Inferring Human Body Parts and Correlations from Images, Pointclouds and Meshes

David Haldimann

Semester Thesis
June 2018

Prof. Dr. Markus Gross

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

cgl
computer graphics laboratory

Abstract

This thesis addresses the development of a novel sample thesis. We analyze the requirements of a general template, as it can be used with the LaTeX text processing system. (And so on…) The abstract should not exceed half a page in size!

Zusammenfassung

Diese Arbeit beschäftigt sich mit der Entwicklung einer neuartigen Beispielausarbeitung. Wir untersuchen die Anforderungen, die sich für eine allgemeine Vorlage ergeben, die innerhalb der LaTeX-Textverarbeitungsumgebung verwendet werden kann. (Und so weiter und so fort…) Die Zusammenfassung sollte nicht länger als eine halbe Textseite sein!

Contents

List of Figures

List of Tables

1

Introduction

The price for breast enhancement surgery was estimated to be \$3718 in 2017 according to the American Society of Plastic Surgery[1]. Next to the financial aspects there are also risks connected to undergoing surgery and also not knowing exactly what the result will look like. Before commiting to this kind of operation, it should be possible to generate a preview of the outcome from a few images. This thesis aims to design a method that is able to predict a 3D model of the outcome by learning a mapping between paramteric models. Additionally, the idea is explored if it is possible to generate a parametric model from a character modelling software.

1.1 Parametric Model

A previous implementation by Biland [Bil17] was used to create parametric models. A parametric model can describe all data that went into the model with its parameters. For example, the physical appearance of a person can be roughly described by their height, skin tone and hair color, where these three are the parameters of this parametric model. This is of course only an approximation as the description of the person would increase with more parameters. It is also possible that one parameter influences multiple features. In the previous example, when the height of a person is raised, the length of the arms is also proportionally increased.

[1] https://www.plasticsurgery.org/cosmetic-procedures/breast-augmentation/cost

1.2 Mapping

A mapping between sets associates each element in the first set with one or more elements of the second set. An example for a simple mapping could be the numbers one to twenty-six as the first set and the letters of the alphabet as the second. In the case of two paramteric models, the goal is to find a mapping that describes the relationship between the parameters of the first and second model. This mapping can either be linear or non-linear. The difference between a linearity and a non-linearity can be described with a simple example. The time it takes to drive $10km$ in a car at $10\frac{km}{h}$ is $1h$. If the distance to drive is doubled to $20km$, so will the time it takes. That is because distance to drive and time it requires are in a linear relationship. On the other hand, given that the braking distance while travelling with $10\frac{km}{h}$ is $1m$, the braking distance while travelling with $20\frac{km}{h}$ is $4m$. This is due to the fact, that speed and braking distance are related in a quadratic, non-linear manner.

1.3 Applications

-Breast shape/look prediction -Medical applications

2

Related Work

3

Methods

In this chapter the methods needed to create a parametric model and a mapping between parametric models are introduced and explained. First off, the data gathering and preprocessing are outlined. Multiple variants of the parametric model are discussed. Different learning approaches for the mapping are reviewed. Lastly, a character editor is presented that is used to generate data for a parametric model.

3.1 Data Acquisition and Preprocessing

Optimally it would be necessary to have a large data set of images of women before and after breast enhancement surgery, where the patients pose topless. It is very unlikely though that such a database exists, due to the fact that having breast surgery is a very personal topic and people generally don't enjoy posing naked. Therefore the images used were downloaded from a website[1] that offered to simulate various plastic surgical procedures including breast enhancement. For each user a 3D model of their torse was displayed side by side with different enhancements varying in size. Each model was made up of a sequence of 24 images displaying the torse from different angles. This dataset fit the requirements nicely as images are available for *before* and *after*, except the *after* is generated and based on their model. Additionally, each after image sequence had a short label, usually describing how much silicon was added, that was also saved for further evaluation. In total 2'937 examples were retrieved and preprocessed. This dataset included images from 748 subjects of which each one was comprised of one *before* and at least one *after* image sequence.

[1] https://my.crisalix.com/

In a next step these image sequences needed to be transformed into point clouds. This was done using a general-purpose Structure-from-Motion (SfM) [SF16] and Multi-View Stereo (MVS) [SZPF16] pipeline called COLMAP. This generated point clouds spanning from 5’000 to 15’000 points. Some of the images needed to be discarded, due to the fact that SfM created a point cloud with less than 1’000 points or the point clouds had holes, such that certain areas had no points and were not defined at all. The remaining point clouds were cleaned using a C++ implementation by Biland [Bil17] that removed white points around the point clouds.

The mapping required to have one set of point clouds of *before* examples and the corresponding[2] *after* examples. Therefore the data was split into sets of *before* and after point clouds. Additionally, to create a better mapping, only the *after* examples that were labelled "350" were included. This resulted in 57 examples in the *before* and 57 in the *after* set. All of these point clouds were further processed in a MATLAB implementation by Biland [Bil17] to generate mesh files.

3.1.1 Alignment

Eventhough the meshes were aligned in the MATLAB implementation by Biland [Bil17] in a general fashion, each corresponding *before* and *after* should be pairwise aligned to avoid that the mapping also learns rotations. This was achieved by using an implementation of Horn’s method [Hor87]. Given two sets of vertices, Horn’s method computes the translation, rotation and possibly scale change from one set to the other. As it is expected that for the same subject, only points defining the breasts should vary from before to after, only a subset of vertices should be used. The points used in the alignment can be seen in figure 3.1.

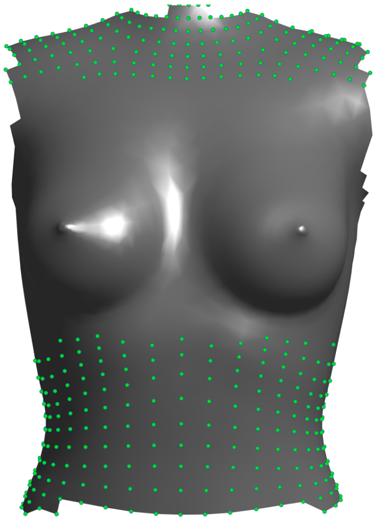


Figure 3.1: *The mesh is depicted as the grey surface. The points drawn in green are used for the alignment.*

[2] Corresponding meaning, based on the same subject.

3.2 Parametric Model from Meshes

In this section it is described how a parametric model is obtained using principle component analysis (PCA). Given n meshes $m_i \in \mathbb{R}^{k \times 3}$, where k describes the amound of vertices m has, each mesh needs to be transformed to be of shape $\mathbb{R}^{1 \times 3k}$. Then, all transformed meshes are stacked into a matrix $M \in \mathbb{R}^{n \times 3k}$. As the differences over each column isn't significant, the mean $\bar{m}$ of the matrix M is subtracted from each row of M.

$$\mathbf{A} := \begin{bmatrix} m'_1 - \bar{m} \\ m'_2 - \bar{m} \\ \vdots \\ m'_n - \bar{m} \end{bmatrix} \in \mathbb{R}^{n \times 3k} \tag{3.1}$$

Next, PCA is run with matrix A as the input. PCA is able to reduce the dimensionality of the data while retaining most of the information from the initial data set. This is achieved by finding orthogonal basis vectors, where the first basis vector is responsible for the largest variance in the data. The second basis vector needs to be orthogonal to the first and is responsible for the second largest variance of the data. This holds for each following basis vector. These basis vectors in PCA are also known as principle components.

The output of the PCA function is:

$$\mathbf{coeff} := \begin{bmatrix} c_1 & c_2 & \cdots & c_{n-1} \end{bmatrix} \in \mathbb{R}^{3k \times n-1} \tag{3.2}$$

$$\mathbf{score} := \begin{bmatrix} s_1 & s_2 & \cdots & s_{n-1} \end{bmatrix} \in \mathbb{R}^{n \times n-1} \tag{3.3}$$

where coefficient c_i is the i-th principle component and score s_i is the i-th parameter vector corresponding to the i-th input. Therefore the input data can be reproduced by computing $\mathbf{A} = \mathbf{score} \times \mathbf{coeff^T}$. Instead of using all $n-1$ coefficients it is possible to only use the first $\mathbf{q}$ coefficients resulting in an approximation of the space. The parameters $\mathbf{p_{q,new}}$ for a new input mesh m_{new} can be easily computed for $\mathbf{q}$ coefficients by first reshaping the mesh to be of the form $\mathbb{R}^{3k \times 1}$ and subtracting the mean $\bar{m}$. This is done the same way as above, adding an additional step to transpose. Then the pseudoinverse of $\mathbf{coeff_{(q)}}$ is multiplied from the left to compute the corresponding parameters $\mathbf{p_{q,new}}$:

$$\mathbf{p_{q,new}} = \mathbf{coeff^{+}_{(q)}} \cdot (m'_{new} - \bar{m})^T \text{ where } \mathbf{coeff_{(q)}} := \begin{bmatrix} c_1 & c_2 & \cdots & c_q \end{bmatrix} \in \mathbb{R}^{3k \times q} \tag{3.4}$$

In this example the parametric model is based on the vertices of the mesh. In the next subsections, two other variants are explored and explained.

3.2.1 Face Deformations

Instead of defining a mesh by its vertices, it is possible to describe it by the deformations of the faces. This is done by defining one source mesh and computing how each face is deformed. One way to quantify a deformation of a face was described by Sumner [SP04], where the idea was to transfer triangle deformations between similar meshes. First, a new vertex needs to be computed such that an affine transformation can be determined. The fourth vertex is defined as follows:

$$\mathbf{v_4} = \mathbf{v_1} + (\mathbf{v_2} - \mathbf{v_1}) \times (\mathbf{v_3} - \mathbf{v_1}) / \sqrt{|(\mathbf{v_2} - \mathbf{v_1}) \times (\mathbf{v_3} - \mathbf{v_1})|} \tag{3.5}$$

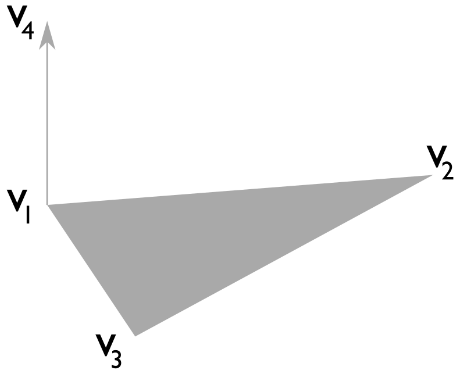


Figure 3.2: A vertex is added to the triangle that lies in the direction of the normal.

See figure 3.2 for a visualization of adding a vertex on the normal. Given both faces with one additional vertex, the following equations can be posed:

$$\mathbf{Q}\mathbf{v_i} + \mathbf{d} = \tilde{\mathbf{v}}_\mathbf{i}, \mathbf{i} \in \mathbf{1} \ldots \mathbf{4} \tag{3.6}$$

where $\mathbf{Q} \in \mathbb{R}^{3\times 3}$ and translation vector $\mathbf{d}$ describe the affine transformation. By subtracting the first equation from the following three equations and rewriting the resulting system in matrix form, the problem can be defined as $\mathbf{Q}\mathbf{V} = \tilde{\mathbf{V}}$ where

$$\mathbf{V} = \begin{bmatrix} \mathbf{v_2} - \mathbf{v_1} & \mathbf{v_2} - \mathbf{v_1} & \mathbf{v_2} - \mathbf{v_1} \end{bmatrix} \tag{3.7}$$

$$\tilde{\mathbf{V}} = \begin{bmatrix} \mathbf{v_2} - \mathbf{v_1} & \mathbf{v_2} - \mathbf{v_1} & \mathbf{v_2} - \mathbf{v_1} \end{bmatrix} \tag{3.8}$$

The closed form expression for $\mathbf{Q}$ is defined as

$$\mathbf{Q} = \tilde{\mathbf{V}}\mathbf{V}^{-1} \tag{3.9}$$

To fully describe a mesh, this $\mathbf{Q}$ matrix needs to be computed for each face of the mesh. Finally, the mesh is represented by $\mathbf{D} \in \mathbb{R}^{3\times 3h}$ where $\mathbf{h}$ is the number of faces the mesh has. To be able to run PCA, each mesh needs to be reshaped to be of form $\mathbf{D}' \in \mathbb{R}^{1\times 9h}$ and the rest of the process is the same as above in section 3.2.

3.2.2 Point Normals

This method is very much similar to one described in section 3.2. In addition to the vertices of the mesh, one vertex per face is computed as described in equation 3.5 and added to the list of vertices of the mesh. Therefore the mesh matrix will be of form $\mathbb{R}^{k+h \times 3}$ where $\mathbf{k}$ is the number of original vertices and $\mathbf{h}$ is the number of faces of the mesh.

3.3 Mapping

The following segment describes how a mapping can be computed in a linear or a non-linear way. The data available is the parameters returned from the both parametric models for the before and after examples. The matrices of the data are

$$\mathbf{P_{before}} = \begin{bmatrix} \mathbf{p_{b,1}} \\ \mathbf{p_{b,2}} \\ \vdots \\ \mathbf{p_{b,n}} \end{bmatrix} \in \mathbb{R}^{n \times n-1}, \mathbf{P_{after}} = \begin{bmatrix} \mathbf{p_{a,1}} \\ \mathbf{p_{a,2}} \\ \vdots \\ \mathbf{p_{a,n}} \end{bmatrix} \in \mathbb{R}^{n \times n-1} \tag{3.10}$$

where each parameter pair $(\mathbf{p_{b,i}}, \mathbf{p_{a,i}})\ \forall i$ is related, as the after was generated from the before.

3.3.1 Linear Method

The first method used, was the linear system solver by MATLAB (also known as backslash solver) that solved the equation

$$\mathbf{P_{before}} \cdot \mathbf{M_{linear}} = \mathbf{P_{after}} \text{where} M_{linear} \in \mathbb{R}^{n-1 \times n-1}. \tag{3.11}$$

It is clear that an explicit linear transformation could be found for each parameter pair, but the goal is to have one mapping that works for each parameter pair. This is also known as linear regression. One basic example in linear regression is the task of fitting a line to a point cloud. As all the points of the point cloud do not lie on one line, it isn't possible to define a straight line. Therefore a line is sought after, such that all points are as close as possible on average. This is depicted in figure 3.3.

3.3.2 Non-Linear Variants

This section deals with non-linear methods to solve the regression problem described above. All the methods used are part of the scikit-learn library [PVG+11]. It is common in a machine learning setting to split data into two sets. One set is used to train the model and the other is used to evaluate how well the method does on data, that was not part of the first set. Both data sets should be drawn from the same distribution. This is necessary because the model shouldn't be penalized for performing badly on problems it didn't encounter in the training phase. During

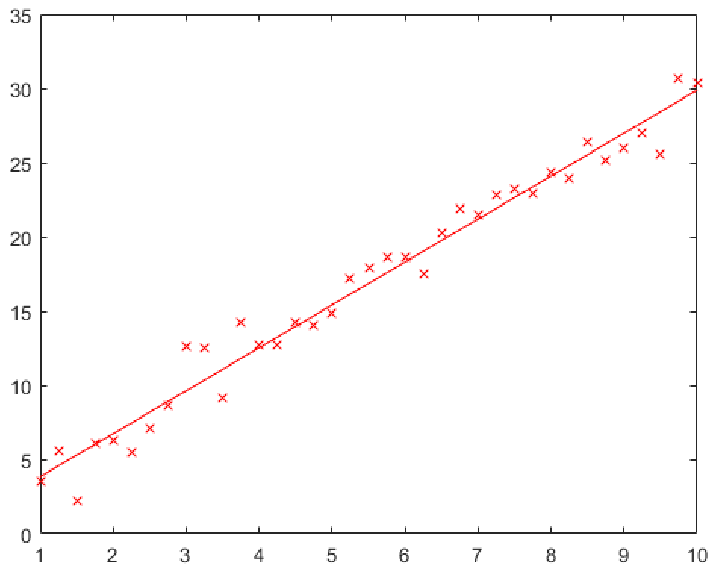


Figure 3.3: *Fitting a line to a point cloud using linear least-squares.*

the training phase it is standard to run grid search. Grid search varies the parameters of the model to find the best set of parameters for a specific task. It needs to be considered that grid search can lead to overfitting, this occurs when the error on the training set is reduced at the cost of increasing the test error. This can be avoided by adding regularization terms or by running grid search along with cross-validation.

Decision Tree

One category of machine learning approaches is based on so called decision trees (DT). A decision tree uses a tree-like approach, where nodes of the tree are defined to split the data set into subsets corresponding to groups. As an example, the goal is to have a model that can predict the weight of a person, based on height, daily activity and daily calorie intake as features. Using a decision tree, the value of height could split the data set, as taller people tend to be heavier than small people. This is done over all features until the complete data set is divided into smaller subsets. Leafs make up the end of a decision tree. Each leaf has a certain value connected to it, in the case mentioned before it would be a certain weight range or value. When the weight of a new person would need to be predicted, one would follow the tree from top to bottom until a leaf was met. The corresponding weight value of the leaf would be equal to the prediction the model returns. An example of a decision tree can be seen in figure 3.4.

Random Forest

The random forest (RF) regressor is also a tree-like approach similar to the decision tree. The main difference is that the decision tree considers all features when splitting and thus can be prone to overfitting. Random forest chooses features at random and builds decision trees based

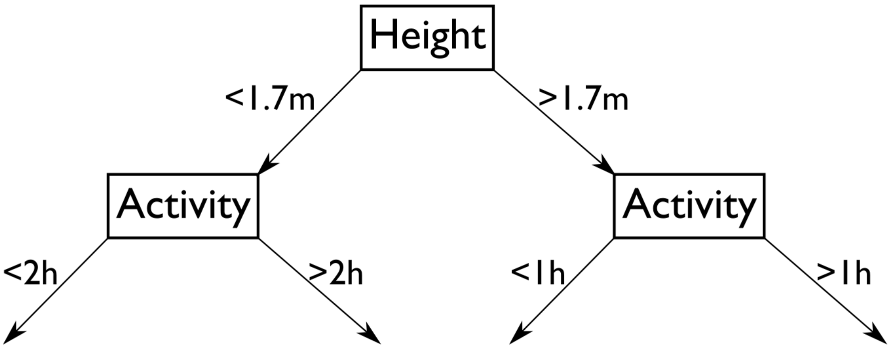


Figure 3.4: *Example of a decision tree following example of section 3.3.2*

on those features. This is done for multiple trees until all subtrees are combined.

Multilayer Perceptron

The multilayer perceptron (MLP) is a type of neural network. It uses a feedforward approach to propagate information through the network. MLP have at least three layers where the first layer is the input layer and the last layer is the output layer. Each layer is made up of neurons, that receive information from the previous layer, apply some operation followed by an activation function and pass the result on to the next layer. These activations can either be linear or non-linear. It doesn't make sense to have multiple intermediate layers with linear activation functions, as each linear combination of linear functions, can be described one other linear function. Each layer is made up of multiple neurons, each with their own weights and biases, and an activation function. All weights and biases are initialized at random. In the training phase a technique known as backpropagation is applied. Given an example, input and outcome, the input data is passed through all the layers. The resulting output is compared to the actual outcome. Given the difference between the two values, weights and biases are modified to reduce the difference, starting from the output layer, moving back through the hidden layers up to the input layer. A basic scheme of a MLP is shown in figure 3.5.

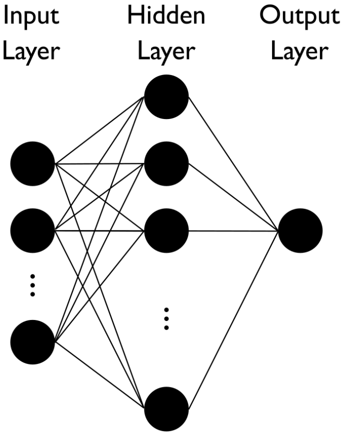
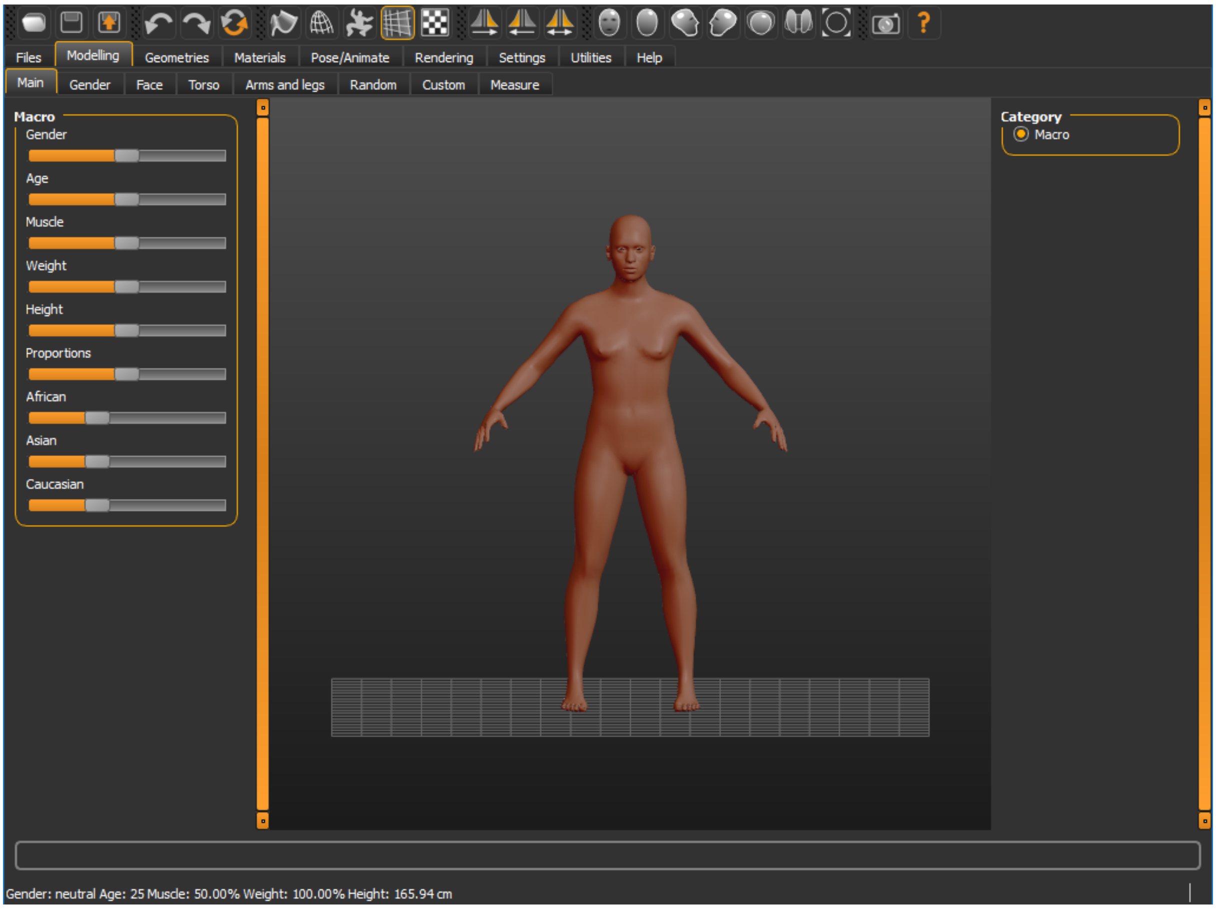


Figure 3.5: *Scheme of a multilayer perceptron for regression.*

3.4 Parametric Model from Editor

In all of the previous sections the parametric model was based off of real world data. In this section the same procedure is done as in section 3.2, except that the meshes used for PCA were generated by software called MakeHuman [3]. MakeHuman is a free and open source character modelling tool. It starts out with a gender neutral base mesh that can be modelled by varying sliders. These sliders apply linear interpolations to the base mesh between the extreme positions. There are over 40 sliders in MakeHuman and the sliders are categorized by what areas of the body they modify. The graphical user interface of MakeHuman can be seen in figure 3.6. Using the scripting feature in MakeHuman multiple meshes could be generated, exported and later imported into MATLAB.



Gender: neutral Age: 25 Muscle: 50.00% Weight: 100.00% Height: 165.94 cm

Figure 3.6: *Graphical user interface of MakeHuman.*

3.5 Fitting parametric models to point clouds

Given a parametric model, it is possible to create a large diversity of possible outputs by varying the parameters. But finding the specific parameters for a new point cloud is a non-linear optimization problem. Therefore a C++ implementation by Biland [Bil17], using an open source

[3] http://www.makehumancommunity.org/

C++ library for optimization problems called the Ceres solver [AMO], was slightly modified and used. This implementation allows to fit a parametric model to a point cloud given a few keypoints of the point cloud. In a first step the keypoints are used to find the parameters of translation, rotation and scaling. Afterwards a select number of points are chosen as correspondences between the model and the point cloud. Iteratively this implementation tries to minimize the distance between all the correspondences by varying the parameters of the model. After a certain number of iterative steps has been reached, a certain threshold in accuracy is passed or the change in parameters is vanishingly small, the resulting mesh and parameters are returned.

4

Results

4.1 Error Metrics

4.1.1 Distance Point

4.1.2 Face orientations

4.2 Evaluation

4.2.1 Mesh Point Error

4.2.2 Fit to Pointcloud

4.3 PCA point/normals/deform

4.4 Learning mapping

4.5 Data MH vs NRICP

4.6 Data Real vs NRICP

5

Conclusion and Outlook

Bibliography

[AMO] Sameer Agarwal, Keir Mierle, and Others. Ceres solver. `http://ceres-solver.org`.

[Bil17] S. Biland. Anatomically based body part modelling and fitting, 2017.

[Hor87] Berthold KP Horn. Closed-form solution of absolute orientation using unit quaternions. *JOSA A*, 4(4):629–642, 1987.

[PVG$^+$11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[SF16] Johannes Lutz Schönberger and Jan-Michael Frahm. Structure-from-motion revisited. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[SP04] Robert W Sumner and Jovan Popović. Deformation transfer for triangle meshes. In *ACM Transactions on Graphics (TOG)*, volume 23, pages 399–405. ACM, 2004.

[SZPF16] Johannes Lutz Schönberger, Enliang Zheng, Marc Pollefeys, and Jan-Michael Frahm. Pixelwise view selection for unstructured multi-view stereo. In *European Conference on Computer Vision (ECCV)*, 2016.