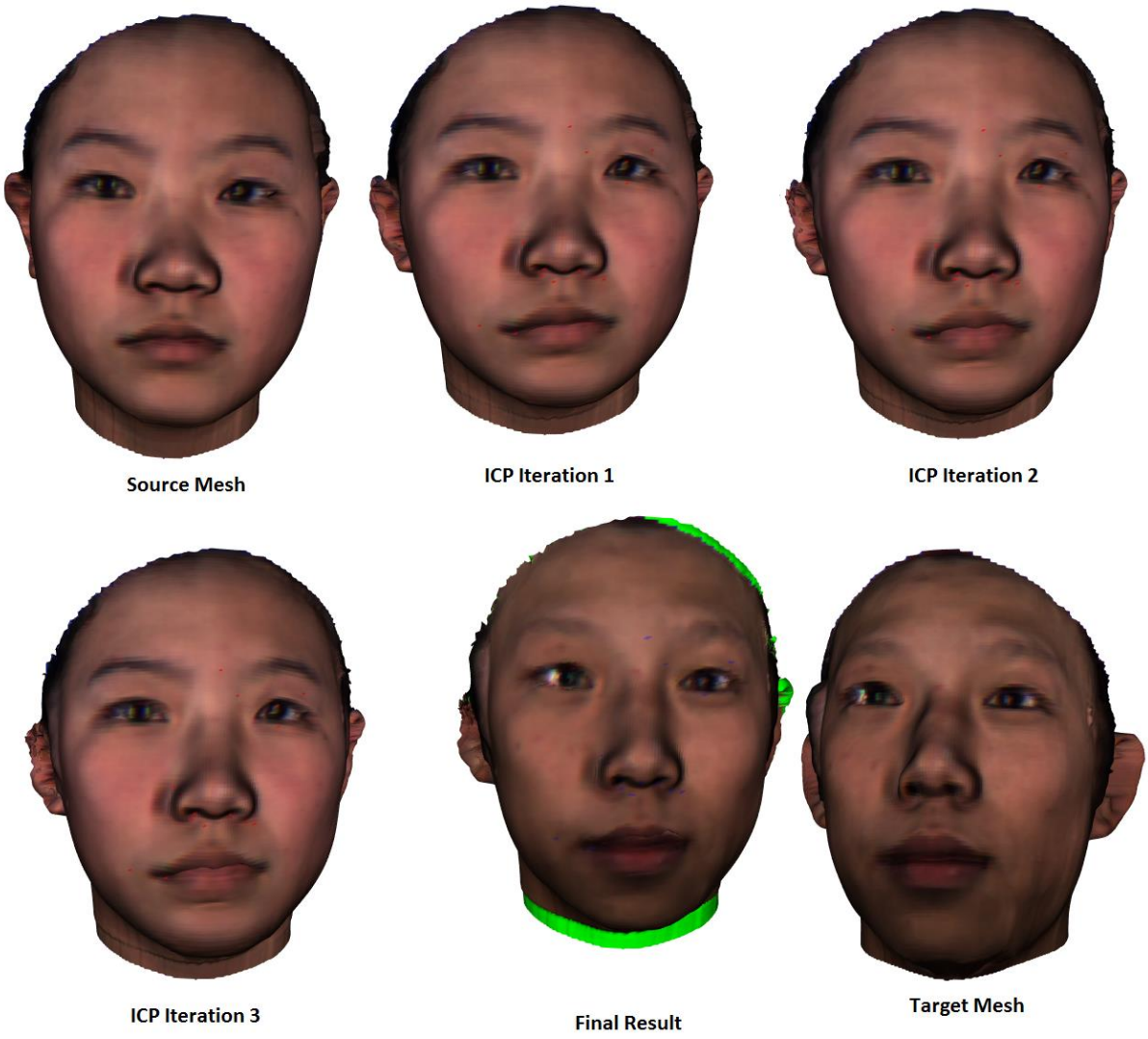


Figure 4.9D ICP Iterations with final result

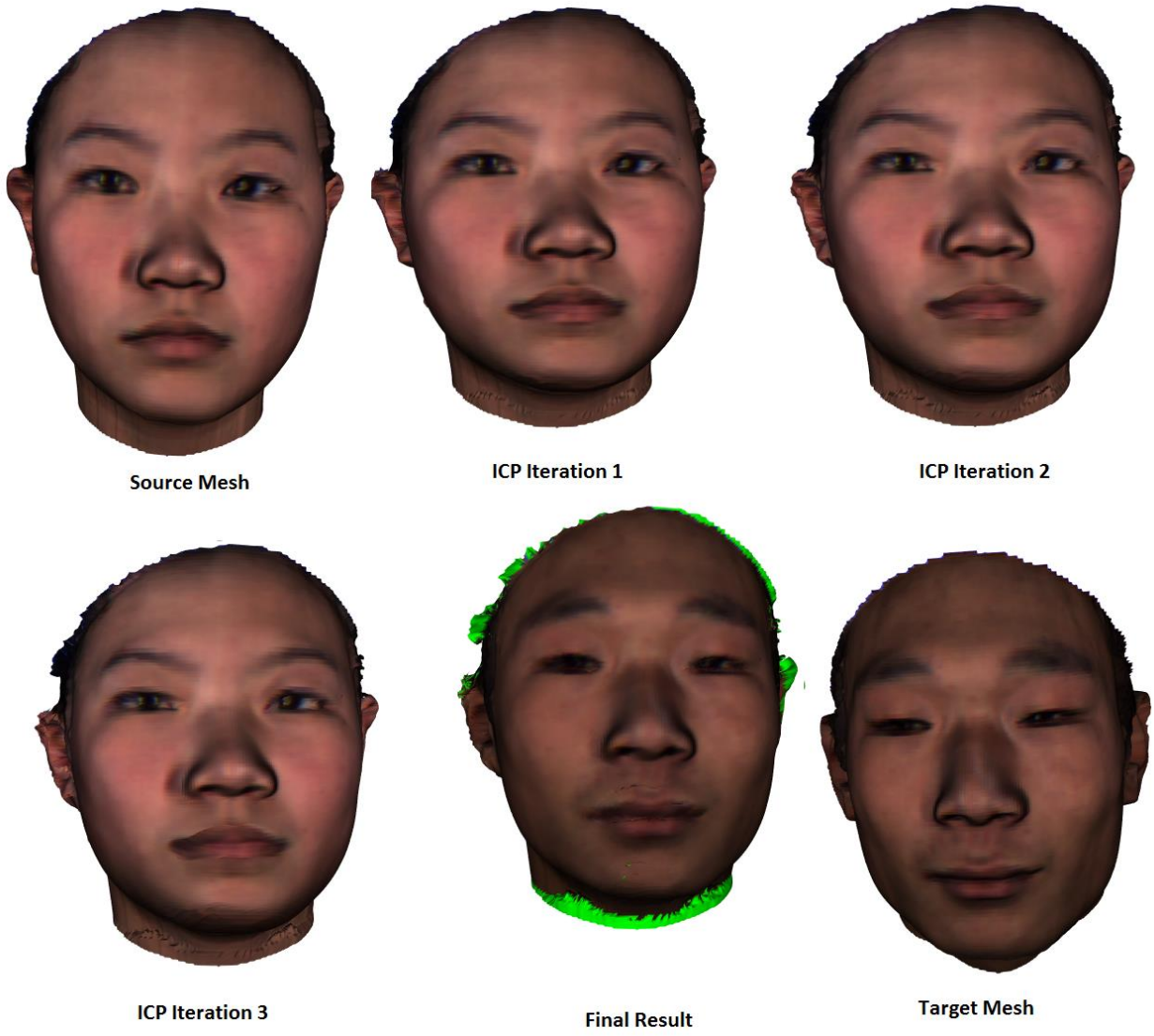


Figure 4.9E ICP Iterations with final result

4.3 Error Analysis

This section, error analysis is done to validate and quantify the results obtained. I order to quantify the results, distance between source and target mesh was recorded between 5 iterations of ICP. The result obtained indicated that the error reduces after each ICP iteration,

which is ideal. Table 4.1 represents the distance between source mesh and target mesh for five faces after each iteration of ICP.

Face	Iteration 1	Iteration 2	Iteration 3	Iteration 4	Iteration 5
Face 1	2.484297771	1.951019575	1.554841732	1.255949831	1.020755918
Face 2	3.470889072	2.677743442	2.130306815	1.723881221	1.400963014
Face 3	3.097707207	2.440079687	1.956640203	1.580732645	1.284457881
Face 4	2.84853309	2.208021829	1.779142449	1.448630599	1.177881752
Face 5	4.319038311	3.361849877	2.676850687	2.172027515	1.775930094

Table 4.1: Distance (in mm) between source mesh and target mesh after each iteration of ICP

Figure 4.10 is the graphical representation of the data in table 4.1, it shows how the distance between source mesh and target mesh is reducing after each iteration of ICP. The distance between source and target mesh greatly depends on the number and location of the landmarks. The more landmarks which are positioned strategically throughout the face, the better the results will be.

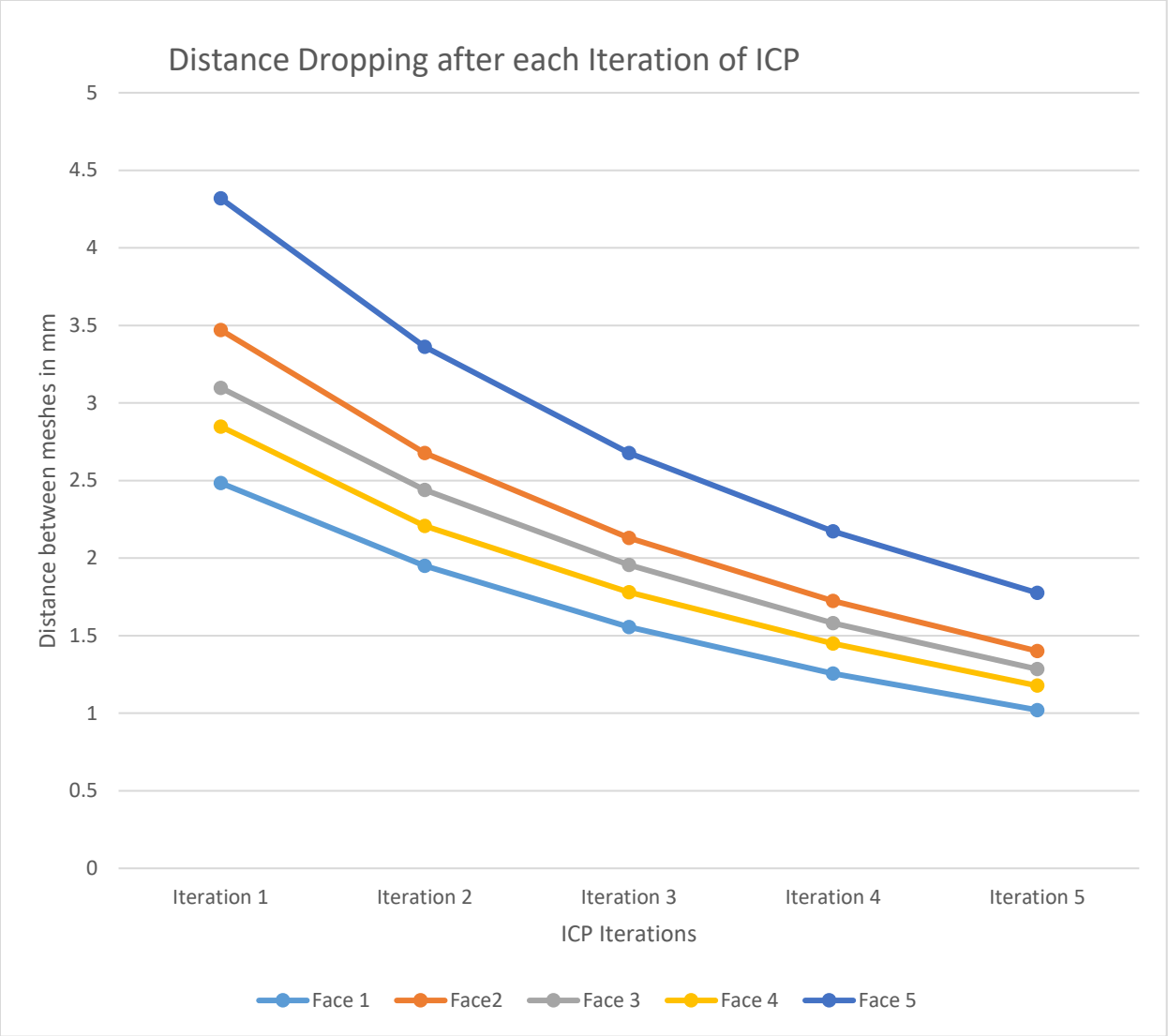


Figure 4.10: Error reduction after each iteration of ICP

4.4 Time Analysis

In this system implementation, MeshSIFT has been used to obtain point matchings between two meshes. The meshes obtained from the database are complex due to which the point matchings from MeshSIFT are very time intensive. To avoid that, an experiment was done by simplifying the meshes using MeshLab which is an open source software for editing 3D

triangular meshes. For this experiment, meshes were simplified to one eighth of their actual sizes, i.e, the number of vertices and faces in the mesh were reduced to one eighth of their original size. Table 4.2A represents the difference between the number of vertices and faces before and after mesh simplification, Figure 4.11A and Figure4.11B represent the change in the number of vertices and faces respectively in the mesh before and after simplification, Table 4.2B depicts the difference between the processing times and the number of point matchings obtained with the source mesh by MeshSIFT before and after the meshes were simplified and Figure 4.11C and Figure 4.11D represent the change in the processing times and number of point matchings respectively obtained from MeshSIFT before and after mesh simplification.

Mesh	Original No. of vertices	No. of vertices after mesh simplification	Original No. of faces	No. of faces after mesh simplification
Source	67891	8597	134694	16836
Face 1	70890	8976	140604	17575
Face 2	69384	8786	137638	17204
Face 3	63131	7992	125204	15650
Face 4	65931	8332	130778	16347
Face 5	65852	8325	130575	16321

Table 4.2A Number of Mesh Faces and Vertices before and after Mesh Simplification

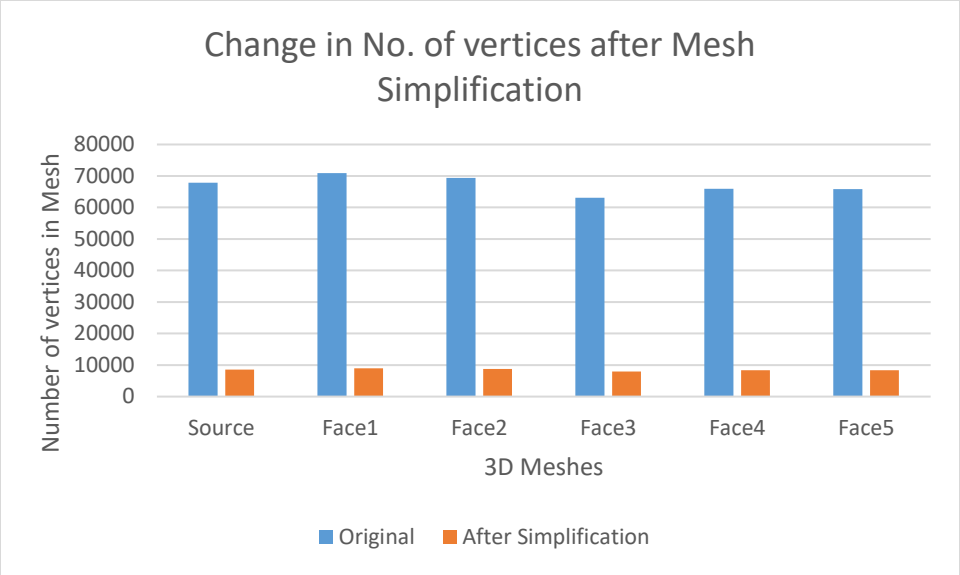


Figure 4.11A Number of vertices in mesh before and after mesh simplification

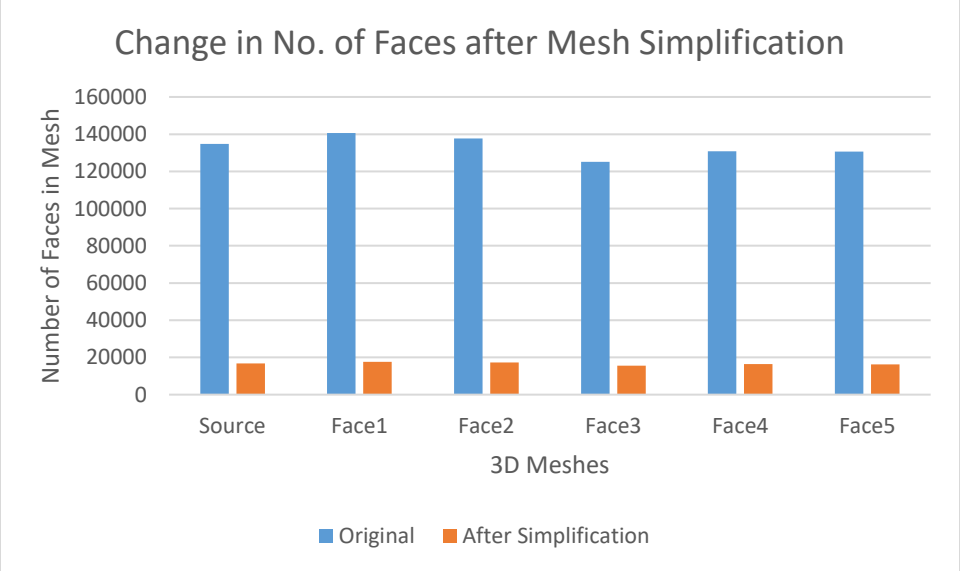


Figure 4.11B Number of faces in mesh before and after mesh simplification

Mesh	MeshSIFT processing time with Original Mesh (in hours)	MeshSIFT processing time with Simplified Mesh (in hours)	MeshSIFT point matchings with Original Mesh	MeshSIFT point matchings with simplified Mesh
Face 1	6	0.333333333	31	7
Face 2	5.5	0.4	23	14
Face 3	5	0.333333333	26	4
Face 4	6.5	0.416666667	30	6
Face 5	5.1	0.383333333	23	12

Table 4.2B Processing times and point matchings obtained from MeshSIFT before and after mesh simplification

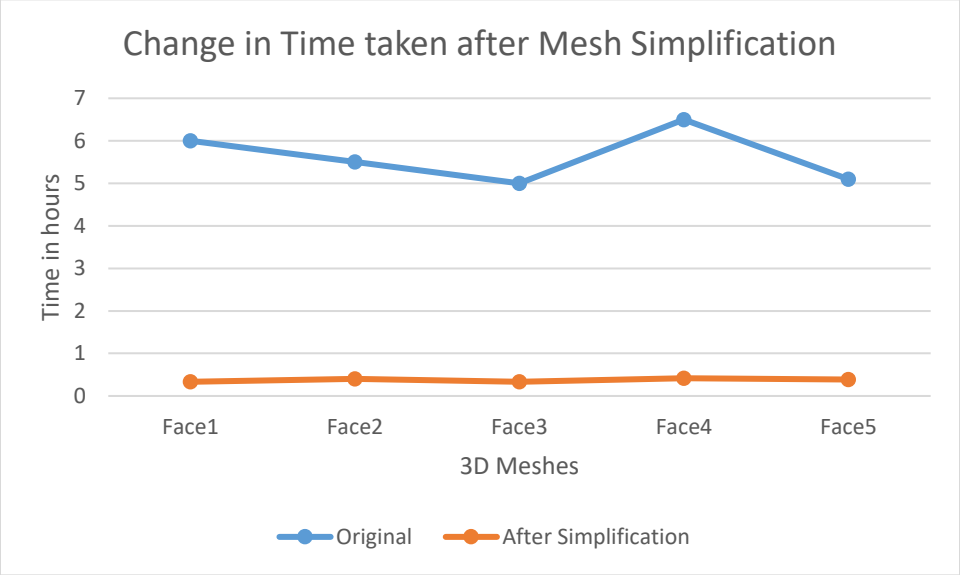


Figure 4.11C MeshSIFT processing times before and after mesh simplification

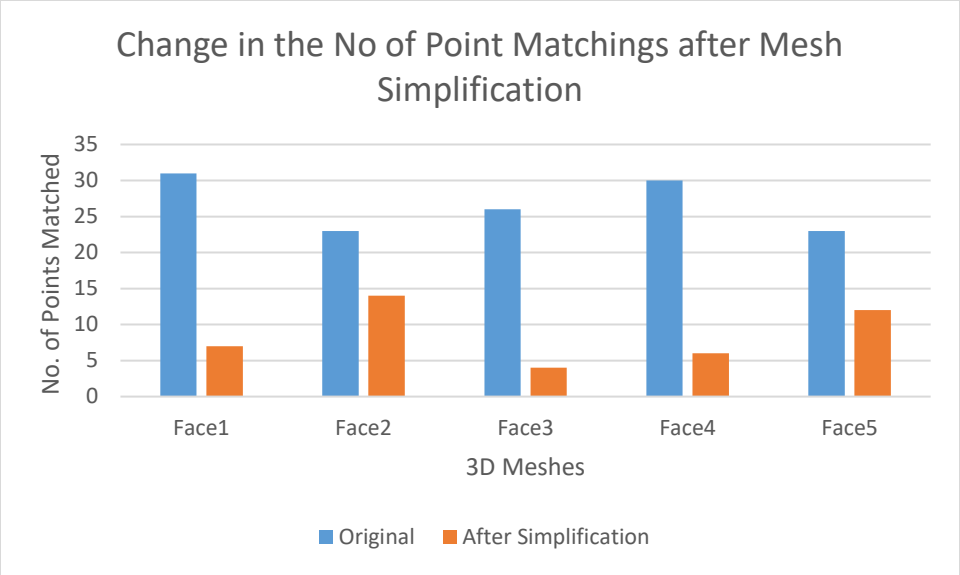


Figure 4.11D Point Matchings obtained from MeshSIFT before and after mesh simplification

From the results of this experiment, it is clear that mesh simplification drastically reduces the processing time of MeshSIFT, which is desirable, but, at the same time, the number of point matchings obtained are also reduced. For the current system, the more point matchings obtained, the better it is, hence, mesh simplification is a tradeoff between speed and the number of point matchings obtained.

Chapter 5 - Conclusion

5.1 Conclusion

The current system accomplishes the automation of 3D point matching for human faces by first reading the meshes from the database and smoothing them using Taubin smoothing as preprocessing to prepare the mesh for further operations. After the preprocessing is done, we proceed with feature detection and detect the tip of the nose. Detecting the tip of the nose is critical as the detection of the top of the forehead and left and right ears are based on it. While these points are being detected, the vertices and faces of both the source and target meshes are loaded in Matlab for MeshSIFT processing. MeshSIFT returns the point matching calculated. The point matching obtained from meshSIFT are not very accurate, some points are mirror images and some are inaccurate. Due to this reason, the point matchings are then filtered using a certain threshold which varies from mesh to mesh. After the point matchings have been filtered, they are combined with the point matchings obtained from the features detected previously. This list of point matching are then input for TPS, which deforms the source mesh based on the point matchings provided to make the source look like target. TPS changes the geometry of the source mesh while the topology is still the same. After the mesh as been transformed using TPS, we use Iterative closest point matching to bring the source and target mesh closer. The closer the meshes are the better the result is. We perform five iterations of ICP. After the fifth iteration, the texture of the target face is copied on the source and the final result is obtained.

In the error analysis, we demonstrate that the results obtained by the pipeline of the current system are reliable and can be used to automate the 3D point matching system for human faces. In time analysis, we observed that mesh simplification drastically reduces the amount of processing time taken by MeshSIFT, but the number of point matchings obtained are also reduced substantially which is undesirable.

5.2 Future Work

This study has a lot of future scope as there are many potential improvements that can be made. Work can be done to integrate the entire system and make the process more optimized and faster. MeshSIFT results are not the best, we get a lot of inaccurate and mirror points, better approaches can be researched in order to find an algorithm which result in better point matchings that are spread throughout the facial mesh and are more accurate with better feature detection capabilities. MeshSIFT is also very time intensive, either an ideal ratio to down sample the mesh can be found which improves efficiency while maintaining the number of point matchings obtained or some other algorithm can be researched and worked on to overcome this issue.

References

[1] R. Rostami, Z. Yu, "3D Point Matching in Face Modeling", Poster Competition 2015, University of Wisconsin - Milwaukee, USA (2015) [On-line access http:// slideplayer.com/slide/7853997/]

[2] Yongli Hu, Mingquan Zhou, Zhongke Wu. An Automatic Dense Point Registration Method for 3D Face Animation. 2nd International Congress on Image and Signal Processing CISP (2009)

[3] Sugato Basu, Arindam Banerjee, Raymond Mooney. Semi-supervised Clustering by Seeding. In the proceedings of the 19th International Conference on Machine Learning, 19-26, Sydney, Australia (2002)

[4] J. Sun, M. Ovsjanikov , L. J. Guibas. A concise and provably informative multi-scale signature based on heat diusion. Proceedings of the Symposium on Geometry Processing (2009)

[5] Multimedia and Intelligent Software Technology Beijing Municipal Key Laboratory, Beijing University of Technology, Beijing, China- The BJUT-3D Large-Scale Chinese Face Database (2005)

[6] Dirk Smeets, Johannes Keustermans, Dirk Vandermeulen, Paul Suetens. “meshSIFT: Local Surface Features for 3D Face Recognition under Expression Variations and Partial Data,” Computer Vision and Image Understanding (2012)

[7] Jarno Elonen. Thin Plate Spline editor- an example program in C++ (2003, 2004, 2005). [Online access https://elonen.iki.fi/code/tpsdemo/]

[8] S. Rusinkiewicz, M. Levoy, "Efficient variants of the ICP algorithm", In Proc. Third International Conference on 3D Digital Imaging and Modeling (3DIM), Quebec City, Canada. IEEE Computer Society, 2001.

[9] S. Allaire, J. Kim, , S. Breen, D. Jaray, V. Pekar, "Full orientation invariance and improved feature selectivity of 3d sift with application to medical image analysis", In IEEE Workshop on Mathematical Methods in Biomedical Image Analysis, 2008.

[10] M. M. Manafzade, A. Harati, "Point Cloud Registration Using MSSIR: Maximally Stable Shape Index Regions", 21st Iranian Conference on Electrical Engineering (ICEE), 2013.

[11] C. Maes, T. Fabry, J. Keustermans, D. Smeets, P. Suetens, and D. Vandermeulen, "Feature detection on 3D face surfaces for pose normalization and recogntion," in BTAS '10, IEEE Int. Conf. on Biometrics: Theory, Applications and Systems, pp. 1-6, September 2010, Washington D.C., USA.

[12] D. Smeets, J. Keustermans, J. Hermans, P. Claes, D. Vandermeulen, P. Suetens, "Symmetric surface-feature based 3D face recognition for partial data," in IJCB '11: The International Joint Conference on Biometrics, pp. 1-5, October 11-13, 2011, Washington D.C., USA.

[13] H. Wu, H. Zha, "Robust consistent correspondence between 3D non-rigid shapes based on Dual Shape-DNA ", in IEEE International Conference on Computer Vision (ICCV), 2011.

[14] B. Gokberk, A. A. Salah, N. Alyuz, L, Akarun, "3D Face Recognition", pp. 263-295, Springer London ,2009.

[15] W. Zhao, R. Chellappa, P. J. Phillips, and A. Rosenfeld. 2003. Face recognition: A literature survey. ACM Comput. Surv. 35, 4 (December 2003)

[16] B. Jian , B. Vemuri, "Robust Point Set Registration Using Gaussian Mixture Models", IEEE Transaction on Pattern Analysis and Machine Intelligence , 2011.

[17] S. Allaire, J. Kim, S. Breen, D. Jaffray, V. Pekar, "Full orientation invariance and improved feature selectivity of 3d sift with application to medical image analysis", In IEEE Workshop on Mathematical Methods in Biomedical Image Analysis, 2008.

[18] Xiaoguang Lu, Anil K jaun, “Deformation modelling of Robust 3D Face Matching”, Dept. of Computer Science & Engineering, Michigan State University.

[19] Y. Adini, Y. Moses and S. Ullman, “Face Recognition: The problem of Compensating for Changes in Illumination Direction,” IEEE Trans on PAMI, Vol.19, pp. 721-732,1997.

[20] Fred L. Bookstein, “Principal Warps: Thin-Plate Splines and the Decomposition of Deformations,”In IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 11, No. 6. June 1989.

[21] Lena Maier-Hein, Alfred M. Franz, Thiago R. dos Santos, Mirko Schmidt, Markus Fangerau, Hans-Peter Meinzer, and J. Michael Fitzpatrick, “Convergent Iterative Closest-PointAlgorithm to Accomodate Anisotropicand Inhomogenous Localization Error,” In IEEE Transactions on Pattern Analysis and machine Intelligence, Vol. 34, No. 8, August 2012.

[22] P. Cignoni, M. Callieri, M. Corsini, M. Dellepiane, F. Ganovelli, G. Ranzuglia, MeshLab: an Open-Source Mesh Processing Tool Sixth Eurographics Italian Chapter Conference, page 129-136, 2008. [Online access: http://www.meshlab.net/]