Masarykova univerzita
Fakulta informatiky

Comparison and Analysis of Multiple 3D Shapes

Master's thesis

Bc. Zuzana Ferková

Brno, Autumn 2015

Declaration

Hereby I declare, that this paper is my original authorial work, which I have worked out by my own. All sources, references and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Bc. Zuzana Ferková

Advisor: RNDr. Igor Chalás

Acknowledgement

I would like to show my gratitude to my supervisor RNDr. Igor Chalás for his support and continual help resolving issues in my thesis. Many thanks also go to AM, KF and AW whose help has been invaluable in finalizing this thesis.

Abstract

Shape comparison and analysis is a quickly developing field, useable in many areas. One of the most common usages nowadays is to compare two meshes, to determine how much they differ, or to compare a mesh with a database, to find the closest match. However, cross-examination of the set of meshes is also an interesting problem, which hasn't got as much attention. It can be used to detail prominent parts of the models in the group and define how much each of the meshes deviates from it. The aim of this thesis was to create an algorithm, which would allow to effectively cross-examine large sets of data and implement it into the *FIDENTIS Analyst* — application for analysis of 3D facial data. This thesis presents the description of the algorithm and compares it to two different approaches which are derived from more common identification and verification approaches.

Keywords

shape comparison, shape analysis, data cross-examination, FIDENTIS, Scaling Iterative Closest Point, Hausdorff distance, mean shape

Contents

1 **Introduction** . . . 1
2 **Literature Overview** . . . 3
2.1 *2D Facial Recognition* . . . 3
2.2 *2.5D Facial Recognition* . . . 4
2.3 *3D facial recognition* . . . 5
2.3.1 3D Facial Recognition Categories . . . 5
2.3.1.1 3D Facial Recognition Based on Facial Curves 5
2.3.1.2 3D Facial Recognition Based on Facial Surface Features . . . 6
2.4 *Registration of Point Sets in 3D* . . . 7
2.4.1 Pair-wise Registration of Point Sets in 3D . . . 8
2.4.2 Registration of Multiple Point Sets in 3D . . . 9
2.5 *Creating the Average Face* . . . 10
3 **Theory** . . . 12
3.1 *Mesh representation* . . . 12
3.2 *Scaling Iterative Closest Point* . . . 13
3.3 *Mesh Analysis* . . . 18
3.3.1 Hausdorff distance . . . 18
3.3.2 Metrics for Analysis . . . 19
3.4 *Mean Shape* . . . 22
3.5 *Compared Algorithms* . . . 23
4 **Implementation** . . . 25
4.1 *FIDENTIS Analyst* . . . 25
4.2 *Implementation of Batch mode* . . . 26
4.3 *Memory Issues* . . . 27
4.4 *Speed Issues* . . . 29
5 **Evaluation** . . . 32
5.1 *Input data* . . . 32
5.2 *Evaluation parameters* . . . 33
5.3 *Speed Evaluation* . . . 35
5.4 *Validation of the algorithms* . . . 39
5.5 *Evaluation of quality of the alignment* . . . 40
5.5.1 Comparing three algorithms . . . 41
5.6 *Comparison of Average Shapes* . . . 45
5.7 *Evaluation Results* . . . 49
6 **Conclusion** . . . 51
References . . . 54

Appendix A 60
Appendix B 66

1 Introduction

Shape analysis of 3D surfaces is a classical problem in computer graphics. The main goals of shape analysis are to compute geometric properties of surfaces and to produce new representations from which important features can be inferred [28]. It is a technique that can be used in many fields. For example, botanists can use mesh representation of the plants to observe their growth, anthropologists can determine differences between human faces, engineers can easily determine which of their mechanical parts are faulty. In addition, 3D meshes can now be used as biometric data as well.

Especially with increased accessibility of scanning devices, many fields can use 3D meshes and all information they provide instead of describing the surfaces by a set of parameters.

Comparison of meshes is primarily performed in three different ways. First, the pair-wise comparison, is used for pairs of meshes to determine how much they differ. This approach is employed in facial verification in biometry, but it can also be used with plastic surgery, when comparing parts of the body before and after an operation.

Another widely used technique is comparison of a single mesh against a model database. Unlike the pair-wise comparison, its primary use is to analyze more than two meshes. In facial recognition, this approach is called identification and it attempts to find the most similar individual in the database. It can be used to find a suspect in the criminal database, or we want to match images of same individuals in various stages of their lives.

Third approach is the batch comparison, and it is focus of this thesis. Similar to comparison with a database, it is mainly used to analyze more than two meshes. However, the previous approach focused on analyzing a single model against the database, whereas our aim is to cross-analyze the set of data, to determine specific attributes of the meshes. This way we can get a comparison between each pair of the set. We can use this to, for example quantify intra- or inter-population variability. As such this approach is not as common as the previous two. It is also more computationally difficult, posing challenges such as memory limitations and speed of computation.

While we describe the algorithm, which can be used and was tested, on different types of meshes, varying from human bones to 3D scans of mice, our main focus is to create an algorithm which would work within the *FIDENTIS Analyst*.

In this thesis we looked at a general problem of analysis and comparison of multiple 3D shapes and focused on a subset of the problem — comparison and analysis of sets of human faces. This decision was made due to a close cooperation with anthropologists, who use the *FIDENTIS Analyst* in their work and required this functionality. It provided us with suitable amount of data to test and evaluate our algorithm.

In this thesis we introduce the algorithm to deal with the batch comparison problem, by aligning all the meshes in the set to the mean shape of the set. We use customized pair-wise comparison and adapted comparison to model database algorithms to compare our approach to. It is our goal to show that for the given problem, our algorithm is faster than pair-wise comparison and more precise than the comparison with database algorithm.

This thesis is structured as follows. Chapter One provides overview of literature dealing with similar problems. In this chapter we show solutions to facial recognition in 2D and 3D. We also provide references to papers which deal with mesh alignment and creation of mean shape.

Chapter Two introduces necessary theory, describing used mesh representation, mathematical background behind alignment and comparison algorithms, as well as metrics used for analysis of meshes. It also describes all three algorithms evaluated in this thesis.

Chapter Three deals with implementation of the batch algorithm, the problems which arose with it, and their solutions. It also describes ways to improve on the speed of the computation.

Chapter Four evaluates our algorithm with two more approaches for dealing with the same problem. It provides speed and quality evaluation and shows how different values of parameters and quality of mesh effect the results of comparison.

2 Literature Overview

The first facial recognition systems developed in 1960s were semi-automated and required the administrator to locate features, such as eyes, ears, nose and mouth, on the photography before it could proceed with facial recognition [1]. With increasing demand for fast and accurate facial recognition techniques it quickly became apparent, that automatic approaches would be required. Currently, facial recognition is a dynamically developing field, which can be divided into several categories based on the data it is working with, as well as the purpose of its use.

This section describes some of the most used approaches to compare and analyze human faces, based on categorization in [22]. It also shows the solutions to automatic superimposition of models, both for a pair of meshes and multiple models. The last section of the chapter details approaches for creating an average shape of the set of meshes.

2.1 2D Facial Recognition

Facial recognition using 2D images is older and hence more developed than its counterpart, 3D facial recognition. Describing face using significant points, called feature points or landmarks, such as corners of the eyes, or tip of the nose is a technique often utilize in anthropology. This approach however, requires either manual placement of the feature points, or using an algorithm to find feature points automatically, which can be problematic because of the image noise.

Due to problems with determining feature points, several algorithms were proposed to avoid their use, such as ones based on eigenface and fisherface approaches, for example [9][48]. Other solutions use Local Binary Patterns, such as [8] and [50], or neural networks, such as [42] and [32]. However, 2D facial recognition still suffers when data is acquired under different conditions, such as changes in lighting, captured pose or noise introduced during capturing process.

The spiritual predecessor of *FIDENTIS Analyst*, *FIDO*, which was created for forensic identification of the faces of children, uses 2D data for facial recognition. Here, the algorithm for determining similarities between faces is derived from techniques used by anthropologists, who use feature points to describe human faces. A set of these points describing a single face is called

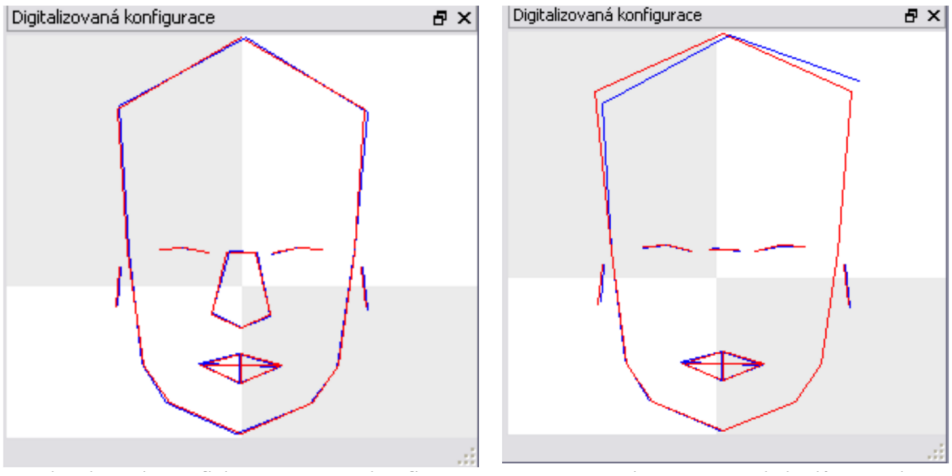

Figure 2.1: 2D Facial comparison in FIDO, based on feature points.

a configuration and differences between faces can then be determined by superimposing configurations one to another and using Procrustes Analysis [29]. This approach is shown in Figure 2.1.

2.2 2.5D Facial Recognition

With increasing computational power and availability of 3D capturing systems, it was only natural for 3D data to be used in similar manner. Lack of depth in 2D images was one of the limitations, which was overcame in part by using intensity of the picture, but it was often difficult to get correct depth information due to the self-occluding nature of human faces. On the contrary, 3D scans contain a third dimension, which could then be used to resolve this issue. Algorithms using information acquired from 3D scans, but using 2D approaches, are called 2.5D algorithms.

These algorithms usually use intensity or range images, which contain depth information. Figure 2.2 shows range image representation of human face. Since this information is acquired from 3D data, depth is computed more accurately than from 2D image. Some of the works dealing with this category of data are [21] and [44].

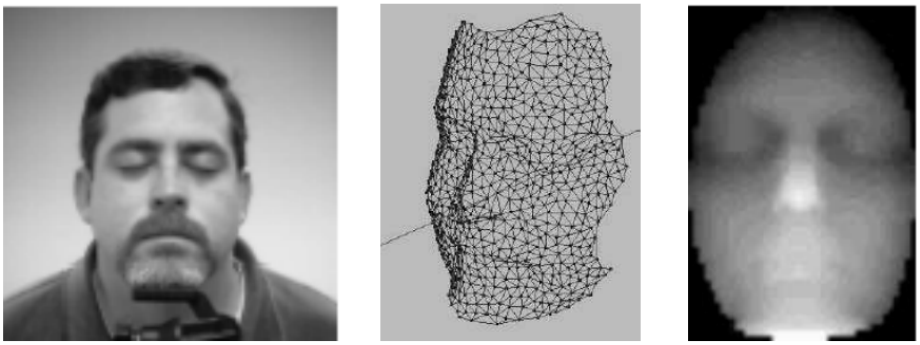

Figure 2.2: Polygonal mesh and its representation through a range image [37].

2.3 3D facial recognition

Using acquired 3D meshes, we can overcome some of the difficulties of 2D facial registration, especially uneven illumination. Using 3D meshes is an approach used even in 2D approaches to render new images with different lighting, as proposed in [15]. This data also offers a more natural way of looking at the human faces and allows to use approaches similar to those people use in everyday interaction with each other.

2.3.1 3D Facial Recognition Categories

In [22] author defines two categories of 3D facial recognition. The first one, based on facial surface features, involves using feature points, distances between surfaces, or curvature of models. The second one, based on facial curves, uses algorithms that define curve between two points on the faces and compare the created curves.

2.3.1.1 3D Facial Recognition Based on Facial Curves

The first category of 3D facial recognition is based on the *Facial Curve* representation of the surface. The advantage of this approach is that it doesn't require superimposition of the data, but instead we can use tools to analyze curves. According to [22] these methods are rather suitable for measuring changes of the face, for example, during aging or due to a surgery. The idea of 3D face recognition using curves is to detect starting and ending points of

the curve on the face and measure the distance between them, either along the line, or along the surface, as demonstrated in [17]. One of the major challenges of the surface analysis using this approach is that there is no canonical coordinate system to represent and study the geometry of a non-linear surface such as a face. Thus the parametrization of the curve may not be unique, even though analysis of shapes of curves modulo all parametrizations are still tractable. However, this task leads to enormous computational challenges [22].

2.3.1.2 3D Facial Recognition Based on Facial Surface Features

An extension of 2D approach is analysis through feature points. Here, manual or automatic detection of landmarks is required. A way of automatic detection was proposed, for example by Segundo et al. [45] and Galvánek et al. in [27]. For this technique of facial recognition it is imperative for feature points to be detected as accurately as possible, which can prove difficult due to noise in the mesh. Figure 2.3 shows configuration of Feature Points found on 3D model. Procrustes Analysis can then be used to determine similarities between two configurations, or a set of faces. In [30] authors describe algorithm for facial recognition using feature points for facial recognition.

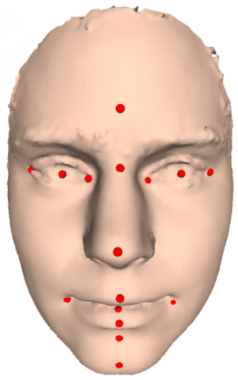

Figure 2.3: Feature Points on the 3D face [27].

Surface based algorithms compute differences between meshes using all the information 3D scans provide. These approaches are divided on those using distances between surfaces, such as [34], which was developed for 2D images, but can easily be extended to third dimension. In [41] authors use

Hausdorff distance, same as the previous article, but utilize it in the 3D facial recognition.

Curvature based algorithms compare and analyze meshes based on the curvature information, that is computed using principal curvature. Some of the approaches using this technique were introduced in [38] and [47].

2.4 Registration of Point Sets in 3D

Many of the algorithms computing facial recognition based on facial surface features require superimposition of meshes prior to their comparison. This is especially true to the algorithms that are measuring distances between surfaces. Figure 2.4 and Figure 2.5 show difference between the comparison of unaligned and aligned models.

Registration of point sets in 3D is relatively well resolved problem. We can divide it into two groups. The first being the pair-wise registration, which can be used in facial verification and other situations where two meshes are compared. The second, registration of multiple point sets, is often used when reconstructing 3D meshes from different still images.

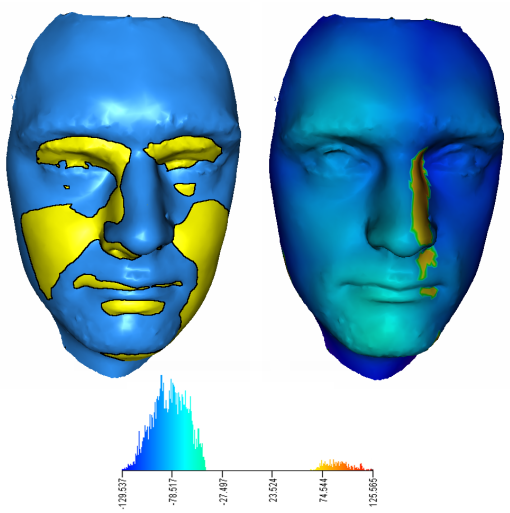

Figure 2.4: Overlaid unaligned faces and their comparison through color map. Deep blue color shows the smallest differences, while red color shows the largest differences in comparison.

2.4.1 Pair-wise Registration of Point Sets in 3D

In order to be able to compare 3D models, many approaches superimpose meshes. One of the most common way to align 3D data onto one another is the *Iterative Closest Point (ICP)* algorithm, presented by Besl [12]. At same time, similar approach was proposed by Zhang in [51]. This method minimizes the difference between two point clouds by decreasing distances between them in every iteration. Rather than using rotation matrices, the method in [11] can be used to compute quaternions to use for rotation. *ICP* algorithm is a widely popular solution and as such was improved on several times.

A review of some of the improvements can be found in [43], along with a proposal of new, efficient method of *ICP*. Another review, showing development of the *ICP* algorithm between years 2002 and 2007 is offered by [40]. One of the *ICP* modifications algorithms allows for computation of scaling transformation. This addition was published by several authors, for example, in [25] and [24]. It allows to compute uniform scaling parameter to avoid inaccuracy, which could occur, when 3D scans were taken from different distances.

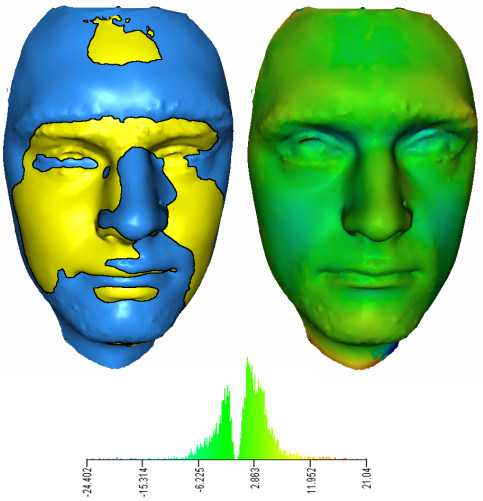

Figure 2.5: Overlaid aligned faces and their comparison through color map. Deep blue color shows the smallest differences, while red color shows the largest differences in comparison.

2.4.2 Registration of Multiple Point Sets in 3D

The problem of aligning two meshes to one another, also called pair-wise registration, was relatively well solved by several authors. However, it often happens, that we need to align more than two meshes. We can use pair-wise registration for this purpose, however this process, especially for facial comparison, can be too lengthy. As pointed out in [10], this approach remains essentially local, and especially with partially overlapping surfaces, it can lead to error propagation. Solutions to resolve this issue were proposed in [10] by using unit quaternions and [46] using a numerical, iterative solution.

Another way to speed up registration of multiple sets can be performed by extending the algorithm of aligning using feature points. While transformations, e.g. translation, rotation and scale, are only computed from the found configuration of feature points, these transformations are applied to each vertex of mesh. This approach, as mentioned before, heavily relies on correct detection of feature points.

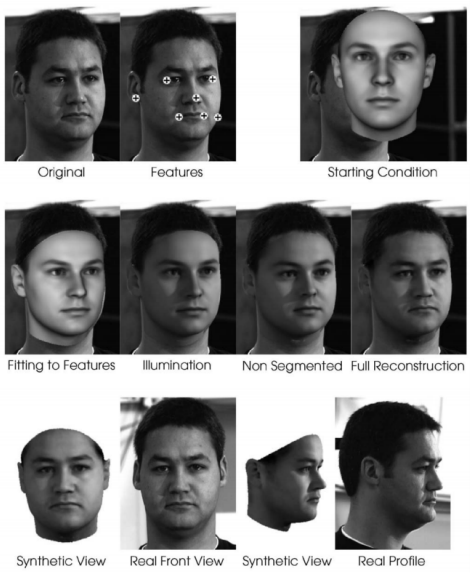

Figure 2.6: Active Model Matching in [15].

Registration of multiple point sets in 3D is a task, which is not only applied to facial recognition, but is also often used when recreating 3D mesh from different still images. This process is called multi-view surface matching

and is common when using 3D scanners. Unlike in 3D facial recognition, multi-view surface matching has only partially overlapping meshes, which are then merged together to create a single model. Solutions to this problem were proposed, for example by [49] and a fully automatic solution can be found in [33].

2.5 Creating the Average Face

Creation of the average face is not a new task and algorithms to resolve this problem have been published for both 2D images and 3D meshes. In 2D we can find two main approaches, first based on optic-flow, such as [13], second using active model matching, such as [26] and [35]. This approach is shown on Figure 2.6. Some of these approaches, such as [15], use both techniques and work for both 2D and 3D data.

The one of the most cited articles on creating morphable faces, which can be used to compute average faces, was published by Blanz et al. [14]. It allows to either recreate 3D model from multiple photographs, or a used set of parameters to create new faces. The paper describes creation of average face and its use for creation of new unique faces. Parameters used are set for different regions of the face and by changing them, the algorithm adds or takes deviations from an average face, creating new meshes. The mesh created by this algorithm and its segmentation is shown in Figure 2.7. This method, however, only uses a single average face computed from the database containing 200 individuals and does not provide computation of a new average every time a new set of meshes are loaded.

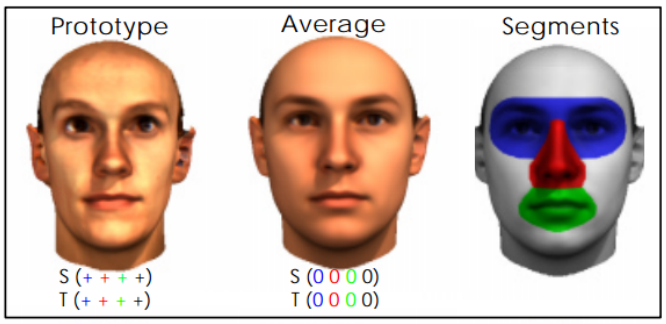

Figure 2.7: Morphable meshes [14].

In [23], another technique was provided to create meshes by using random anthropometric measurements. By averaging these values we can get an average face.

3 Theory

Many ways to approach the problem of determining how one shape differs from another were proposed in the past. In this thesis we are taking this general problem and focusing on showing one of possible solutions on a subset of shapes, specifically human faces. In this chapter, we describe the theory necessary for using our technique for comparing meshes in a set and compare our approach with other possibilities for resolving the same problem.

3.1 Mesh representation

Meshes of the human faces we used for evaluation of the algorithm in this thesis were acquired by scanning volunteers. The scans were taken by the Department of Anthropology of Faculty of Science, Masaryk University in Brno. Reconstruction of 3D models is performed by software provided with scanning tools and as such we are able to acquire polygonal meshes, such as one shown in Figure 3.1, which is a common way of representing surfaces in computer graphics.

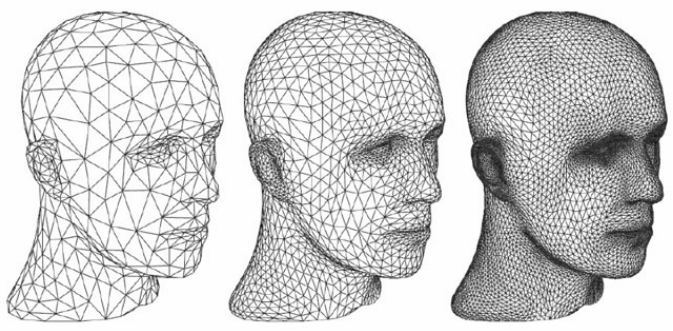

Figure 3.1: Example of Polygonal Mesh [31].

One of the most common ways to represent polygonal meshes is by storing a set of individual faces by their vertex points, with at least three vertices required per face. If each polygon is defined by exactly three vertices, the mesh is referred to as triangulated. An example of a simple polygonal mesh format that uses this representation is the OBJ file, developed by Wavefront Technologies (see [3] for specification). This format stores mesh vertices, nor-

mal directions and texture coordinates as a tuple of three decimal numbers that determine its position in the space. Polygons are then represented by the vertices that define them.

This simplistic representation is sufficient for many tasks we need to perform, although it has its limitations. As stated in [16], without more processing of the information, we can't use this format for operations, such as:

- Oriented traversal of the edges.
- Accessing the faces sharing an edge.
- Accessing the endpoints of a given edge.
- Accessing the faces and the edges containing a given vertex.

And other operations which require traversing the mesh.

The *FIDENTIS Analyst*, in which the algorithm presented in this thesis is implemented, already incorporates data structures that compute additional information countering the limitations of OBJ file format, more specifically Double Connected Edge List. It also supports STL format (see [7]) and PLY format (see [5]).

In general, currently implemented algorithms for analysis of database of entries use vertex-to-vertex metric, see 3.2(a), and as such work even with point cloud representation of mesh. Unlike polygonal mesh, point clouds are only defined by vertices and don't contain information about faces. To get polygonal mesh out of point cloud representation, we can use triangulation algorithms, such as Delaunay triangulation, to create faces for our surface.

3.2 Scaling Iterative Closest Point

To compare how the faces in the set differ, we need to superimpose them. By aligning two meshes as close to one another we can then analyze the differences between corresponding vertices on each mesh. In theory, the smaller the differences, the more closely related the models are.

Iterative Closest Point (ICP), [12] and [51], is one of the most widely used algorithms that minimize the difference between two point sets. Its relatively simple concept allows it to be used in real-time applications. Figure 3.3

shows the minimization of differences between two meshes with an increased number of iterations.

Let the secondary mesh, which we register to primary mesh, through its vertices, be defined as $A = (a_1, a_2, \ldots, a_n)$, and primary mesh as $B = (b_1, b_2, \ldots, b_k)$, without imposing restriction of both meshes needing to have same number of vertices. We then establish correspondence of each vertex in secondary mesh A with primary mesh B, resulting in vector $C = (c_1, c_2, \ldots, c_n)$, where a_i is in relation with c_i for $1 \leq i \leq n$. The

For our algorithm, we establish this correspondence based on the nearest neighbor criteria, using vertex-to-vertex metric. We can find vertex c_i, by determining distance d, as defined in:

$$d(a_i, B) = \min_{b_j \in B} \|b_j - a_i\| \tag{3.1}$$

resulting in minimal distance d computed for vertex b_i we set $c_i = b_i$. In [36], Krajíček describes more criteria, which can be used, such as fuzzy correspondence, or normal intersection correspondence. Visual representation of this criteria is shown in Figure 3.2.

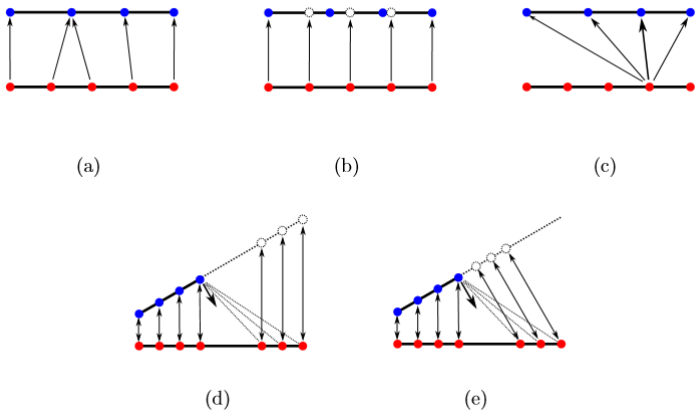

Figure 3.2: Different mesh correspondence approaches: (a) vertex-to-vertex, (b) vertex-to-point, (c) fuzzy correspondence, (d) plane intersection, (e) normal intersection [36]. Blue color depicts vertices of primary mesh, while red color depicts vertices of secondary mesh.

We can then define the minimization function, which we are trying to solve, as described in [51]:

$$F(R,t) = \frac{1}{N}\sum_{i=1}^{N} \|Ra_i + t - c_i\|^2 \tag{3.2}$$

where R represents rotation, t represents translation parameter and N is cardinality of point set of secondary mesh —- vectors A and C have the same size.

To compute translation and rotation parameters, we can use any optimization method, such as steepest descent or conjugate gradient [51]. In our approach, we compute translation vector as:

$$t = \frac{1}{N}\sum_{i=1}^{N} \|c_i - a_i\| \tag{3.3}$$

After computing translation parameter, we compute rotation. Unlike in [12] we use quaternions for rotations, using method in [11]. First we compute matrix M as:

$$M = \sum_{i=1}^{N} {A_i}^T C_i \tag{3.4}$$

where A_i is a matrix representation of vertex $a_i = (a_{ix}, a_{iy}, a_{iz})$ defined as:

$$A_i = \begin{bmatrix} 0 & -a_{ix} & -a_{iy} & -a_{iz} \\ a_{ix} & 0 & a_{iz} & -a_{iy} \\ a_{iy} & -a_{iz} & 0 & a_{ix} \\ a_{iz} & a_{iy} & -a_{ix} & 0 \end{bmatrix} \tag{3.5}$$

and C_i is a matrix representation of vertex $c_i = (c_{ix}, c_{iy}, c_{iz})$, defined as:

$$C_i = \begin{bmatrix} 0 & -c_{ix} & -c_{iy} & -c_{iz} \\ c_{ix} & 0 & -c_{iz} & c_{iy} \\ c_{iy} & c_{iz} & 0 & -c_{ix} \\ c_{iz} & -c_{iy} & c_{ix} & 0 \end{bmatrix} \tag{3.6}$$

Quaternion q which is then used for rotation is computed as an eigenvector representing the largest eigenvalue of matrix M.

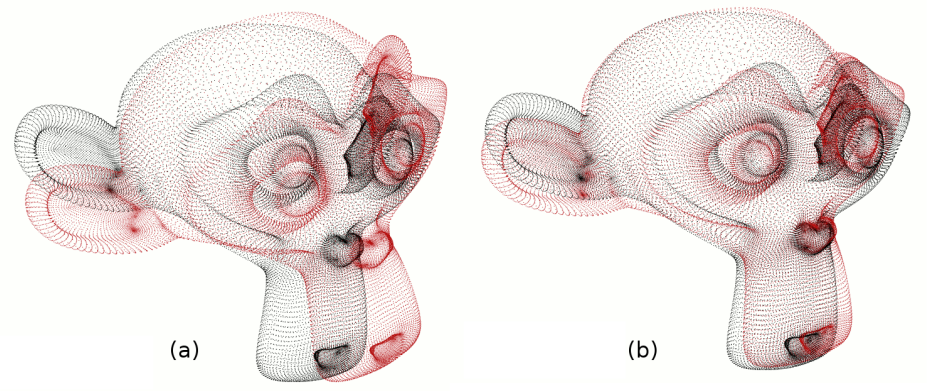

Figure 3.3: Example of alignment of two meshes using ICP algorithm after (a) one iteration and (b) seven iterations [6].

Original paper by Besl [12], showing solution for aligning two point sets one to another has several limitations, one of which was the assumption that both point sets have the same size. This was resolved in later years by adding mechanics to automatically establish the correspondence between meshes, such as the nearest neighbor search. Another of these limitations was lack of scaling parameter, which was resolved in several papers, for example [25] and [24]. The scaling of the mesh is presented in Figure 3.4.

Based on [24] we firstly need to extend minimization function, which we will be optimizing, to:

$$F(R,t,S) = \frac{1}{N}\sum_{i=1}^{N} \|RSa_i + t - c_i\|^2 \tag{3.7}$$

where S denotes scaling parameter.

In this approach, translation and rotation computation remains unchanged from methods mentioned above, however the computation of S assumes rotation was computed in matrix format. As we computed quaternion, we first need to compute rotation matrix. Having quaternion $q = (x, y, z, w)$, we get R where:

$$R = \begin{bmatrix} 1-2y^2-2z^2 & 2xy-2zw & 2xz+2yw \\ 2xy+2zw & 1-2x^2-2z^2 & 2yz-2xw \\ 2xz-2yw & 2yz+2xw & 1-2x^2-2y^2 \end{bmatrix} \tag{3.8}$$

We can then compute S as:

$$S = \frac{\sum_{i=1}^{N} {c_i}^T R E a_i}{\sum_{i=1}^{N} {a_i}^T E a_i} \tag{3.9}$$

where E is an identity matrix.

While sole computation of transformation parameters is relatively simple, finding correspondence remains bottleneck of the algorithm. To speed up the computation of finding the nearest neighbor we used *kD-trees* to partition space, enabling us to search through point set much faster.

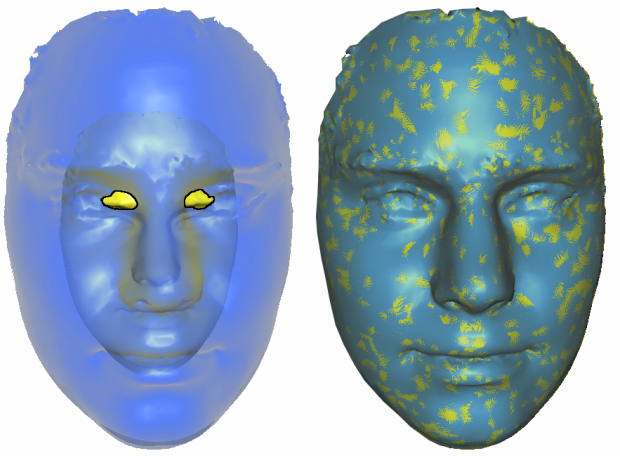

Figure 3.4: Example of SICP algorithm, showing (a) unscaled and (b) scaled model using SICP algorithm. Models displayed have partial transparency applied to show overlaid meshes.

Despite advantages of using *ICP* and *SICP* algorithms for registration, there are still limitations imposed by its implementation. Firstly, rough initial alignment is required, to be able to find the nearest neighbor efficiently. If this alignment is not provided, it is possible that many points of secondary face may be mapped to single vertex of main face, which may lead to erroneous computation of transformation parameters.

Secondly, we have to take the *ICP* error into consideration, as two meshes are rarely aligned perfectly. This does not pose large issue when comparing

two models, however, if more than two meshes are to be aligned, and there exists correspondence between each pair of the models, we might witness propagation of the error [10].

3.3 Mesh Analysis

There are a number of methods that can be used to analyze similarities between two or more models. In previous chapter we mentioned *Procrustes Analysis*, which computes distances between corresponding points of two configurations, and *Generalized Procrustes Analysis*, which computes mean shape from all configurations in analysis and uses it to compare configurations.

These methods are used in anthropology with facial points, a way to simplify description of the mesh to smaller subset of vertices. In our case, however, we work with entire mesh, taking advantage of all the information it can provide. For this purpose, we decided to utilize approach suggested in [34] and analyze meshes using *Hausdorff distance* metric and the nearest neighbor approach.

3.3.1 Hausdorff distance

Hausdorff distance, also known as Hausdorff metric, is a method which allows us to determine the extent to which each point of a main mesh lies near some point of secondary mesh [34]. In order to perform this computation, we need to at first establish correspondence between each point of main and secondary mesh. For this, we use same system as with *Iterative Closest Point* algorithm, by matching vertices based on their proximity. The meshes that are more similar will have distances between corresponding vertices smaller than those which are not. To achieve more accurate results, used models need to be aligned as close to each other as possible.

Let compared meshes be defined as set $A = (a_1, a_2, ..., a_n)$ and $B = (b_1, b_2, ..., b_k)$ of finite points, without imposing restriction of both having the same cardinality. We then define *Hausdorff distance* H as:

$$H(A, B) = max(h(A, B), h(B, A)) \tag{3.10}$$

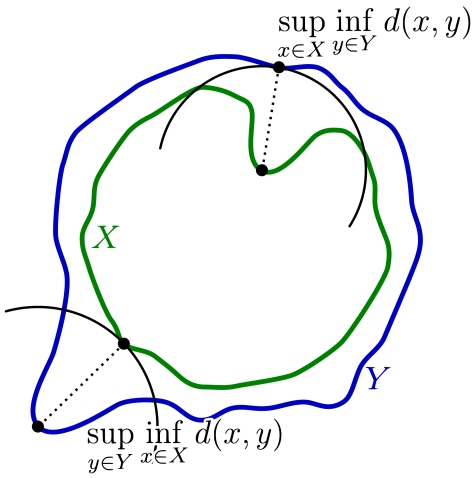

Figure 3.5: Visual representation of Hausdorff Distance [2].

where

$$h(A,B) = \max_{a \in A} \min_{b \in B} \|a - b\| \tag{3.11}$$

We can see function 3.11 computing distance of points in set A from set B, where $a \in A$ is identified as most mismatched point. Intuitively, this means, if we denote $h(A,B) = d$, distance of each point $a \in A$ to its corresponding point in mesh B is smaller or equal to d. Using function 3.11, which is called *Directed Hausdorff distance*, we can then compute difference between primary mesh B and secondary mesh A. Note that *Directed Hausdorff distance* is not symmetrical and depending on the vertex pairing in both meshes, $h(A,B)$ may not equal $h(B,A)$. This can also be seen in Figure 3.5.

3.3.2 Metrics for Analysis

Hausdorff metric allows us to see the difference between two sets of points by determining the distance of the most mismatched pair of points. This approach however, does not take into consideration defects of the meshes, such as the fact that edges of the mesh may be different due to process of capturing, which is especially common when comparing human faces. In [36] Krajíček suggests using thresholding, which would be able to effectively

eliminate values, which are too distant and in most cases could be considered just artifacts of the mesh.

In our case, rather than using maximal distance between corresponding points, we use *75 Percentile*. Disadvantage of manual thresholding by setting threshold value by user, is the fact that it has to be unique to each model. Instead, we automatically threshold twenty-five percent of the largest distances and only use maximal value of the thresholded values. Figure 3.6 displays computed set variation using *75 Percentile*.

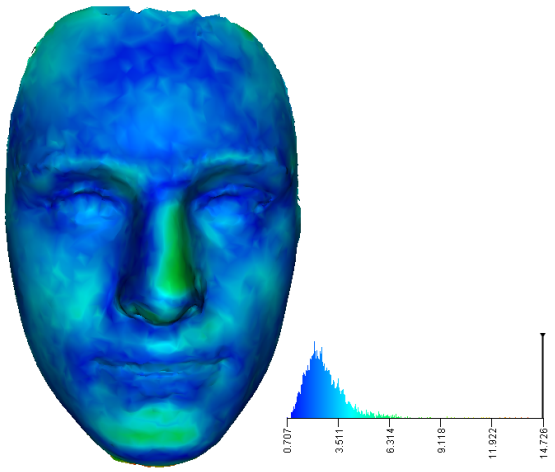

Figure 3.6: Visualization of mesh variation in the set using 75 percentile and absolute values of distances. Green color depicts the larger distances, while blue color depicts the smaller distances.

Similar to maximal distance, is *minimal distance*. Using the nearest neighbor criteria, we compute *minimal* distance from vector of the nearest neighbors $C = (c_1, c_2, ..., c_n)$:

$$minD = min(C) \tag{3.12}$$

Minimal distance allows us to analyze, how precisely are two meshes aligned. The value is smaller for models aligned better, and we can assume that if difference between minimal and maximal values is small, two shapes are similar.

Another method of analysis is to compute mean value of distances between corresponding points. We can do this by using *Arithmetic mean (AM)* of all values, computed as:

$$AM = \frac{1}{n}\sum_{i=1}^{n} c_i \tag{3.13}$$

where n is cardinality of vector of computed nearest neighbor distances for each vertex of secondary mesh, denoted as $C = (c_1, c_2, ..., c_n)$.

Using *Root Mean Square (RMS)* value, we can determine how much values vary from the mean value. Similarly to previously described methods, we expect variation of compared mesh to be much smaller if two models are similar. We can compute RMS as:

$$RMS = \sqrt{\frac{1}{n}\sum_{i=1}^{n} {c_i}^2} \tag{3.14}$$

Similar to *Root Mean Square* is *Geometric Mean (GM)*, which determines how values vary from zero. We compute GM as:

$$GM = \sqrt[n]{\prod_{i=1}^{n} c_i} \tag{3.15}$$

While zero value of *Geometric Mean* would show two meshes are identical and punishing deviations is logical, using this method in anthropology showed that, while it is suitable for human faces, it might not be beneficial for meshes which are more variable. Figure 3.7 shows histogram of distances between vertices of aligned models. Due to vertex-to-vertex metric used there is minimal number of values equal to zero. Therefore, in analysis of general mesh, *Root Mean Square* is preferred over *Geometric Mean* value.

Closely resembling *Root Mean Square* is also computation of variance. This value shows how much values vary between the minimal and maximal value. We can compute it as:

$$\sigma^2 = \sum_{i=1}^{n} p_i (c_i - E(C))^2 \tag{3.16}$$

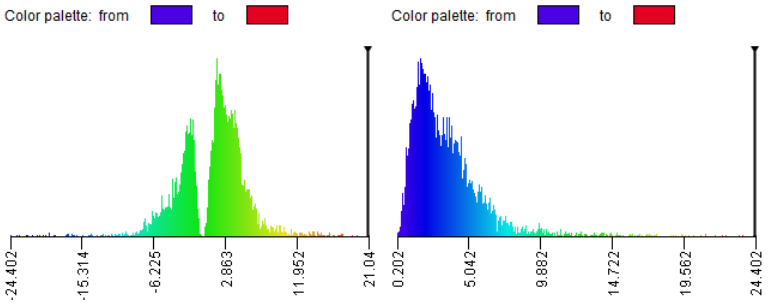

Figure 3.7: Histogram of distances between aligned models using relative and absolute values.

$$E(C) = \sum_{i=2}^{n} p_i c_i \tag{3.17}$$

where $E(C)$ is expected value of vector C of nearest neighbor distances, p_i is probability of occurring for c_i, for our purposes computed as:

$$p_i = \frac{1}{n} \tag{3.18}$$

and is same for each value of c_i.

When used within *FIDENTIS Analyst*, the preferred methods to analyze meshes by anthropologists were *Root Mean Square*, *75 Percentile* and *Geometric Mean*.

3.4 Mean Shape

When creating a mean shape of the set of meshes, we want it to take some characteristics from each model in the set. This can be described in the same way as in [14]:

$$S_{fin} = \sum_{i=1}^{m} a_i S_i \tag{3.19}$$

$$\sum_{i=1}^{m} a_i = 1 \tag{3.20}$$

where S_{fin} is a mean shape of set $S = (S_1, S_2, ..., S_m)$ and parameter a_i determines how much each model will contribute to the mean shape. Hence, if we want each mesh to contribute the same way, we set:

$$a_i = \frac{1}{m} \tag{3.21}$$

for $1 \leq i \leq m$.

In our algorithm, we compute the mean shape be determining translation vector for each vertex of the template. At the beginning, this template is chosen to be one of the models in set and it is then iteratively improved upon, based on the alignment of the meshes. We create a new mean shape by determining vector between each vertex of template mesh and its nearest neighbor in each of shapes in set S. Determining how each vertex will be distorted, we sum all the vectors for given vertex together and compute their mean value.

Let $S = (S_1, S_2, ..., S_m)$ be a set of shapes from which we pick template $T = S_i$ where $1 \leq i \leq m$. We consider template as point cloud $T = (t_1, t_2, ..., t_n)$. After aligning set using pair-wise registration method, such as *Iterative Closest Point*, we can compute translation vector u_k for each vertex of T as:

$$u_k = \frac{1}{m} \sum_{i=1}^{m} a_k - t_k \tag{3.22}$$

$$a_k = \min_{s \in S_i} \|t_k - s\| \tag{3.23}$$

where a_k is the nearest neighbor of vertex t_k in shape S.

We can then compute new position of each vertex of the mean shape T_k as:

$$t_{k_{new}} = u_k + t_k \tag{3.24}$$

for $1 \leq i \leq n$.

3.5 Compared Algorithms

In this thesis we evaluated our algorithm with two other possibilities of solving the same issue. Firstly, we evaluate it against classical pair-wise align-

ment, where we align and compare each mesh to every mesh in the set. This approach is used to compare two meshes and can be naturally extended to analyze all models in set by performing pair-wise alignment on each pair.

Second algorithm uses one of the shapes in the set and aligns all shapes to this template. This approach is used in identification process and as such already provides alignment of all meshes in the set.

Our algorithm is inspired by *Generalize Procrustes Analysis* and as such combines aforementioned algorithms. It aligns shapes of the set to mean shape, attempting to resolve error propagation of *Iterative Closest Point* algorithm.

Our algorithm aims to combine precision of pair-wise alignment with speed of aligning all shapes to the a single template. To compare these approaches, we use mesh comparison algorithm, which is same for all three approaches. We then compare triads of the most similar models, detected by each approach, the similarity of computed numerical results, and distances of our and second approach to results computed by the first approach.

4 Implementation

During the implementation phase of this thesis we encountered several issues, which we needed to resolve. Due to the fact we are usually working with large sets of data, which we are trying to process on personal computers, not specifically customized for this task, our algorithm had to be optimized to work within *FIDENTIS Analyst*, while using a relatively small amount of memory. Depending on the quality of the mesh, we were also attempting to minimize time necessary to process each mesh.

4.1 FIDENTIS Analyst

FIDENTIS is an open source project, developed at Faculty of Informatics, Masaryk University, Brno, in collaboration with Department of Anthropology, Faculty of Science, Masaryk University, Brno. It is divided into two parts, the *FIDENTIS Database*, database of faces of scanned volunteers, currently containing more than 2000 entries, and the *FIDENTIS Analyst*, application for analysis of meshes, primarily focused on the human faces. One of the goals of this thesis was to implement the algorithm for aligning and comparing data into the *FIDENTIS Analyst*.

The *FIDENTIS Analyst* is being developed in Java language, using JOGL, Java OpenGL binding, for rendering and NetBeans platform for UI design. The application allows for three different methods of mesh analysis.

First, the pair-wise mode, aligns and compares two meshes, primary and secondary, to each other. One of the main goals of this mode is to determine, whether both meshes depict the same person. It is also possible to use this mode to compare changes to individual's face, for example, during process of aging, or after plastic surgery.

Second, compare with database mode, allows to align all loaded meshes onto one primary mesh and compare them to the primary mesh. This mode serves mainly for identification purposes, where we try to determine the closest match of the primary face., e.g. for finding a suspect within the criminal database.

Third, so-called batch mode, employs algorithm described in this thesis to allow for comparing all loaded meshes. It is designed to work with large data sets, for example, to determine the facial variation in the population,

or determine the average shape of given models and quantify how meshes in the set differ from it.

4.2 Implementation of Batch mode

Our batch algorithm is trying to capitalize on advantages of both pair-wise and compare with database modes. It attempts to achieve quality of alignment closer to that of pair-wise comparison but to compute the results faster. As such, we currently use vertex-to-vertex metric for both mesh alignment with *Iterative Closest Point* algorithm and the nearest neighbor criteria of mesh comparison.

Finding the nearest neighbor is, however, the bottleneck of the computation. Rather than inspecting each vertex of the mesh to find one, which has the smallest distance, we decided to employ *kD-tree* data structure. This structure allows us to sort and partition space based on vertex position, creating tree structure, which we can search faster than array of values. This principle is demonstrated in Figure 4.1. To increase efficiency of the search, it is important to create balanced binary tree. In order to achieve this, before creating the tree itself, we first sort all vertices using merge sort algorithm, first by x value of their position vector, then by y value of their position vector and finally by z value of their position vector. This results in three sorted lists, which we can use to split given space, in our case three dimensional Euclidean space, in the middle, based on current depth of the tree.

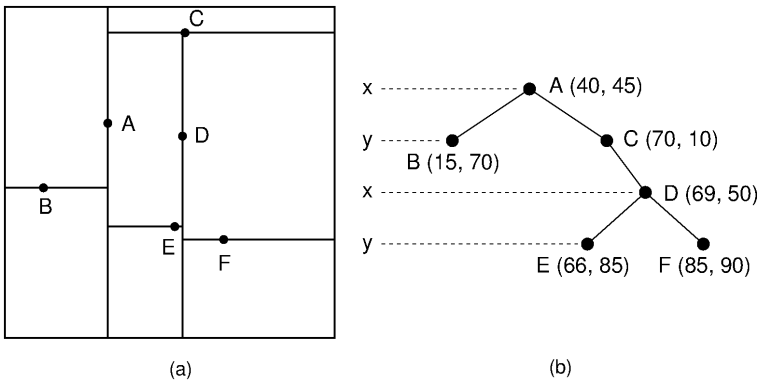

Figure 4.1: Visual representation of kD-Tree in 2D [4].

While using vertex-to-vertex criteria for searching the nearest neighbor of the point is straightforward once we have working *kD-trees*, due to the way of saving meshes in used formats, such as OBJ, it still has its disadvantages. Firstly, the quality of results differs based on quality of the mesh. When we use the nearest neighbor search to align and compare models, corresponding vertices might be too far away from each other, resulting in less accurate alignment and subsequently comparison. Distance between corresponding vertices is consequently decreased with increased sampling of the model.

Secondly, this form of criteria introduces information which is not entirely based on physical form of the mesh, as position and density of point cloud depends highly on scanning device, the procedure used to reconstruct 3D model and mesh decimation algorithm.

It is nonetheless a faster way to compute model comparison, as it can work directly with point clouds and there is no need to create additional data structure that would hold the edge or the face information.

The alternative to this approach is using point projection metric, where instead of computing vertex-to-vertex distance, we can measure distance of vertex to plane, in our case a triangle of polygonal mesh. Distance is determined as the distance between point and plane if our vertex is projected onto triangle, or distance between the point and the closest edge of the triangle in case the vertex is projected outside of the triangle. As we can see, this approach requires more information than the point cloud can offer on its own and is hence more computationally difficult.

When it comes to memory demands of the algorithm, we tried to minimize memory usage to only data which we require for computation. In alignment phase, we only keep template and models which are being aligned to it, while for comparison phase we only require final N:N matrix of results to be shown, along with the data required for visualization.

4.3 Memory Issues

As we mentioned previously, even though analysis of the sets of meshes is often computationally difficult task, which usually require larger amount of computer resources, such as memory space and processor usage, it was our aim to try to bring our algorithm even to personal computers, which are not specialized to handle the large data streams.

At first, we kept all meshes loaded in the memory, as the algorithm was extension of pair-wise module, where having two models in memory did not pose a problem. However, after attempting to analyze over 170 high definition meshes (e.g containing more than 50 thousand vertices per mesh), it quickly became apparent, this is not a suitable approach. Instead, we proceeded to only store meshes, that are being worked with, in the memory. These were the template and model which is being currently aligned to it. Models, which are already aligned in the iteration are then stored on the disk and are not loaded back into memory until they are required. This process is shown in Figure 4.2.

This allowed us to compute the alignment of models to the template, however, comparison of results proved to be another challenge. Due to need to recompute results, for example, when values are changed from relative to absolute, or due to thresholding, we needed to keep distances between each corresponding vertex pair for each model. This resulted in two-dimensional matrix for single model and three-dimensional jagged array for the final numerical results.

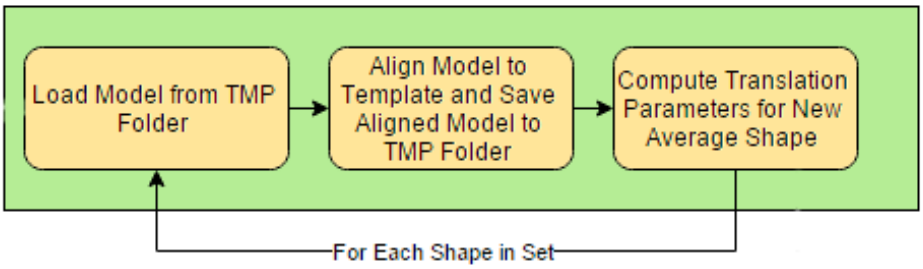

Figure 4.2: Diagram of computation of the alignment for a single model in Batch Mode.

When analyzing 170 high definition meshes, matrix for a single model already reached size of 700MB. As we couldn't store all the data in the memory, we once again had to move computed results into a temporary folder on the disk. This way we created folder structure, where for each model in the set of cardinality N we created a single folder, where we store the results of the pair-wise comparison as a single file.

4.4 Speed Issues

We already talked about the nearest neighbor search being the bottleneck of the algorithm. However there were several other speed related issues we needed to deal with over the course of implementation.

The speed of alignment using *Iterative Closest point* algorithm, without any adjustments, depends on the number of vertices in the model and as such it does not scale well, with increased mesh density. Nonetheless, in [43] the author pointed out, it is not necessary to use all the vertices of the mesh to compute alignment parameters. For example, in scene with incised plane, which is a model that is mostly flat, save for its middle, where we can see displacement, it is not necessary to search for correspondence between vertices of flat parts of the mesh. Instead, we only need to align displaced parts.

To allow for more efficient way of alignment, we implemented undersampling options. It allows user to only pick subset of vertices to compute alignment parameters from. Our current methods of undersampling are limited to *Random*, *Disk* and *Curvature*. In *Random* undersampling, we randomly pick certain number of samples of the mesh, based on the value input of the user. *Disk* undersampling allows to only consider vertices, which are in the distance greater than disk radius, computed as edge distance between them. However, neither of these methods take mesh morphology into account, which can lead into inaccurate alignment. Instead, we implemented undersampling based on the *Curvature*, which takes certain amount of vertices with the highest curvature. This allows to find significant vertices on the mesh, for example, with human faces, these tend to be area of nose, mouth, eyes and ears, taking shape of the model into consideration.

While undersampling is an easy and quick way to speed up the alignment algorithm, we cannot apply the same principle for comparison, either because we need all distance when computing visual results, or due to need to recompute results later on.

However, during computation we managed to identify similar tasks, and divide them into several groups. In the alignment phase, these tasks are aligning meshes to the mean shape, computing the translation vectors for new mean shape and creating new mean shape. In the comparison phase, these tasks are computing the new average face, computing visual results and computing numerical results.

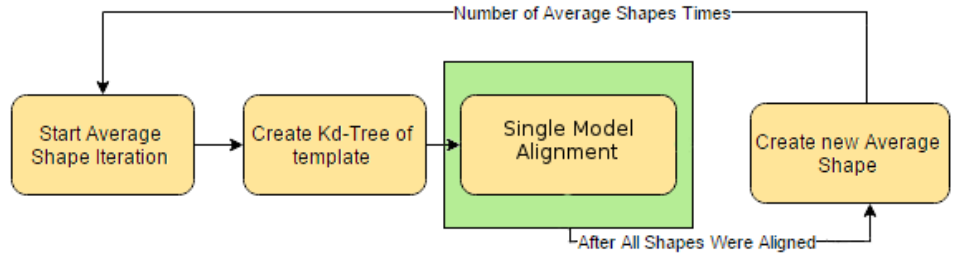

Figure 4.3: Diagram of computation of the alignment for entire Batch Mode.

When taking a closer look, we realized that several of these tasks are unrelated, they do not attempt to modify data, but instead return independent values, such as transformation parameters, or numerical results for single comparison.

This allowed us to determine good and relatively safe parallelization of the problem. In the end we decided to compute alignment in several threads, with each thread saving aligned faces to hard drive. Translation vectors for new mean shape are also computed in separate thread for each face, since we only get vectors as result, but do not yet modify the template. On the other hand, we create new mean shape sequentially. In the end, this task is reduced to adding vectors to vertices of the template and as such, we decided it was relatively fast operation, which could run sequentially.

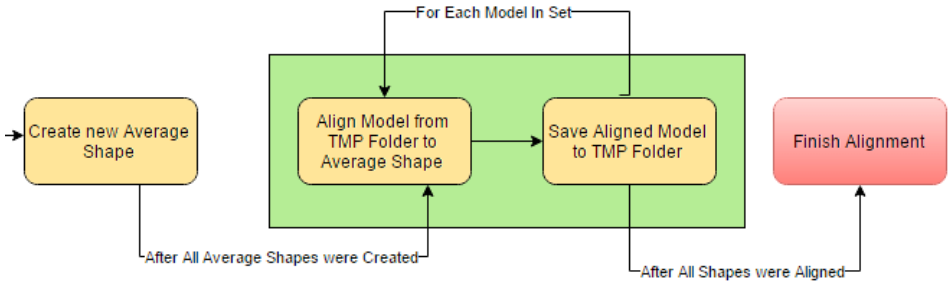

Figure 4.4: Diagram of alignment in Batch Mode, after all mean shapes were created.

In Figure 4.3 and Figure 4.4 we show process of computation of our algorithm, with parallelized tasks being encapsulated in green rectangle.

In the comparison phase, all tasks are performed in parallel as well, safe for computation of the average shape. Number of threads we run for each phase is equal to number of available processors. This also takes into consideration hyper-threading of modern CPUs. We originally attempted to set number of used threads manually, but exceeding number of available processors led to decrease of the application performance and rendered entire system practically unusable.

5 Evaluation

In this thesis we presented algorithm for analysis of sets of data based on aligning all meshes to the mean shape of the set. In the evaluation phase, we were interested in determining its precision and speed, compared to two other approaches which could be used to deal with the same problem. These are either pair-wise alignment of all possible pairs of meshes, or aligning all models to single mesh in the set.

5.1 Input data

While this algorithm is aimed to work on general meshes, for our evaluation we used models of human faces. These were acquired by scanning volunteers, with 3D model reconstruction performed by the scanning software.

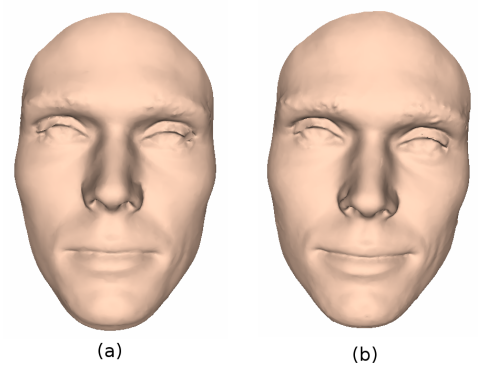

Figure 5.1: Example of used data, (a) neutral facial expression and (b) smiling facial expression of the same individual.

As with most scanning devices, the process of mesh acquisition introduced unwanted artifacts, such as noise around areas where hair occurred, for example, eyebrows, facial hair, or top of the head. For this reason the models were preprocessed, removing parts of the face, such as ears. The edges of the mesh were also manipulated to avoid jagged effect.

We divided our data into two main groups. First, containing four individuals, each in neutral facial expression and with slight change in facial expression, as can be seen in Figure 5.1. We decreased the number of vertices

for this group to 10 thousand, to unify meshes and to avoid any effects the density of model would have on the results.

Second group contained only meshes of unique indivudals in neutral facial pose, as shown in Figure 5.2. This group contained twenty models. Original, high definition meshes had large variance of number of vertices, starting from 50 thousand up to approximately 80 thousand vertices. To be able to evaluate speed performance of the algorithm, we unified mesh sampling of each face, creating three subgroups – 5 thousand, 10 thousand and 20 thousand vertices groups. We used these groups to evaluate speed and quality of the algorithms.

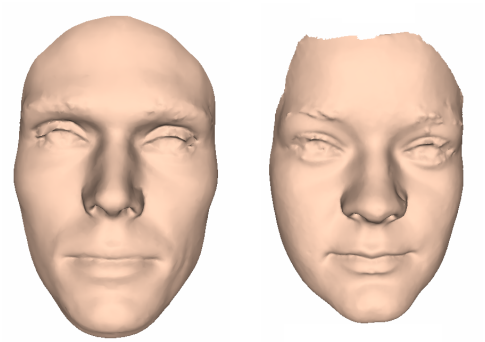

Figure 5.2: Example of used data, two different individuals.

5.2 Evaluation parameters

Our goal was to create the algorithm that resembles pair-wise alignment in quality, and the alignment to one mesh in its speed. In order to be able to compare quality of the alignment of the algorithms, we used mesh comparison algorithm, which is same for all the methods. We expect our results to be closer to pair-wise approach of alignment, when using our algorithm than when alignment to one mesh is used. More precise explanation of this approach is provided in Quality Evaluation sub chapter 5.5.

While we were able to run our algorithm in more threads, to improve on its speed, in order to be able to compare all algorithms together, we decided to run all of them in one thread only. The speed evaluation sub chapter 5.3, however also shows how time of computation improves when more threads are used.

All of our algorithms use *Scaling Iterative Closest Point* algorithm for the alignment. We also use different values for maximum allowed iterations and error rate. For maximum number of allowed iterations, we evaluated algorithms using value 4, 20, 40 and 60, for each of three groups, in combination with error rates of value 0, 0.01, 0.05, 0.1 and 0.5. We also used 3 iterations of mean shape computation in our algorithm, with template being first mesh in the set to compare result with other two algorithms.

For the separate evaluation of our algorithm we tried to determine how changing the mean shape template affects acquired results. We also evaluated how mean shape changes in each iteration.

Since we are using three different algorithms, we tried to pair results to match used alignment parameters. For both pair-wise alignment and alignment to single model, finding corresponding parameters was simple. Both algorithms use same number of maximal allowed iterations and error rate, e.g. if pair-wise alignment was performed with 40 max. iterations and error rate of 0.05, so was alignment to single mesh.

Our algorithm uses slightly different approach, as stated number of the mean shapes will always be created. In the end we decided to pair parameters, which add up to the same number of maximal iterations. If pair-wise alignment was performed with 40 max. iterations and 0.05 error rate, our algorithm, ran with 3 mean shapes, will be performed with 10 max. iterations and 0.05 error rate. This added up to maximal 40 iterations during our algorithm, as max. 10 are performed with each created shape and 10 more can be performed when aligning to final mean shape.

We evaluated our algorithms using mainly *Root Mean Square*, although other metrics, primarily *75 Percentile* and *Geometric Mean* were considered as well. Due to large amount of data evaluated, not all of the results are presented in this thesis, as we only decided to present those, we found most interesting and expressive.

For the rest of the chapter we will use *To Each* to denote pair-wise alignment, *To One* to denote alignment to one mesh and *To Average* for our algorithm. For used mesh groups, we use 5k for group of meshes with 5 000 vertices, 10k for group with 10 000 vertices and 20k for group with 20 000 vertices.

5.3 Speed Evaluation

We evaluated the speed performance of the algorithms by measuring time it took for computation of each of them to return the results. Several tasks, varying based on the performed algorithms were measured.

For each of the algorithms we measured how much alignment took, considering entire process of alignment, but also separate runs of *Scaling Iterative Closest Point* algorithm, to determine how long it took for *kD-tree* to be created and how much of taken time is the actual alignment. We also measured how long it took for comparison of set of meshes to be performed. Number of iterations performed was also noted down, to determine when algorithm converges with given parameters, and to see how long it took. For our algorithm we also measured time necessary to create the new mean shape.

We performed evaluation on the system using these specifications:

Operating system: Windows 10 Education x64-based processor

CPU: Intel(R) Core(TM)2 Quad CPU Q9550 @ 2.83GHz

RAM: 8.00 GB

GPU: NVIDIA GeForce GTX 470

Results of alignment for default parameters used within the *FIDENTIS Analyst* for our algorithm, e.g. 10 iterations, 0.05 error rate, 3 mean shapes, corresponding to 40 iterations, 0.05 error rate for remaining algorithms are shown in table 5.1.

Actual maximal number of iterations of SICP performed shown in Table 5.2

	To Each	**To One**	**To Average**
5k	182.6194 s	5.7589 s	45.5401 s
10k	308.4175 s	84.8640 s	157.8220 s
20k	790.7471 s	252.1516 s	540.6621 s

Table 5.1: Time of computation for default parameters in the FIDENTIS Analyst.

	To Each	To One	To Average
5k	17	9	25
10k	17	10	25
20k	18	10	22

Table 5.2: Maximal number of performed iterations of SICP algorithm with default parameters for each approach.

As we can see from the results, our algorithm has in general better speed than pair-wise alignment algorithm, despite needing to perform more iterations. We can also observe the scaling of algorithm with increasing density of the mesh is better than the one of pair-wise alignment algorithm. Originally we assumed, the creation of *kD-Trees* is one of the most costly operations in algorithm.

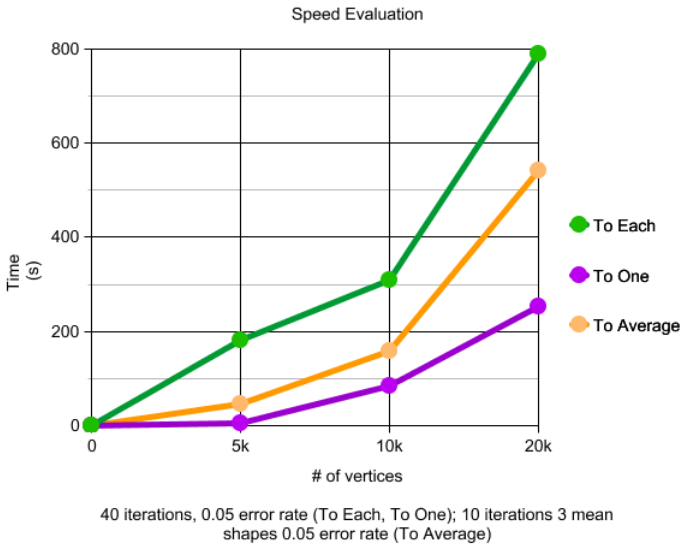

Figure 5.3: Graph showing time increase based on density of analyzed mesh for each approach.

For this reason we looked into the speed performance of all three algorithms when number of iterations performed is the same. We set parameters to 15 iterations, 0 error rate and 3 mean shapes for our algorithm, corre-

sponding to 60 iterations and 0 error rate for remaining algorithms. Table 5.3 shows computed times for each algorithm.

	To Each	**To One**	**To Average**
5k	660.1926 s	35.4451 s	69.0136 s
10k	1605.4392 s	88.0112 s	211.7501 s
20k	3800.9140 s	205.9599 s	678.0431 s

Table 5.3: Time of computation for 60 performed iterations.

We notice a rapid increase in time of computation, when compared to results in Table 5.1, especially with pair-wise alignment algorithm. However, the pair-wise algorithm was optimized to create as few new *kD-trees* as possible without affecting quality of alignment. Table 5.4 shows number of *kD-Trees* created for each algorithm.

To Each	**To One**	**To Average**
20	1	64

Table 5.4: Number of kD-Trees created for each algorithm when aligning 20 faces and creating 3 average shapes.

The large amount of *kD-Trees* created for our algorithm is due to the mean shape creation, where we need to find the nearest neighbor for each vertex of template in each mesh of the set, hence creating 20 new *kD-Trees* every time we compute the new average mesh. Nonetheless our algorithm appears to be faster than the pair-wise algorithm, despite requiring more time on creating structures to help to find the nearest neighbor.

When trying to determine the reason for this, we analyzed number of lookups into *kD-Tree* to find the nearest vertex, for each of the algorithms. For simplicity, let's assume each model of set has same number of vertices and that there were sixty iterations performed for each face alignment, with each algorithm.

For *To Each* algorithm this means, there were 400 pair-wise alignments performed, each taking sixty iterations. For *To One* algorithm, there were then 20 pair-wise alignments performed, each taking sixty iterations. Finally,

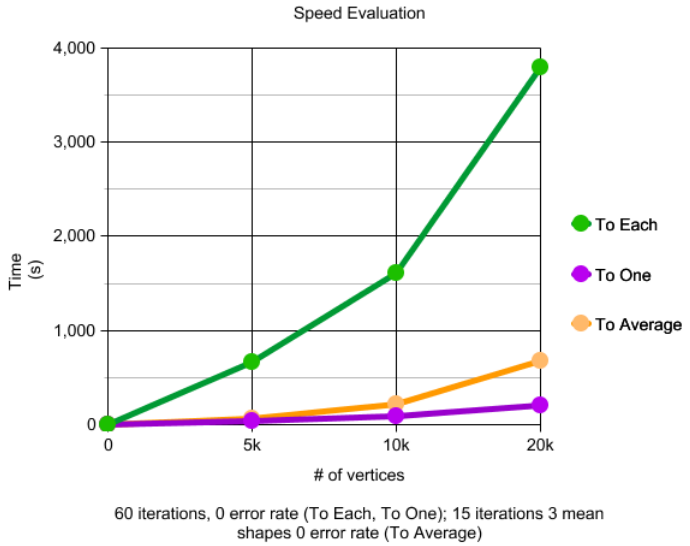

Figure 5.4: Graph showing time increase based on density of analyzed mesh for each approach with 60 alignment iterations performed for each mesh in the set.

for our algorithms, this means there were 40 pair-wise alignments performed, each taking fifteen iterations and another 20 lookups for single mean shape creation, adding up to sixty additional lookups in total, to create three average shapes.

Table 5.5 shows number of lookups into *kD-Tree* to search for the nearest neighbor for each algorithm, in this scenario. In the table n is the number of vertices of the mesh. All of these results are solely for alignment phase.

To Each	**To One**	**To Average**
24 000n	1 200n	1260n

Table 5.5: Number of seraches for the nearest neighbor in the kD-Tree sturcture.

We can see the pair-wise alignment algorithm has excessive number of the nearest neighbor queries, compared to the remaining two algorithms. The difference in the speed between our algorithm and *To One* method can then be explained by larger number of *kD-Trees* created by our approach.

This allows us to generalize our original assumption, stating it was the creation of *kD-Trees* which was the bottleneck of ICP-based algorithms for mesh alignment, and state it is in fact the entire process of the nearest neighbor search, which is computationally difficult.

As mentioned previously we also analyzed our algorithm when running parallely, using four threads, our approach computed alignment with given times for each group, using default parameters employed in the *FIDENTIS Analyst*. The results can be seen in Table 5.6.

5k	**10k**	**20k**
22.8797 s	57.1674 s	187.9513 s

Table 5.6: Time of alignment for To Average algorithm running in four threads for default parameters.

We can see rapid improvement in speed, especially with meshes with more vertices. It is important to mention, that our algorithm divide computation into separate threads based on the tasks described in Implementation chapter 4 and does not process forward with computation until all the parts of the task are computed. This means, that our algorithm will run more efficiently when number of meshes in the set is dividable by number of available processors.

5.4 Validation of the algorithms

In order to validate all used algorithms, we used group of four individuals. Each individual in the set was captured with neutral facial expression and with slightly changed facial expression. This resulted into set of eight meshes, two per individual. To show all algorithms are able to find similar faces, we determined most similar mesh to each mesh in the set, using *Root Mean Square* metric. Results of the evaluation are shown in Table 5.7.

Model 1_a is face with slightly changed facial expression and 1_b is a face with neutral facial expression. Same naming conventions apply to remaining pairs as well.

We can see that each of the algorithms managed to detect the most similar individual correctly. Moreover, the differences between numeric results

Same pairs	**To Each**	**To One**	**To Average**
1_a	1_b	1_b	1_b
1_b	1_a	1_a	1_a
2_a	2_b	2_b	2_b
2_b	2_a	2_a	2_a
3_a	3_b	3_b	3_b
3_b	3_a	3_a	3_a
4_a	4_b	4_b	4_b
4_b	4_a	4_a	4_a

Table 5.7: Most similar mesh detected for group containing same individuals.

of facial comparison of the same individuals are much smaller than those of different ones, as seen in Table 5.8.

Same Pairs	**To Each**	**To One**	**To Average**
1_a	1.2465, 2.7464	1.2070, 3.0176	1.1099, 2.8307
1_b	1.2465, 3.0902	1.2070, 3.4049	1.1099, 3.0834
2_a	1.1531, 2.4155	1.1072, 2.4828	1.0898, 2.4680
2_b	1.1531, 2.3598	1.1072, 2.3716	1.0898, 2.3379
3_a	1.1399, 2.4006	1.1510, 2.5054	1.1199, 2.4272
3_b	1.1399, 2.3598	1.1510, 2.3716	1.1199, 2.3379
4_a	1.2817, 2.4006	1.3789, 2.4957	1.3244, 2.4272
4_b	1.2817, 2.6054	1.3789, 2.7467	1.3244, 2.6149

Table 5.8: Root Mean Square values of comparison between group containing same individuals, showing values for two most similar meshes detected.

5.5 Evaluation of quality of the alignment

When deciding on methods to evaluate quality of the alignment, we concluded, it was sensible for us not to focus on comparison of results with real-life objects, but rather to determine ability of our algorithm to identify the most similar meshes. This is due to the fact, that human faces, on which we concluded our evaluation are in general more specific than arbitrary meshes and approaches to compare them extend past mere alignment techniques.

For this reason, we took results of *To Each* algorithm as the most precise outcome we can achieve using *Scaling Iterative Closest Point* algorithm and try to determine which algorithm matches them more closely.

5.5.1 Comparing three algorithms

To evaluate the precision of all three approaches we chose to detect three most similar meshes for each model in the set, using different parameters of alignment. Another technique of evaluation was to determine distance from results of *To Each* approach, using *Root Mean Square* metric acquired for *To One* and *To Average* algorithms. Finally, we also took statistical approach to evaluation, employing *Mantel Test*, described in [39]. This technique tests correlation between two matrices, resulting into percentage similarity of two matrices as well as statistical significance parameter, commonly denoted by p.

To show how density of mesh affects alignment and comparison of meshes using vertex-to-vertex metric and how both *To One* and *To Average* algorithms perform comparing to *To Each* algorithm, we used default values of 40 max. iterations, 0.05 error rate and *Root Mean Square* technique, corresponding to 10 max. iteration, 0.05 error rate and 3 mean shapes for our approach.

Tables 5.9, 5.10 and 5.11 provide summary for results of most similar triads of models for each of the mesh groups, while full tables can be found in Appendix A.

From these results we can observe the precision of the algorithm varies depending on the mesh density, due to vertex-to-vertex metric used. Especially with group with five thousand vertices and ten thousand vertices we can see the number of correctly detected triads doubled when using our algo-

5k Group	**To One**	**To Average**
Correct	4	5
Wrong order	2	3
One mismatched	12	12
Two mismatched	2	0

Table 5.9: The result summary of the most similar triads of meshes for 5k group and default parameters from *FIDENTIS Analyst* software for To Average and To One algorithms from Table 6.1.

10k Group	**To One**	**To Average**
Correct	5	10
Wrong order	1	1
One mismatched	11	9
Two mismatched	3	0

Table 5.10: The result summary of the most similar triads of meshes for 10k group and default parameters from *FIDENTIS Analyst* software for To Average and To One algorithms from Table 6.2.

20k Group	**To One**	**To Average**
Correct	4	9
Wrong order	2	2
One mismatched	12	9
Two mismatched	2	0

Table 5.11: The result summary of the most similar triads of meshes for 20k group and default parameters from *FIDENTIS Analyst* software for To Average and To One algorithms from Table 6.3.

rithm. In general we observe our algorithm shows larger number of correctly detected triads, smaller number of amount of triads detected with one face mismatched and no triads with two faces mismatched.

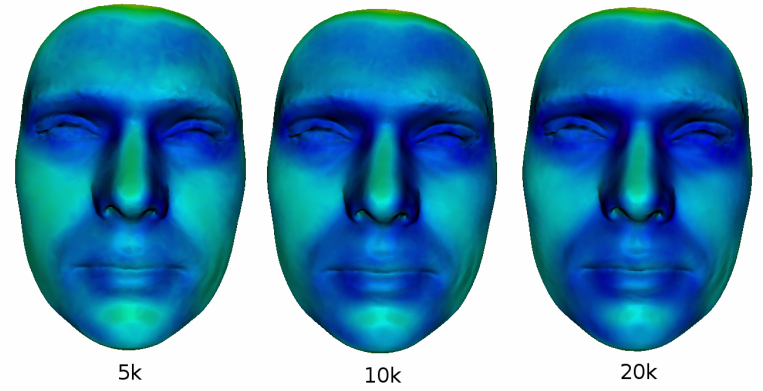

Figure 5.5: Visual representation of variation of meshes in the set on the average shape in each of analyzed density groups.

The group of ten thousand vertices appears to be dense enough, so that performing same algorithm on the group with twenty thousand vertices does not vary much. Figure 5.5 shows visual representation of variation of meshes in the set, displayed on the average shape, for each analyzed density group. Dark blue color depicts smaller differences, while green and red colors depict larger differences.

Second method of comparing quality of the alignment was comparing the distance of *To One* and *To Average* algorithm to values of *To Each*, using *Root Mean Square* metric. As the computed matrix would be too large, we only show simplified table representation, showing either plus (+) when *To One* algorithm is closer to *To Each*, or minus (-), when *To Average* algorithm is closer to *To Each*. In case both values are the same, table shows 0. We also color coded the result for easier readability, showing minus cells in green and plus cells in red. These results are computed on group of twenty thousand vertices, using 40 max. iterations, 0.05 error rate and *Root Mean Square* metric. The results can be seen in Figure 5.6.

20k	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
1	*0*	+	-	+	-	-	-	-	-	-	-	-	+	-	-	-	-	-	-	-
2	+	*0*	+	+	-	-	+	-	-	-	-	-	-	-	+	-	-	-	-	-
3	-	+	*0*	-	-	-	-	-	-	-	-	-	-	-	+	-	-	-	-	-
4	+	+	-	*0*	-	-	+	+	+	+	+	+	+	+	+	-	+	-	+	-
5	-	-	-	-	*0*	+	+	+	+	-	-	+	+	-	-	-	-	-	-	-
6	-	-	-	-	+	*0*	-	-	-	-	-	+	-	-	-	-	-	-	-	-
7	-	+	-	+	+	-	*0*	+	+	+	+	+	+	+	-	+	+	-	+	+
8	-	-	-	+	+	-	+	*0*	+	+	+	+	+	+	-	+	+	+	+	-
9	-	-	-	+	+	-	+	+	*0*	+	+	+	+	+	-	+	+	-	+	+
10	-	-	-	+	-	-	+	+	+	*0*	+	+	+	+	-	-	-	-	-	+
11	-	-	-	+	-	-	+	+	+	+	*0*	+	+	+	-	+	+	-	+	+
12	-	-	-	+	+	+	+	+	+	+	+	*0*	+	+	+	+	+	+	+	-
13	+	-	-	+	+	-	+	+	+	+	+	+	*0*	-	-	-	-	+	-	+
14	-	-	-	+	-	-	+	+	+	+	+	+	-	*0*	-	+	+	+	+	+
15	-	+	+	+	-	-	-	-	-	-	-	+	-	-	*0*	-	-	-	-	-
16	-	-	-	-	-	-	+	+	+	-	+	+	-	+	-	*0*	+	+	+	-
17	-	-	-	+	-	-	+	+	+	-	+	+	-	+	-	+	*0*	+	+	-
18	-	-	-	-	-	-	-	+	-	-	-	+	+	+	-	+	+	*0*	+	-
19	-	-	-	+	-	-	+	+	+	-	+	+	-	+	-	+	+	+	*0*	-
20	-	-	-	-	-	-	+	-	+	+	+	-	+	+	-	-	-	-	-	*0*

Figure 5.6: Distances of RMS values from To Each algorithm computed on 20k vertices group. Green color shows results where To Average algorithm is closer, while red color shows results where To One algorithm is closer.

This results into *To Average* being closer to results of *To Each* 204 times out of 400, while *To One* being closer to results of *To Each* 176 times out of

20k Group	**To One**	**To Average**
Correct	4	9
Wrong order	3	2
One mismatched	11	7
Two mismatched	2	2

Table 5.12: The result summary for the most similar triads of meshes for 20k group, 20 iterations and 0.05 error rate from Table 6.4.

20k Group	**To One**	**To Average**
Correct	3	7
Wrong order	2	4
One mismatched	13	7
Two mismatched	2	2

Table 5.13: The result summary for the most similar triads of meshes for 20k group, 40 iterations and 0.1 error rate from Table 6.5.

400 and 20 times distances being the same. This shows slight advantage of our algorithm, when it comes to distances of results to *To Each* alignment.

We also wanted to determine how number of maximal iterations affects precision of the algorithms. We only show table of detected triads for each algorithm with parameters of 20 max. iterations, 0.05 error rate, using *Root Mean Square* metric, corresponding to 5 max. iterations, 0.05 error rate and 3 mean shapes created for our algorithm performed on group with twenty thousand vertices.

Table 5.12 provides summary of Table 6.4 found in Appendix A. It shows increased number of two mismatches detected with our algorithm. As pointed out in Speed Evaluation chapter our algorithm takes longer to converge, due to changing shape of the template and this indicates, that five iterations per creation of the new mean shape is not enough.

On the other hand, when comparing effect of error rate parameter on alignment for each approach, using 40 max. iterations, 0.1 error rate and *Root Mean Square* metric, corresponding to 10 max. iterations, 0.1 error rate and 3 mean shapes for our algorithm, we get results on group with twenty thousand vertices as shown in Table 5.13.

When compared to same computation, with error rate set to 0.05 (see Table 5.11), we can see the number of correctly detected triads decreased. *To One* and *To Average* algorithm show increased detection of triads with one

20k Group	**To Each**	**To One**	**To Average**
To Each	100%	95.09%	94.78%
To One	95.09%	100%	98.44%
To Average	94.78%	98.44%	100%

Table 5.14: Mantel Test for RMS results of meshes comparison for 20k group, 20 iterations and 0.05 error rate.

and two mismatches. Nonetheless our algorithm still detects more correct triads.

Table 5.14 compares similarity of matrices for 20 max. iterations with 0.05 error rate, on group of meshes with twenty thousand vertices, using *Mantel Test*.

Table 5.15 compares similarity of matrices for 40 max. iteration with 0.05 error rate.

20k Group	**To Each**	**To One**	**To Average**
To Each	100%	95.09%	94.73%
To One	95.09%	100%	98.38%
To Average	94.73%	98.38%	100%

Table 5.15: Mantel Test for RMS results of meshes comparison for 20k group, 20 iterations and 0.05 error rate.

These results show, that statistically, there is not much difference between results by either algorithm, when compared to *To Each* method, as both have $p = 0.0001$.

Analyzing error rate of 0.01 and 0 we discovered these values are in fact too small, for algorithms to converge, resulting into oscillation of results, which may lead to inaccurate detection of similar meshes, if differences between comparisons are small. These results are not shown in this thesis, however they are attached to this thesis repository.

5.6 Comparison of Average Shapes

By default we created three mean shapes to compare our algorithm with remaining two. We also used first mesh in the set to create the template from.

However, we are also interested in determining whether choosing different base model for template has effect on alignment, and if it does, whether it can be resolved when more mean shapes are created. We also wanted to demonstrate how much mean shape changes in each iteration.

First we compared triads of the most similar faces found by our algorithm with the base for the template changing. We used 10 max. iterations, 0.05 error rate, using *Root Mean Square* metric, on the group of meshes with ten thousand vertices. For simplicity we only show triads of the most similar meshes found with first mesh of set as the base for the template and only note whether same triad was found for other bases. If the triad found was the same, corresponding table cell contains plus sign (+) and is colored in green, otherwise, it contains minus sign (-) and is colored in red.

10k	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
1	*15, 2, 3*	+	+	-	+	+	-	+	-	+	+	+	+	+	+	+	-	+	+	+
2	*3, 1, 5*	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
3	*2, 4, 15*	-	+	+	-	-	-	-	-	-	+	-	-	-	+	-	-	-	-	-
4	*3, 11, 15*	+	+	+	+	+	-	+	+	-	+	+	+	-	+	+	-	+	+	-
5	*6, 13, 17*	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
6	*5, 12, 13*	-	-	-	-	+	-	+	-	+	-	+	-	-	-	+	+	+	+	+
7	*9, 18, 10*	+	+	+	+	+	+	+	+	+	+	+	+	+	-	+	+	+	+	+
8	*9, 18, 10*	+	-	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
9	*10, 18, 8*	+	+	+	-	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
10	*9, 11, 13*	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
11	*10, 9, 20*	-	+	-	-	-	+	+	+	+	+	-	-	-	+	+	+	+	+	+
12	*16, 17, 6*	-	-	-	-	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
13	*5, 10, 11*	-	+	+	+	-	-	-	-	+	-	+	+	+	+	+	+	-	+	+
14	*18, 10, 11*	-	+	+	+	+	+	+	+	+	+	+	+	+	-	+	+	+	+	+
15	*1, 3, 4*	+	+	+	+	+	-	+	+	+	+	+	+	+	+	+	+	+	+	+
16	*17, 12, 5*	+	-	+	+	+	+	+	+	+	+	+	+	+	-	+	+	+	+	+
17	*16, 12, 5*	+	+	-	-	-	+	+	-	-	+	+	-	+	+	-	-	-	+	+
18	*9, 14, 10*	-	-	-	-	-	-	-	-	-	-	-	-	-	+	-	-	-	-	-
19	*18, 8, 9*	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	-	+	+	+
20	*10, 18, 11*	-	-	-	-	-	-	-	-	-	-	-	-	-	+	-	-	-	-	-

Figure 5.7: Triads of most similar faces detected when first mesh was set as base for template and when base was changed.

The Figure 5.7 shows, that changing the template our algorithm was able to identify same triads in 273 cases out of 380 and it determined different triads in 107 out of 380 cases, with faces number 18 and 20 being the most erroneously detected. At the same time, mesh number 7 had the most differently detected triads. When creating five mean shapes, the detected triads changed minimally compared to triads detected when creating three mean shapes.

Using the *FIDENTIS Analyst* we created the average shape of the set using mesh number 7 as the base and compared it to the mean shape of the set created using first mesh as the base.

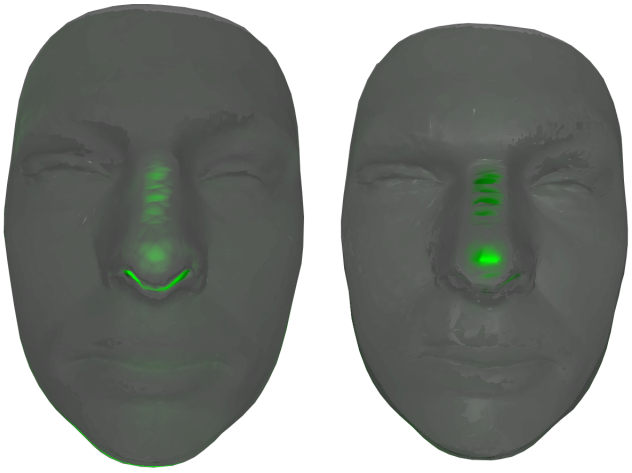

Figure 5.8: Comparison of (a) third average shape when using first and seventh model as a template base and (b) fifth average shape when using first and seventh model as a template base.

The Figure 5.8 *(a)* shows differences of two mean shapes, with higher saturation showing greater differences of overlaid meshes. We can see that in our case, mean meshes differ especially in area of nose and slightly in the area of the mouth. We also analyzed differences using *Root Mean Square* metric, resulting into value 1.3701. When determining if this difference can be decreased with the larger number of the mean shapes created, we compared fifth iteration of creating mean mesh using both first and seventh model as the base. The results of comparison are shown in Figure 5.8 *(b)*.

The difference between two meshes in area of the mouth seems to have vanished, however difference of the nose area is still pronounced. *Root Mean Square* metric of comparison of the two meshes now results into value of 1.1517, showing slight improvement with the larger number of mean meshes created.

The visual results in both cases, when first and seventh mesh was set as the base for the template, detected same areas with the largest deviations

within set of meshes, as can be seen on the Figure 5.9. These are the areas colored in red, orange and yellow, more specifically the nose, the chin and area of cheeks.

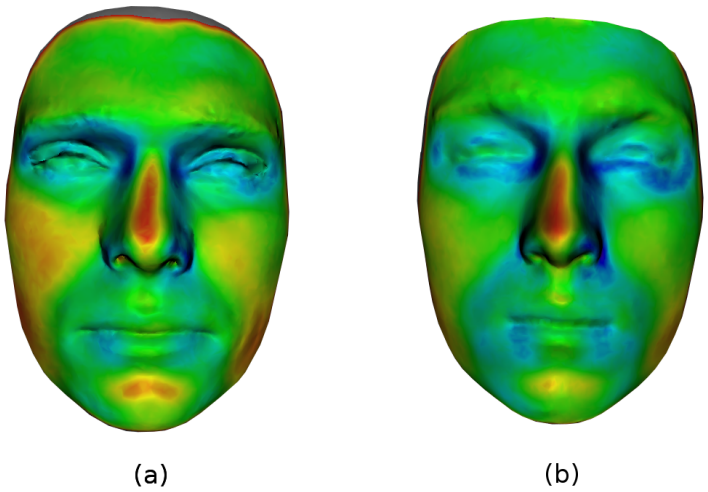

Figure 5.9: Variation of the meshes in the set displayed on the mean shape (a) with first model as a template base, (b) with seventh model as a template base.

Nonetheless, creating the large number of the mean shapes can excessively prolong the computation. We analyzed how mean shape changes in each iteration to determine how many mean meshes we needed to create, for algorithm to converge.

Figure 5.10 *(a)* shows comparison of first iteration of the mean shape to base of the template. In this case, first iteration of the mean shape creation shows large deviations from original model, especially in area of cheeks and jaw. Numerical comparison of base mesh and first average shape, using *Root Mean Square* metric resulted in value of 2.0212.

Comparing third and first mean shape shows much smaller deviations, as can be seen on the Figure 5.10 *(b)*, with *Root Mean Square* value resulting into 0.9569. When comparing fifth and first mean shape, *Root Mean Square* value resulted into 1.0475.

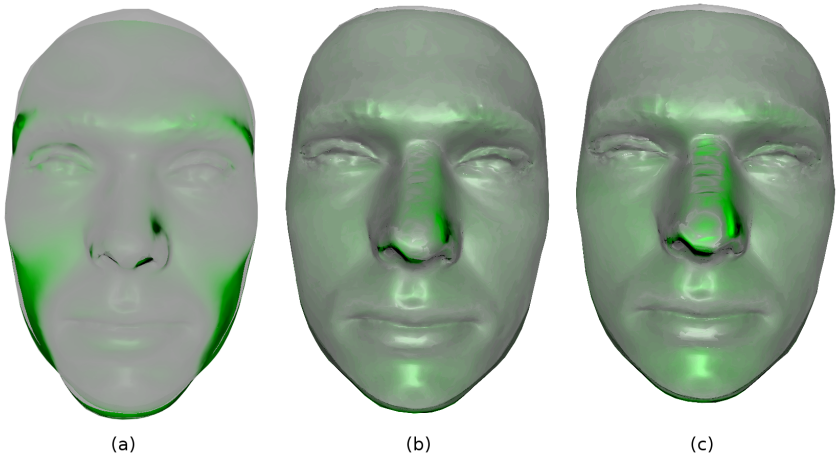

Figure 5.10: Differences between (a) first average shape with first model as a template base and the first model, (b) third average shape and first average shape and (c) fifth average shape and first average shape.

The numerical results show, that with each iteration the changes of mean shape are smaller, and that differences between third and fifth mesh are not pronounced. This shows, that instead of creating more mean shapes, the choice of the base of the template has larger impact on quality of alignment than number of created mean shapes.

5.7 Evaluation Results

In this chapter we used several methods to compare our algorithm, with two approaches which can be used to resolve same problem. For the speed evaluation we measured time necessary to compute alignment of the meshes for each algorithm. The results show, that our approach performs and scales better than *To Each* approach and also that we can have substantial speed improvement, when more than on thread is used for computation.

In precision evaluation, we compared triads of the most similar meshes found by each algorithm and compared results to the triads found by *To Each* approach. For all three density groups, our algorithm provided results closer to *To Each* algorithm, while at the same time, unlike *To One* approach, it didn't detect any triad with two faces different from that detected by *To Each* approach.

Comparing distances of *Root Mean Square* results for each algorithm show our approach having slight advantage, however, *Mantel Test* showed matrices of results for *To One* algorithm being more similar to that of *To Each* approach.

6 Conclusion

The aim of this thesis was to design and implement the algorithm which would allow to compare and analyze all the meshes in the set and include it in the *FIDENTIS Analyst* — tool for analyzing 3D meshes, with focus on the human faces. This algorithm was successfully implemented into Comparison module of the software. This thesis also extended previously employed *Iterative Closest Point* algorithm, used for mesh alignment, by adding possibility to compute scaling parameter.

As this algorithm was design to work primarily within the *FIDENTIS Analyst*, bulk of our thesis focuses on analyzing human faces. Nonetheless, our algorithm was successfully used for analysis of different types of meshes, including, but not limited to the human bones and 3D scan of mice.

We examined approaches used for facial recognition and mesh reconstruction, to analyze currently employed techniques of alignment of multiple meshes, as well as creating the mean shapes of the set. The main focus of facial recognition dwells in verification of the face, where we compare two faces, and facial identification, where we compare one face against the database. Our approach aims to instead cross-analyze all entries in the set.

This results into increasing computational difficulty of the task, which we tried to resolve in this thesis. We provided evaluation of our algorithm and compared it with two different approaches of solving the same problem. This thesis also shows the implementation issues we had to deal with.

In the evaluation phase we were testing the speed and the precision of our algorithm. The main focus of our algorithm was to combine advantages of remaining two approaches it was tested against, by providing precision closer to pair-wise comparison, but at the same time reducing the time necessary for computation. We evaluated all three algorithms on three groups of meshes, all with the same individuals, but with different amount of vertices per group.

During the speed evaluation we looked at the time required for computation to finish for each approach and determined, why each of the algorithms took given time. We also showed number of iterations required for *Scaling Iterative Closest Point* algorithm to converge for each approach.

In precision evaluation, we used three techniques to evaluate each algorithm. Firstly, we let each approach to determine triad of most similar meshes to each mesh in the set. We then compared the results to results of pair-wise alignment, as we consider it the most precise of three. Secondly,

we computed distance of *Root Mean Square* metric for each algorithm and determined which of the algorithms is closer to pair-wise comparison. Lastly, we also used statistical approach to evaluation of the algorithms using *Mantel test*, showing similarities between matrices of results of each approach and their statistical significance.

The evaluation phase showed, that our algorithm scales better than pair-wise algorithm both, with increased density and number of iterations. The precision evaluation of our algorithm shows that, unlike approach of aligning all meshes to one, it didn't detected any triad of most similar meshes with more than one mismatch. The number of correctly detected triads in all three groups was also higher than faster alignment to one mesh. When evaluating distances to pair-wise results, our algorithm showed slight advantage over alignment to one mesh. *Mantel test* showed, that alignment to one mesh produces result matrix more similar to those of pair-wise computation, however the percentage differences of the matrices were minimal.

We also analyzed how number of the mean shapes created during computation and choice of the base for the mean shape template affected the precision of the algorithm. We analyzed differences between the mean shapes, concluding that while first mean shape is significantly different from the template base, the deviations between next iterations are much smaller. When using different base for the template of average shape, we discovered triads of the most similar faces found changed. Therefore, we concluded, that choice of the appropriate base for the mean shape has much more significance on the precision of the algorithm than number of mean shapes created.

The implemented algorithm was a base for three accepted conference entries. These were the Surface-based visualization on human face variation [19], Generation of variable human faces from 3D scan dataset [18], and Cloud-computing based system for analysis of 3D facial data [20].

The evaluation phase gave us good basis for future improvement. Most importantly, we would like to provide technique to automatically choose the most appropriate base of the template. Several of precision problem which arose during evaluation were due to vertex-to-vertex metric we used and as such we would like to extended this to vertex-to-triangle metric to increase the precision of our algorithm. Determining the exact bottlenecks of our algorithm we would also like to speed up creation of mean shape.

There are several areas of interest for the future, however some of them would lead to more specialized algorithms, focused mostly on human faces. Some of these are for example, using undersampling for mesh alignment,

either by one of techniques we mentioned, or by segmenting mesh and trying to only use significant parts of the surface.

Bibliography

[1] Face Recognition. [Online; accessed 6-December-2015]. URL: https://www.fbi.gov/about-us/cjis/fingerprints_biometrics/biometric-center-of-excellence/files/face-recognition.pdf.

[2] Hausdorff distance. [Online; accessed 8-December-2015]. URL: https://en.wikipedia.org/wiki/Hausdorff_distance.

[3] OBJ Specification. [Online; accessed 6-December-2015]. URL: http://www.martinreddy.net/gfx/3d/OBJ.spec.

[4] OpenDSA: k-D Trees. [Online; accessed 8-December-2015]. URL: http://algoviz.org/OpenDSA/Books/Everything/html/KDtree.html.

[5] PLY Specification. [Online; accessed 6-December-2015]. URL: http://paulbourke.net/dataformats/ply/.

[6] PointCloud Documentation - Interactive Iterative Closest Point. [Online; accessed 8-December-2015]. URL: http://pointclouds.org/documentation/tutorials/interactive_icp.php.

[7] STL Specification. [Online; accessed 6-December-2015]. URL: http://www.fabbers.com/tech/STL_Format.

[8] T. Ahonen, A. Hadid, and M. Pietikainen. Face description with local binary patterns: Application to face recognition. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 28(12):2037–2041, Dec 2006. doi:10.1109/TPAMI.2006.244.

[9] P. Belhumeur, J. Hespanha, and D. Kriegman. Eigenfaces vs. Fisherfaces: recognition using class specific linear projection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 19(7):711–720, Jul 1997. doi:10.1109/34.598228.

[10] R. Benjemaa and F. Schmitt. A solution for the registration of multiple 3D point sets using unit quaternions. In H. Burkhardt and B. Neumann, editors, *Computer Vision — ECCV'98*, volume 1407 of *Lecture Notes in Computer Science*, pages 34–50. Springer Berlin Heidelberg, 1998. URL: http://dx.doi.org/10.1007/BFb0054732, doi:10.1007/BFb0054732.

[11] K. Berthold and P. Horn. Closed-form solution of absolute orientation using unit quaternions. *J. Opt. Soc. Am. A*, 4(4):629–642, Apr 1987. URL: http://josaa.osa.org/abstract.cfm?URI=josaa-4-4-629, doi:10.1364/JOSAA.4.000629.

[12] P. Besl and N. D. McKay. A method for registration of 3-D shapes. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 14(2):239–256, Feb 1992. doi:10.1109/34.121791.

[13] D. Beymer, A. Shashua, and T. Poggio. Example based image analysis and synthesis. Technical report, Cambridge, MA, USA, 1993.

[14] V. Blanz and T. Vetter. A morphable model for the synthesis of 3D faces. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '99, pages 187–194, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co. URL: http://dx.doi.org/10.1145/311535.311556, doi:10.1145/311535.311556.

[15] V. Blanz and T. Vetter. Face recognition based on fitting a 3D morphable model. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 25(9):1063–1074, Sept 2003. doi:10.1109/TPAMI.2003.1227983.

[16] M. Botsch, L. Kobbelt, M. Pauly, P. Alliez, and B. Lévy. *Polygon mesh processing*. CRC press, 2010.

[17] A. Bronstein, M. Bronstein, and R. Kimmel. Three-dimensional face recognition. *International Journal of Computer Vision*, 64(1):5–30, 2005. URL: http://dx.doi.org/10.1007/s11263-005-1085-y, doi:10.1007/s11263-005-1085-y.

[18] I. Chalás, Z. Ferková, K. Furmanová, J. Sochor, and B. Kozlíková. Generation of variable human faces from 3D scan dataset. In *In Proceedings of the 7th International Conference on Games and Virtual Worlds for Serious Applications*, pages 38–45. IEEE Computer Society, 2015.

[19] I. Chalás, Z. Ferková, P. Urbanová, B. Kozlíková, Z. Kotulanová, and J. Sochor. Surface-based visualization on human face variation. In T. R. Ivan Viola, Katja Buhler, editor, *Poster Session Presented at the Eurographics Workshop on Visual Computing for Biology and Medicine*. Eurographics Association, 2014.

[20] I. Chalás, L. Kohút, Z. Ferková, and J. Sochor. Cloud-computing based system for analysis of 3D facial data. In *Abstract Book of the 7th European Academy of Forensic Science Conference*, 2015.

[21] C.-F. Chen, Y.-S. Tseng, and C.-Y. Chen. Combination of PCA and wavelet transforms for face recognition on 2.5D images.

[22] M. Daoudi, A. Srivastava, and R. Veltkamp. *3D Face Modeling, Analysis and Recognition*. Wiley Publishing, 1st edition, 2013.

[23] D. DeCarlo, D. Metaxas, and M. Stone. An anthropometric face model using variational techniques. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '98, pages 67–74, New York, NY, USA, 1998. ACM. URL: `http://doi.acm.org/10.1145/280814.280823`, `doi:10.1145/280814.280823`.

[24] S. Du, N. Zheng, L. Xiong, S. Ying, and J. Xue. Scaling iterative closest point algorithm for registration of m–D point sets. *Journal of Visual Communication and Image Representation*, 21(5–6):442 – 452, 2010. Special issue on Multi-camera Imaging, Coding and Innovative Display. URL: `http://www.sciencedirect.com/science/article/pii/S1047320310000258`, `doi:http://dx.doi.org/10.1016/j.jvcir.2010.02.005`.

[25] S. Du, N. Zheng, S. Ying, and J. Wei. ICP with bounded scale for registration of M-D point sets. In *Multimedia and Expo, 2007 IEEE International Conference on*, pages 1291–1294, July 2007. `doi:10.1109/ICME.2007.4284894`.

[26] G. Edwards, A. Lanitis, C. Taylor, and T. Cootes. Modelling the variability in face images. In *Automatic Face and Gesture Recognition, 1996., Proceedings of the Second International Conference on*, pages 328–333, Oct 1996. `doi:10.1109/AFGR.1996.557286`.

[27] M. Galvánek, K. Furmanová, I. Chalás, and J. Sochor. Automated facial landmark detection, comparison and visualization. In *Proceedings of the 31st Spring Conference on Computer Graphics*, pages 21–28, Bratislava, Slovakia, 2015. Comenius University.

[28] A. Golovinskiy and T. Funkhouser. Randomized cuts for 3D mesh analysis. *ACM Trans. Graph.*, 27(5):145:1–145:12, Dec. 2008. URL: `http://doi.acm.org/10.1145/1409060.1409098`, `doi:10.1145/1409060.1409098`.

[29] J. Gower. Generalized Procrustes Analysis. *Psychometrika*, 40(1):33–51, 1975. URL: http://dx.doi.org/10.1007/BF02291478, doi:10.1007/BF02291478.

[30] S. Gupta, M. Markey, and A. Bovik. Anthropometric 3D face recognition. *International Journal of Computer Vision*, 90(3):331–349, 2010. URL: http://dx.doi.org/10.1007/s11263-010-0360-8, doi:10.1007/s11263-010-0360-8.

[31] H. Hagedoorn. ATI Radeon HD 2900 XT 512MB review - Page 9 - A new beginning - Tessellation. [Online; accessed 8-December-2015]. URL: http://www.guru3d.com/articles-pages/ati-radeon-hd-2900-xt-512mb-review,9.html.

[32] G. B. Huang. Learning hierarchical representations for face verification with convolutional deep belief networks. In *Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, CVPR '12, pages 2518–2525, Washington, DC, USA, 2012. IEEE Computer Society. URL: http://dl.acm.org/citation.cfm?id=2354409.2354890.

[33] D. F. Huber and M. Hebert. Fully automatic registration of multiple 3D data sets. *Image and Vision Computing*, 21(7):637 – 650, 2003. Computer Vision beyond the visible spectrum. URL: http://www.sciencedirect.com/science/article/pii/S026288560300060X, doi:http://dx.doi.org/10.1016/S0262-8856(03)00060-X.

[34] D. Huttenlocher, G. Klanderman, and W. Rucklidge. Comparing images using the Hausdorff distance. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 15(9):850–863, Sep 1993. doi:10.1109/34.232073.

[35] M. Jones and T. Poggio. Multidimensional morphable models: A framework for representing and matching object classes. *International Journal of Computer Vision*, 29(2):107–131, 1998. URL: http://dx.doi.org/10.1023/A%3A1008074226832, doi:10.1023/A:1008074226832.

[36] V. Krajíček. Correspondence problem in geometrics morphometric tasks, 2015. [Online; accessed 6-December-2015]. URL: https://is.cuni.cz/webapps/zzp/download/140044009/?lang=en.

[37] J. Lee. 3D face recognition using range images. *Final Report, Multidimensional DSP, Spring*, 5, 2005.

[38] Y. Lei, Q. Li, X. Song, Z. Shi, and D. Chen. 3D face hierarchical recognition based on geometric and curvature features. In *Computer Network and Multimedia Technology, 2009. CNMT 2009. International Symposium on*, pages 1–4, Jan 2009. doi:10.1109/CNMT.2009.5374521.

[39] N. Mantel. The detection of disease clustering and a generalized regression approach. *Cancer research*, 27(2 Part 1):209–220, 1967.

[40] Michael Wild. Recent Development of the Iterative Closest Point (ICP) Algorithm. [Online; accessed 6-December-2015]. URL: http://students.asl.ethz.ch/upl_pdf/314-report.pdf.

[41] G. Pan, Z. Wu, and Y. Pan. Automatic 3D face verification from range data. In *Acoustics, Speech, and Signal Processing, 2003. Proceedings. (ICASSP '03). 2003 IEEE International Conference on*, volume 3, pages III–193–6 vol.3, April 2003. doi:10.1109/ICASSP.2003.1199140.

[42] H. Rowley, S. Baluja, and T. Kanade. Neural network-based face detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 20(1):23–38, Jan 1998. doi:10.1109/34.655647.

[43] S. Rusinkiewicz and M. Levoy. Efficient variants of the ICP algorithm. In *3-D Digital Imaging and Modeling, 2001. Proceedings. Third International Conference on*, pages 145–152, 2001. doi:10.1109/IM.2001.924423.

[44] A. Samani, J. Winkler, and M. Niranjan. Automatic face recognition using stereo images. In *Acoustics, Speech and Signal Processing, 2006. ICASSP 2006 Proceedings. 2006 IEEE International Conference on*, volume 5, pages V–V, May 2006. doi:10.1109/ICASSP.2006.1661425.

[45] M. Segundo, C. Queirolo, O. Bellon, and L. Silva. Automatic 3D facial segmentation and landmark detection. In *Image Analysis and Processing, 2007. ICIAP 2007. 14th International Conference on*, pages 431–436, Sept 2007. doi:10.1109/ICIAP.2007.4362816.

[46] A. Stoddart and A. Hilton. Registration of multiple point sets. In *Pattern Recognition, 1996., Proceedings of the 13th International Conference on*, volume 2, pages 40–44 vol.2, Aug 1996. doi:10.1109/ICPR.1996.546720.

[47] H. Tanaka, M. Ikeda, and H. Chiaki. Curvature-based face surface recognition using spherical correlation. principal directions for curved

object recognition. In *Automatic Face and Gesture Recognition, 1998. Proceedings. Third IEEE International Conference on*, pages 372–377, Apr 1998. doi:10.1109/AFGR.1998.670977.

[48] M. Turk and A. Pentland. Face recognition using eigenfaces. In *Computer Vision and Pattern Recognition, 1991. Proceedings CVPR '91., IEEE Computer Society Conference on*, pages 586–591, Jun 1991. doi:10.1109/CVPR.1991.139758.

[49] J. Williams and M. Bennamoun. Simultaneous registration of multiple corresponding point sets. *Computer Vision and Image Understanding*, 81(1):117 – 142, 2001. URL: http://www.sciencedirect.com/science/article/pii/S1077314200908841, doi:http://dx.doi.org/10.1006/cviu.2000.0884.

[50] W. Zhang, S. Shan, W. Gao, X. Chen, and H. Zhang. Local gabor binary pattern histogram sequence (LGBPHS): a novel non-statistical model for face representation and recognition. In *Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on*, volume 1, pages 786–791 Vol. 1, Oct 2005. doi:10.1109/ICCV.2005.147.

[51] Z. Zhang. Iterative point matching for registration of free-form curves and surfaces. *International Journal of Computer Vision*, 13(2):119–152, 1994. URL: http://dx.doi.org/10.1007/BF01427149, doi:10.1007/BF01427149.

Appendix A

Numerical results

This section provides some of the numerical results summed up in the main body of the thesis. Tables 6.1, 6.2 and 6.3 show the most similar triads found for Tables 5.9, 5.10 and 5.11 respectively. Table 6.4 then shows the most similar triads detected by each algorithm when using smaller number of iterations. Table 6.5 shows the most similar triads detected by each algorithm whe error rate parameter is set to larger value. Numerical results not presented in the text of the thesis can be found in thesis' repositary.

5k Group	**To Each**	**To One**	**To Average**
1	15, 3, 11	15, 3, 11	15, 3, 2
2	3, 5, 6	3, 15, 1	3, 1, 5
3	2, 4, 15	2, 15, 4	2, 15, 1
4	3, 5, 1	3, 15, 11	3, 11, 1
5	6, 13, 17	6, 13, 12	6, 13, 17
6	5, 13, 17	5, 12, 13	5, 12, 13
7	18, 20, 9	9, 20, 10	9, 18, 20
8	9, 18, 10	9, 18, 10	9, 18, 10
9	10, 18, 8	10, 8, 11	10, 8, 18
10	9, 11, 18	9, 11, 20	9, 11, 13
11	10, 20, 9	10, 9, 20	10, 13, 9
12	16, 17, 6	16, 17, 6	16, 17, 5
13	5, 10, 6	5, 10, 11	5, 10, 11
14	18, 10, 11	18, 10, 12	18, 10, 20
15	1, 3, 4	1, 3, 4	1, 3, 4
16	17, 12, 5	17, 12, 18	17, 12, 18
17	16, 5, 12	16, 12, 18	16, 12, 5
18	9, 10, 8	14, 8, 9	9, 8, 14
19	18, 8, 9	18, 8, 14	18, 8, 9
20	10, 18, 11	10, 11, 9	10, 18, 11

Table 6.1: The most similar triads of meshes for 5k group and default parameters of To Average algorithm.

10k Group	**To Each**	**To One**	**To Average**
1	15, 11, 3	15, 11, 3	15, 2, 3
2	3, 5, 6	3, 5, 4	3, 1, 5
3	2, 4, 15	2, 4, 15	2, 4, 15
4	3, 15, 2	3, 15, 11	3, 11, 15
5	6, 13, 17	6, 13, 12	6, 13, 17
6	5, 13, 17	5, 13, 17	5, 12, 13
7	18, 9, 20	9, 11, 10	9, 18, 10
8	9, 18, 10	9, 18, 10	9, 18, 10
9	10, 18, 8	10, 11, 8	10, 18, 8
10	9, 11, 18	9, 11, 20	9, 11, 13
11	10, 20, 9	10, 9, 20	10, 9, 20
12	16, 17, 6	16, 17, 6	16, 17, 6
13	5, 6, 10	5, 10, 11	5, 10, 11
14	18, 10, 11	18, 10, 20	18, 10, 11
15	1, 3, 4	1, 3, 4	1, 3, 4
16	17, 12, 5	17, 12, 18	17, 12, 5
17	16, 5, 6	16, 12, 18	16, 12, 5
18	9, 10, 20	14, 9, 8	9, 14, 10
19	18, 8, 9	18, 8, 14	18, 8, 9
20	10, 18, 11	10, 11, 9	10, 18, 11

Table 6.2: The most similar triads of meshes for 10k group and default parameters of To Average algorithm.

20k Group	**To Each**	**To One**	**To Average**
1	15, 11, 3	15, 11, 4	15, 11, 4
2	3, 5, 6	3, 5, 4	3, 5, 1
3	2, 4, 15	2, 4, 15	2, 4, 15
4	3, 2, 15	3, 15, 11	3, 11, 15
5	6, 13, 17	6, 13, 12	6, 13, 17
6	5, 13, 17	5, 12, 17	5, 12, 17
7	18, 9, 20	9, 10, 11	9, 18, 10
8	9, 18, 10	9, 18, 10	9, 18, 10
9	10, 18, 8	10, 11, 8	10, 18, 8
10	9, 11, 18	9, 11, 20	9, 11, 18
11	10, 20, 9	10, 9, 20	10, 9, 20
12	16, 6, 17	16, 17, 6	16, 17, 6
13	5, 6, 10	5, 10, 11	5, 10, 11
14	18, 10, 11	18, 10, 11	18, 10, 11
15	1, 3, 4	1, 3, 4	1, 3, 4
16	17, 12, 5	17, 12, 18	17, 12, 5
17	16, 5, 6	16, 12, 18	16, 5, 12
18	9, 10, 20	14, 9, 10	9, 10, 14
19	18, 8, 10	18, 8, 4	18, 8, 9
20	10, 18, 11	10, 11, 9	10, 18, 11

Table 6.3: The most similar triads of meshes for 20k group and default parameters of To Average algorithm.

20k Group	**To Each**	**To One**	**To Average**
1	15, 11, 3	15, 11, 4	15, 11, 4
2	3, 5, 6	3, 5, 4	3, 5, 6
3	2, 4, 15	2, 4, 15	2, 4, 15
4	3, 2, 15	3, 15, 11	3, 11, 10
5	6, 13, 17	6, 13, 12	6, 13, 17
6	5, 13, 17	5, 12, 17	5, 12, 17
7	18, 9, 20	9, 10, 11	9, 11, 10
8	9, 18, 10	9, 18, 10	9, 10, 18
9	10, 18, 8	10, 11, 8	10, 18, 8
10	9, 11, 18	9, 11, 20	9, 11, 13
11	10, 20, 9	10, 9, 20	10, 9, 20
12	16, 6, 17	16, 17, 6	16, 6, 17
13	5, 6, 10	5, 10, 11	5, 10, 11
14	18, 10, 11	18, 10, 11	18, 10, 11
15	1, 3, 4	1, 3, 4	1, 3, 4
16	17, 12, 5	17, 12, 18	17, 12, 5
17	16, 5, 6	16, 12, 18	16, 5, 12
18	9, 10, 20	14, 9, 10	9, 10, 14
19	18, 8, 10	18, 8, 4	18, 8, 9
20	10, 18, 11	10, 11, 9	10, 18, 11

Table 6.4: The most similar triads of meshes for 20k group, 20 iterations and 0.05 error rate.

20k Group	**To Each**	**To One**	**To Average**
1	15, 11, 3	15, 11, 2	15, 11, 3
2	3, 5, 6	3, 5, 4	3, 5, 6
3	2, 4, 15	2, 4, 15	2, 4, 15
4	3, 2, 5	3, 2, 15	3, 11, 15
5	6, 13, 17	6, 13, 12	6, 13, 17
6	5, 13, 17	5, 12, 17	5, 12, 17
7	18, 9, 20	9, 10, 11	9, 10, 8
8	9, 18, 10	18, 9, 10	9, 10, 18
9	10, 18, 8	10, 11, 8	10, 18, 8
10	9, 11, 18	9, 11, 20	9, 11, 13
11	10, 20, 9	10, 9, 20	10, 9, 20
12	16, 17, 6	16, 17, 6	16, 17, 6
13	5, 6, 10	5, 10, 11	5, 10, 11
14	10, 18, 11	18, 10, 20	18, 10, 11
15	1, 3, 4	1, 3, 4	1, 3, 4
16	17, 12, 5	17, 12, 18	17, 12, 18
17	16, 5, 6	16, 12, 18	16, 12, 5
18	10, 9, 8	14, 9, 8	9, 10, 14
19	18, 8, 20	18, 8, 14	18, 8, 14
20	10, 11, 18	10, 11, 9	10, 18, 11

Table 6.5: The most similar triads of meshes for 20k group, 40 iterations and 0.1 error rate.

Appendix B

Repository content

Thesis repository contains several files. Firstly, archive *Numerical results* contains computed numerical results used in evaluation. It is structured as follows:

- **AvgFace Comp** — this folder contains computed numerical results when changing the base of the template used for computation of new average face.
- **Mantel** — this folder contains results of Mantel tests.
- **RAW** — this folder contains numerical results of comparisons used.

Secondly, repository contains demo application which allows to run all three of used algorithms. It allows to change parameters of maximal number of iterations, number of average faces created, error rate and whether to use the scaling during alignment. For comparison, it allows to choose metric type and whether to use signed distance. Application also allows to export computed results, providing information about computation times of the algorithms, computed numerical results, and symmetric results matrix.

Finally, repository contains samples of five meshes which can be used to test the application. These are not the results, which were used for the evaluation phase, as data used in the thesis are considered personal data and can not be distributed freely.