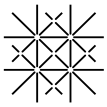


University of Basel

Efficient 3D Morphable Model Face Fitting using Depth Sensing Technologies

Master's Thesis

Natural Science Faculty of the University of Basel
Department of Mathematics and Computer Science
Graphics and Vision Research Group
https://gravis.dmi.unibas.ch

Examiner: Prof. Dr. Thomas Vetter
Supervisor: Dr. Marcel Lüthi

Giorgi Grigalashvili
g.grigalashvili@unibas.ch
12-059-754

13.11.2019

This page intentionally left blank.

Acknowledgments

First, I would like to thank my supervisor Dr. Marcel Lüthi for his continuous support and guidance over the course of this thesis. Further, I would like to thank Professor Thomas Vetter for giving me the opportunity to work in the Graphics and Vision Research Group. I also thank Dr. Andreas-Morel Foster for his help and valuable input, as well as, PhD students Patrick Kahr, Dana Rahbani and Dennis Madsen for their involvement in this project. Last but not least, I would like to thank my friends (especially those who proofread my work) and family for their support.

This page intentionally left blank.

Abstract

Human face analysis is an important branch of Computer Vision. One of the interesting topics in this area of research is the analysis of a human face based on a single image. The complexity of this problem is rather high due to restricted information that is available in a single image. Especially challenging is a task of the 3D shape reconstruction of the face based on a single image. The reconstruction process has little information to reliably recover the shape information (depth), which is the third dimension that is completely missing from an image. Using a single image does not always produce an accurate 3D reconstruction, especially in settings where lighting conditions cannot be controlled. This thesis aims to build an efficient and stable system, which adds, in addition to a photo of a face, also a 3D depth information to the reconstruction process. By incorporating a 3D depth information into this process, we aim to improve the overall quality of the final shape reconstruction, as well as, the visual appearance of the reconstructed face. We build the complete *fitting* pipeline that consists of separate modules dealing with data acquisition, shape reconstruction, and analysis of color and illumination. We show that including the complete 3D depth information into the framework in the form of a point cloud indeed improves the final reconstruction quality. To obtain a 3D depth information alongside the other input requirements, we use an affordable consumer depth camera technology offered by Intel® RealSense™ platform. The pipeline is flexible enough that it can be seamlessly integrated into the existing demo-framework (the face-fitting web service[1]) for various use-cases. The applications of this work include but not limited to photo-realistic face manipulation[2], 3D face modelling and analysis.

[1] Scalismo Face Morpher — https://face-morpher.scalismo.org
[2] Photo-realistic Face Manipulation — https://gravis.dmi.unibas.ch/PMM/demo/face-manipulation/

This page intentionally left blank.

Table of Contents

1

Introduction

Depth camera technologies have gotten a lot of attention lately. The key reason behind its substantial popularity growth and technological advancements is the fact that a lot of computer vision applications rely on its 3D sensing capabilities. As the demand for Machine Learning (ML) systems grow, we observe a rapid growth of the technologies using 3D sensing systems across different technology areas, be it a consumer electronics, self-driving cars or drones. The one such market which has gotten noticeably better and cheaper lately is the consumer depth camera market. Embedded into the latest generation of smartphones, depth camera modules found its way to scale of consumers. Standalone depth cameras also got better, cheaper and smaller while maintaining its qualitative advantages. While the former are designed to enhance overall image quality, stability and introduce industry's first dedicated sensor based Augmented Reality (AR) features inside the mobile operating system, the latter are designed to deal with a wide range of computationally heavy and accuracy focused applications. Such applications include object detection, motion tracking, face scanning, object reconstruction, etc.

In this thesis, we focus on the 3D reconstruction of the human face. A dense 3D face reconstruction from a single 2D image is a long-standing and widely researched topic in computer vision. Approaches include, model based probabilistic approaches[1–3] or nowadays the more popular convolution neural networks (CNN)[4, 5]. The aim of this thesis is to incorporate consumer depth camera capabilities into the existing model based 3D face reconstruction pipeline hereafter referred to as a face fitting pipeline using 3D Morphable Model (3DMM)[1, 6]. We also attempt to replicate and potentially improve the result of the original 2D face fitting pipeline [2] as well as augmented depth fitting [7] pipeline, which proved to improve the original approach. The original 2D fitting approach was performing the reconstruction based solely on a single image, while its improved version was utilizing an additional depth image information. Our main goal is to design and implement a stable client-server based system which robustly and efficiently takes care of all the sensitive details of both the image capturing procedure and later face fitting pipeline. In addition to that, we will investigate the possibility to speed up the fitting process so that the final pipeline could be used for live demos and could seamlessly be integrated into the existing

Scalismo Face Morpher web service (Figure 1.1).

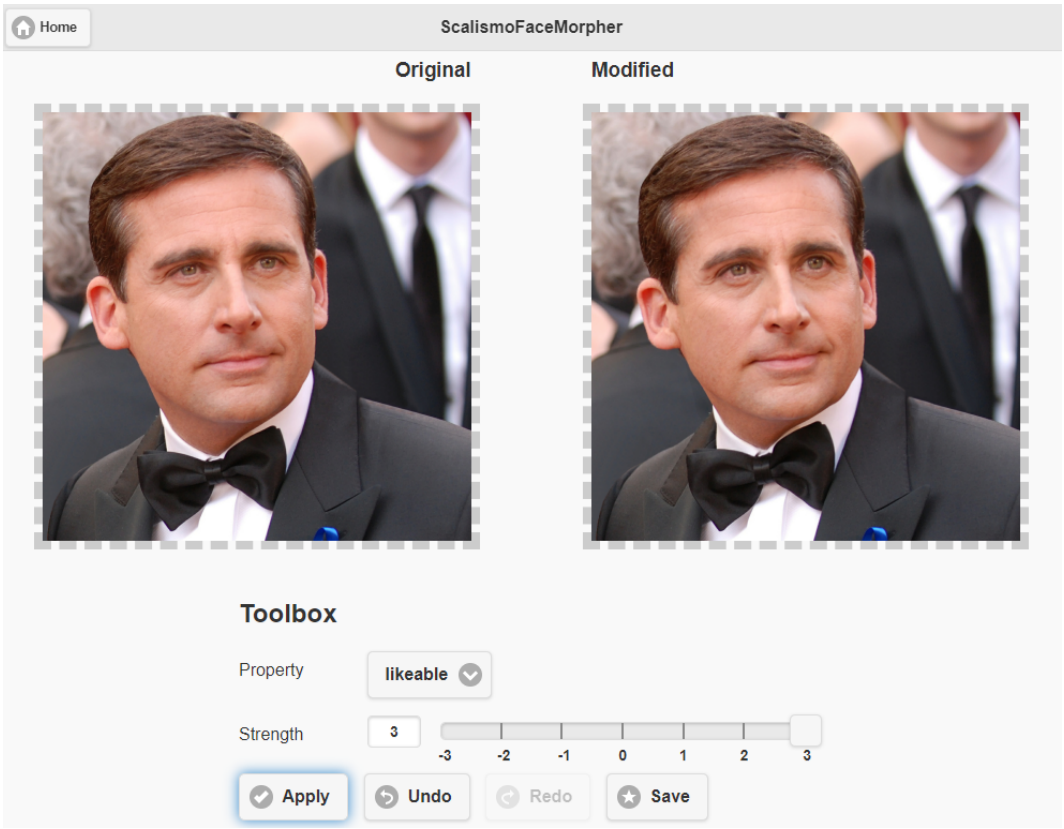


Figure 1.1: Scalismo Face Morpher website — https://face-morpher.scalismo.org.

1.1 Motivation & Previous Work

Reconstructing a 3D face from a single 2D image is a challenging task. The major issue the fitting process has to overcome is to efficiently recover the 3D shape information and overall appearance of the face in the image. The appearance of the face in the 3DMM environment is being controlled by a set of model parameters that control faces pose, shape, expression, color, and illumination. Face fitting, therefore, is a pipeline that takes a 2D color image with a human face in it (hereafter referred to as a target image) as an input and attempts to find the best possible set of parameters mentioned above by using Analysis-by-Synthesis paradigm[3, 8]. The key idea of the Analysis-by-Synthesis is to generate a synthetic face image which is close to an observed target image. By doing so, the fitting process fits 3DMM face model to the face appearing in the image such that, the fitted model instance (hereafter referred to as the best fit or simply fit) represents the target face as accurately as possible. Thus, the result of the face fitting pipeline is a synthetic face matching to the target face in the image and replicating all the important characteristics of it.

The complexity of inferring a face appearance from a single 2D image using the Analysis-by-Synthesis approach was investigated by Schönborn et al. [2, 3, 8]. Considering the

fact that the project was solely relying on a single 2D image that only holds information about the color intensities, quite good results were achieved. However, since the amount of information that can be utilized from a 2D image is limited, the authors proposed an idea of improving the standard 2D face fitting pipeline by combining the RGB data with the depth information obtained from a dedicated depth camera. Betschard et al. [7] investigated this idea by using Intel® RealSense™ F200 depth camera and demonstrated that it is possible to improve the reconstruction quality of the initial method by up to 35%. The authors of the latter research also proposed an idea of further improving the pipeline by relying on an additional point cloud information, which we thoroughly investigate in this thesis, and at the same time attempt to improve the efficiency and robustness of the fitting pipeline as a whole.

The main difference between a 3D depth camera and a normal 2D image capturing camera is that a depth camera instead of taking just a 2D RGB-color image also has an ability to produce a point cloud of the scene. A point cloud is a set of data points produced by the camera based on the combination of information obtained from the depth sensors and cameras' intrinsic parameters. Each point is located in the 3D coordinate system representing a direct mapping of pixels in the color image. This means that the number of points in a point cloud object directly correlates to the number of pixels in the corresponding color image. Therefore, each point in a point cloud represents an (x, y, z) tuple (coordinates) with the coordinate $(0, 0, 0)$ referring to the center of the physical camera sensor[9], the z axes in this tuple is called depth (or z-Buffer). It corresponds to a distance, measured from the camera to each corresponding pixel in the 2D color image. It is clear that having (x, y, z) coordinates will represent the pixel location in 3D coordinate space more accurately than having just a z-Buffer which was used by [7]; and obviously, this would give us significantly more information (especially, regarding the shape information of the face), than standard fitting can extract from a 2D image. Therefore, the performance, efficiency, and quality of majority applications, that only make use of the information extracted from a single 2D image could be potentially improved by employing a 3D depth information obtained from the depth camera.

We will be leveraging the latest generation Intel® RealSense™ D400 series depth camera shown in Figure 1.2 to obtain all necessary data for the fitting pipeline - including an RGB color image, a mesh constructed from point cloud, and a set of facial landmarks. The camera retails under 200CHF and it is one of the best affordable 3D depth cameras currently available on the market. Designed specificity for 3D scanning, portable D415 offers a good accuracy (Z-Accuracy $\leq$2% also known as the absolute error)[3] and dedicated powerful computer vision processor for on-the-fly calculations.

[3] The accuracy depends on the camera calibration procedure.

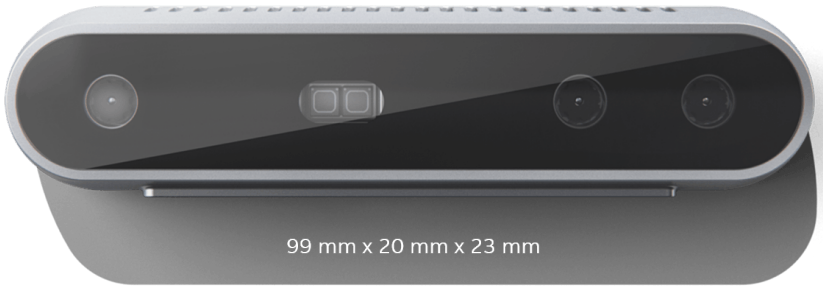


Figure 1.2: Intel® RealSense™ D415 camera equipped with an array of sensors (left to right) depth sensor 1, infrared (IR) projector, depth sensor 2, color sensor. Source: Intel

1.2 Contributions

In this work, we have made the following contributions. 1) We have implemented a new fitting pipeline, which in addition to color image also integrates a 3D depth information into the fitting process. We propose a new modular approach for solving this problem which utilizes point cloud information that is being captured from the camera. 2) We have implemented a client-server based modular system, which splits camera configuration, data acquisition, shape fitting, and color fitting procedures from each other. Thanks to its modularity, the system introduces additional flexibility and ease of use for any potential future applications. 3) We have successfully integrated our system into already existing Scaismo Face Morpher service (Figure 1.1), which provides the possibility to use our modules for demo purposes. 4) We evaluated our proposed implementation based on a dataset that consists of a series of sample shots that we have made using our data capturing procedure, and present results at the end of the thesis.

1.3 Thesis Structure

We structure this thesis as follows. The details of all the above mentioned procedures combined with background material necessary to follow the flow of events and concepts we employ throughout this thesis will be elaborated in Chapter 2. We introduce the model that we will be using for our application, as well as, describe the idea and implementation of a novel fitting pipeline. We also give a quick overview of depth camera technologies and how they work. Including how pixel-to-point and point-to-pixel de-projection is performed based on a pinhole camera model. We then proceed with describing the methods (Chapter 3) we used to implement a reliable system. Chapter starts with describing our module based system, including the description of each of its key modules and components. We first introduce our camera of choice, its calibration procedure, and parameter configuration together with a software development kit (SDK) that allows us to seamlessly communicate with the camera and utilize its depth sensing capabilities. We further intro-

duce a client-server based architecture that successfully delivers the data from the camera to the Scala client. The data client receives from the server is then being used to perform our proposed shape fitting pipeline and later slightly modified color fitting pipeline. Chapter 4 contains the results and the evaluation of our method and its performance comparison to a novel fitting method. We also provide intermediate comparison results between the shape and color fitting modules to ensure that the final output is within the limits of our desired result and it actually improves previous achievements. An example result of our proposed method can be seen in Figure 1.3.

Figure 1.3: Example fitting result with target image (left) and synthetic fit produced by our method (right).

In Chapter 5 we summarize our work and provide conclusions, followed by Chapter 6 where we propose a couple of potential improvements that can be done to further improve the stability and performance of our system.

2
Background

In this chapter, the foundations of the thesis are elaborated. We formally define methods, concepts and tools used. Starting with the description of the 3D Morphable Model (3DMM) [1, 6] that we utilize, followed by an explanation of a novel fully probabilistic method to interpret a single face image with the 3DMM[2]. As well as, the important characteristics of the depth cameras.

2.1 3D Morphable Model

The 3DMM is a fully parametric generative face model constructed from 200 high quality face scans. The model contains probabilistic PCA (PPCA) models for shape, color and expression [10, 11]. In a traditional PCA-based settings the shape S, color C and facial expression E models are constructed through the parameter set $\theta = \{\theta_S, \theta_C, \theta_E\}$ as follows:

$$\begin{aligned} S(\theta) &= \mu_S + U_S D_S \theta_S + \mu_E + U_E D_E \theta_E \\ C(\theta) &= \mu_C + U_C D_C \end{aligned} \tag{2.1}$$

Where μ denotes the mean of the corresponding model, U is a matrix that contains the principal components, and D denotes a diagonal matrix consisting of the variances along the principal directions. The expression model here, is modeled as a deformation of the neutral (mean) face shape. In a probabilistic setting the shape and color models are transformed into a distribution of shape $P(S \mid \theta)$ and color $P(C \mid \theta)$ components with the additive Gaussian noise[11, 12]:

$$\begin{aligned} P(S \mid \theta) &= \mathcal{N}(S \mid \mu_S + U_S D_S \theta_S + \mu_E + U_E D_E \theta_E, \sigma_S^2 I) \\ P(C \mid \theta) &= \mathcal{N}(C \mid \mu_C + U_C D_C, \sigma_C^2 I) \end{aligned} \tag{2.2}$$

The parameter set θ follow a standard normal distribution in latent vector space[11] which is also defined outside the linear span of 200 scans:

$$P(\theta) = \mathcal{N}(\theta \mid 0, I) \tag{2.3}$$

To generate realistic 3DMM instances (Figure 2.1) and render synthetic face images $\mathcal{I}$, parameter set θ contains an additional camera θ_P and illumination θ_L parameters that are modeled separately from the other three.

Sample Mean Sample with expression

Figure 2.1: 3DMM face model instances.

This allows us to synthesize the real world appearance of the face. By tuning all the mentioned parameters, we can approximate the appearance of the synthetic face to the target face. Besides parameter space, the model is also able to take care of the image rendering process $\Re$ which renders an image $\mathcal{I}(\theta)$ based on given parameters through

$$\mathcal{I}(\theta) = \Re(\mathcal{M}(\theta_S, \theta_C, \theta_E); \theta_P, \theta_L). \tag{2.4}$$

The 3DMM model also has a set of modified model versions that are either restricted to a certain region or are down-scaled, low-resolution versions[4] of the original model. The model restricted to a certain region is important for domain specific applications that only utilize specific parts of the model. For example, we use the model (Figure 2.2) from [2] which only covers the face region and discards other parts like ears, neck, and part of the forehead.

Figure 2.2: Restricted face model displayed over the full model(dark)

Beneficially to our application, this restriction also ignores parts of the model that are too

[4] Down-scaled versions usually have a lesser number of points and triangles hence are less flexible, but they perform the mesh operations faster

complex, noisy or low quality, for example ears, that could potentially cause some problems during the fitting.

2.2 Fitting Pipeline

Having the flexible Parametric Appearance Model (PAM) as a basis of the fitting framework allows us to model a variety of real world scenes with challenging illumination conditions. In this section, we explain how standard fitting pipeline and its augmented variant utilizes Markov Chain Monte Carlo sampling to perform approximate inference.

2.2.1 Markov Chain Monte Carlo (MCMC)

To generate parametric face instances, the Analysis-by-Synthesis method employs a probabilistic image analysis using Bayesian inference. Formally, the fitting pipeline is a procedure that for any given face image returns a model fit which best corresponds to the face in the input image. It is important to note that commonly applications aim to obtain a single solution as a result, however, this is not the case for fitting pipeline. Instead of aiming for a single solution the fitting pipeline produces a posterior distribution of possible solutions. Since the true posterior distribution is unknown, the fitting process needs to approximate it. To perform approximate inference in this setting the Metropolis-Hastings (MH) algorithm is used. The algorithm is a method of Markov Chain Monte Carlo sampling and it is especially useful when sampling from distributions from which direct sampling is not feasible. The algorithm is an iterative process that draws random samples θ' from a proposal distribution $Q(\theta' \mid \theta)$ that are being evaluated based on their likelihood value. In the figure bellow single step of MH algorithm is shown.

Algorithm 1: Metropolis-Hastings algorithm

- Initialize sample $\theta \sim Q(\theta)$;
- Generate the next sample with current sample θ:
  1. Propose a sample θ' based on current sample: $\theta' \sim Q(\theta' \mid \theta)$;
  2. With probability: $\alpha = min\left\{ \frac{P(\theta')}{P(\theta)} \frac{Q(\theta|\theta')}{Q(\theta'|\theta)}, 1 \right\}$ accept or reject samples;
  3. If sample θ' is accepted set θ' to be a new state θ, otherwise keep θ unchanged;

The two principal parts of the algorithm are the proposal distribution Q from which samples are drawn and the likelihood estimation $P(\theta)$. Let us describe both of these key components in detail.

The proposal distribution should be simple enough to draw random samples from it. In the settings of the fitting pipeline commonly used proposal distribution is a simple Gaussian random walk proposal distribution:

$$Q(\theta' \mid \theta) = \mathcal{N}(\theta' \mid \theta, \sigma^2 I_d). \tag{2.5}$$

Where d stands for the dimension and σ controls the step size the algorithm makes at each sampling iteration. In the fitting pipeline we usually have more than one proposal distributions for each 3DMM parameter update $\theta = \{\theta_S, \theta_C, \theta_E\, \theta_P, \theta_L\}$. Therefore, in practice, $Q(\theta' \mid \theta)$ consists of multiple proposal distributions Q_i combined into one mixture proposal distribution:

$$Q(\theta'|\theta) = \sum_i c_i Q_i(\theta'|\theta), \sum_i c_i = 1. \tag{2.6}$$

Where θ is a current sample, θ' is a newly proposed sample drawn from a proposal distribution Q_i, and c_i coefficient controls how often samples are being drawn from the specific Q_i distribution[2].

The second key part of the MH algorithm is a likelihood estimation term $P(\theta)$ commonly written as $P(\theta \mid \mathcal{I})$. It determines a posterior belief of the current sample parameter set θ conditioned on the target image $\mathcal{I}$. Computing the posterior directly is not possible, therefore, we employ the classical Bayes' theorem[5] to approximate it based on the prior knowledge $P(\theta)$ and an image likelihood $P(\mathcal{I} \mid \theta)$:

$$P(\theta \mid \mathcal{I}) = \frac{P(\theta)P(\mathcal{I} \mid \theta)}{\int P(\mathcal{I} \mid \theta)P(\theta)d\theta} \tag{2.7}$$

In this setting finding a single solution then becomes a problem of a maximum-a-posteriori (MAP) inference, which is finding the parameters with the highest posterior probability[3]. Since we are only interested in the ratio of two different likelihoods $P(\theta \mid \mathcal{I})$ and $P(\theta' \mid \mathcal{I})$ the normalization term can be ignored, and we can rewrite the above formulation as:

$$P(\theta|\mathcal{I}) \propto P(\theta)P(\mathcal{I}|\theta) \tag{2.8}$$

The prior probability $P(\theta)$ of the model estimate θ is usually defined as a normal distribution:

$$P(\theta) = \mathcal{N}(\theta|0, \mathcal{I}). \tag{2.9}$$

The only missing part of the Equation 2.10 than is an image likelihood $P(\mathcal{I} \mid \theta)$ which can be reformulated as $P(\mathcal{I} \mid \theta) = \mathcal{L}(\theta; \mathcal{I})$ with equation 2.10 transforming to:

$$P(\theta|\mathcal{I}) \propto \mathcal{L}(\theta; \mathcal{I})P(\theta) \tag{2.10}$$

An image $\mathcal{I}$ that appears in likelihood term means that during the fitting target image is treated as an observation of the 3DMM model instance. Depending on the phase the fitting pipeline is in, a combination of likelihood functions is used. We briefly discuss the most important likelihood functions used in the fitting pipeline alongside with the phase they are used in.

[5] Bayes' theorem — https://en.wikipedia.org/wiki/Bayes'_theorem

2.2.2 Landmark Fitting

During the starting phase of the fitting, it is important to have good pose estimation parameters since the pipeline is crucially dependent on it. If the pose is wrong, the rest of the model parameters cannot be inferred reliably. To deal with this problem, the standard fitting pipeline relies on the landmark fitting phase. During this phase fitting only modifies 3DMMs θ_P and θ_S parameters. It does so by evaluating a set of landmark point locations x_i detected (by detection algorithm or manually) onto the target image and projected to the model instance using computer graphics. The projection method uses a pinhole camera model to align model instance to the target face by applying a rotation and translation parameters and then render individual landmarks onto the image plane. By treating those points as an observation of corresponding model points, the algorithm evaluates them using isotropic Gaussian likelihood function:

$$\mathcal{L}(\theta; x_1, x_2, \ldots, x_N) = \prod_{i=1}^{N} \mathcal{N}(x_i \mid y_i(\theta), \sigma_{LM}^2 I_2)[2]. \tag{2.11}$$

Where x_i are observed landmark positions and $y_i(\theta)$ are model landmark positions based on θ. Proposed samples with new camera and shape parameters are getting accepted or rejected based on its likelihood value according to the procedure discussed earlier in Algorithm 1. Pose parameters that are part of θ_P are a combination of Euler rotation angles $\{yaw, nick, roll\}$, translation vector $\vec{t} = \{t_x, t_y, t_z\}$ with t_z indicating the distance from the camera, and scaling parameter to control the focal length of the camera. The proposal distribution (Equation 2.6) for landmark fitting consists of a mixture distribution over shape and camera (pose) parameter update proposals. As discussed previously, shape parameters θ_S are PCA coefficients of the low-rank expansion of the Gaussian Process[13] model that are proposed by a weak isotropic Gaussian perturbation proposal[2]. The described process adjusts the pose and at the same time makes initial shape correction. Landmark fitting is usually a very fast process since the only few landmark points (usually less than 10) are getting evaluated at a time.

2.2.3 Color Fitting

After the landmark fitting phase the pose of the model instance usually fits well with the target face in the image and the model instance is ready to be utilized for color, illumination and optionally expression proposals as well as shape update proposals. Thus, during the color fitting phase the mixture proposal $Q(\theta' \mid \theta)$ will be a combination of all the above mentioned parameter proposals with various step size and drawing probability as of Equation 2.6. To evaluate proposed samples in this phase the standard fitting pipeline together with landmark evaluation (Equation 2.11) relies on independent pixel evaluator likelihood which distinguishes foreground and background pixels[14] and is formulated as follows:

$$\mathcal{L}\left(\theta; \tilde{\mathcal{I}}\right) = \prod_{i \in \mathcal{F}} \mathcal{N}\left(\tilde{\mathcal{I}}_i \,\middle|\, \mathcal{I}_i(\theta), \sigma^2 I_3\right) \prod_{i \in \mathcal{B}} \mathcal{L}_{\mathrm{BG}}\left(\tilde{\mathcal{I}}_i\right). \tag{2.12}$$

Where the first product evaluates foreground pixels $i \in \mathcal{F}$ and second product evaluates background pixels $i \in \mathcal{B}$. The motivation behind incorporating background pixels into this

likelihood and not ignoring them is that, when they are ignored, their likelihood value is assumed to be 1, which is not necessarily true in most cases.

So called propose-and-verify flow of the initial architecture of the Metropolis-Hastings algorithm used by [2, 8] is shown in Figure 2.3. As we have mentioned previously, the architecture only uses color image and 2D landmarks as an input.

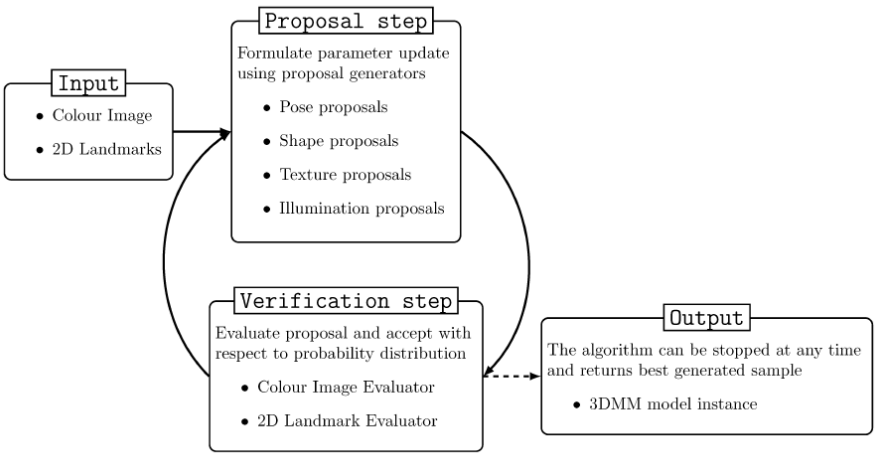


Figure 2.3: The architecture of the MH algorithm used by [2] taken from [7]

.

An augmented version of this flow by [7] can be seen in Figure 2.4, where authors introduced additional depth image and 3D landmark input sources, with respective evaluators.

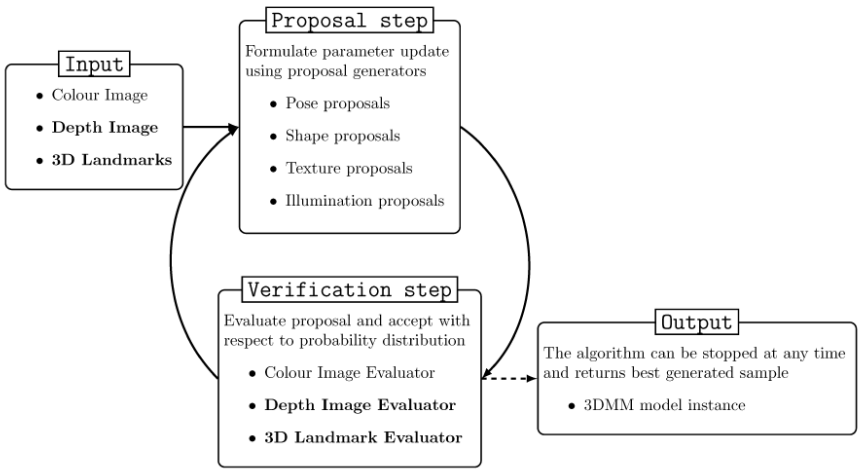


Figure 2.4: Augmented MH architecture proposed by [7], changes made in the architecture are in bold.

We introduce an alternative way of dealing with this architecture. Instead of using just

a depth image (only z-Buffer) we construct a triangle mesh from a point cloud obtained using the depth camera. Details of this process are described in Section 3.3.3. We were also forced to find a work-around way to obtain 3D landmarks (Section 3.3.2), since they are not provided by the camera SDK anymore. We split the standard fitting pipeline into two sub-fitting pipelines, one that deals with the shape and pose parameter estimation hereafter referred to as *shape fitting module* and the other with color, illumination, and expression with slight shape and pose parameter updates hereafter referred as *color fitting module*. Both of these modules are described in Section 3.4.

2.3 Depth Camera

Inferring the exact size of an object when analyzing color images is a challenging task, there is no effective way to reliably recover this information based on color intensities. The depth camera helps us to resolve this issue by providing a distance measure for each pixel (if it is available). There are a few different ways distance is obtained by the depth cameras. One of the algorithms commonly used (and the one our camera uses) to calculate distance is the depth from stereo algorithm better known as **Stereoscopic Vision**[6] which is a realization of a natural Binocular vision. The basic idea behind Stereoscopic vision is to estimate the distance from the camera to each point by calculating disparities between two parallel view-ports[15, 16] (Figure 2.5). View-port parallelism is achieved with the traditional Image Rectification[7] approach, which transforms images onto a common virtual image plane and makes sure that two view-port points are matching.

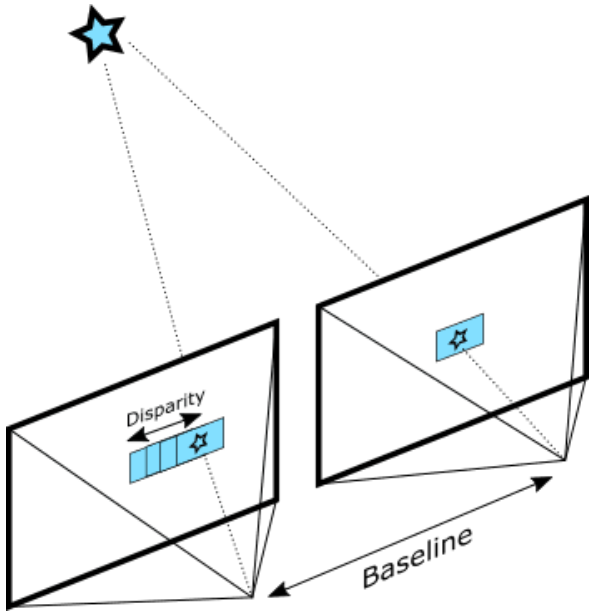


Figure 2.5: Depth from stereo. Visualizing two separate view-ports that are being used to calculate disparities. Source: Intel

[6] https://en.wikipedia.org/wiki/Computer_stereo_vision
[7] https://en.wikipedia.org/wiki/Image_rectification

2.3.1 De-projection

Pixel-to-Point and Point-to-Pixel de-projection is a common use-case in computer vision. When needed this feature offers pixel to point mapping and vice-versa by relying on cameras' depth information and intrinsic parameters. Successful de-projection is based on the traditional pinhole camera model [17] shown in Figure 2.6.

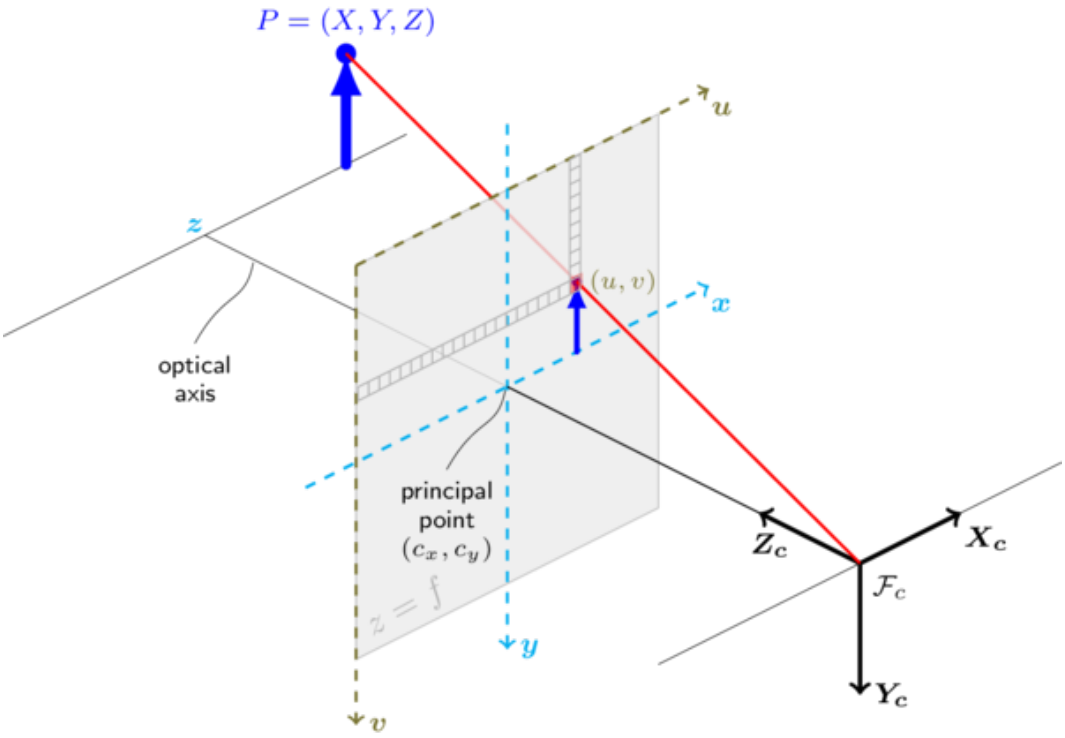


Figure 2.6: Pinhole camera model. $\mathcal{F}_c$ is the center of the camera, (X, Y, Z) are coordinates of a 3D point in the world coordinate system, (u,v) are coordinates of a respective point projection in pixels, (c_x, c_y) indicates principal points usually located at the image center. Source: https://docs.opencv.org

The problem of point to pixel projection is formulated as follows. Given a 3D point coordinates $P_{3D}(X, Y, Z)$ with camera intrinsic parameters[8] containing (*width*, *height*, *ppx*, *ppy*, f_x, f_y), calculate the respective pixel coordinates $P(u, v)$ with no distortion introduced. Where in camera parameters, *width* and *height* are image dimensions in pixels, *ppx* and *ppy* are the horizontal and vertical coordinate of the principal point of the image given as an offset from the left and top edge respectively, and the f_x and f_y are the focal lengths of the image as a multiple of pixel width and height. The focal length is usually the distance from the center of the camera to the image plane also known as the focal plane. It is shown as gray square in Figure 2.6. Once we have all these variables, then the u and v pixel coordinates of $P(u, v)$ are calculated as follows:

[8] https://github.com/IntelRealSense/librealsense/blob/master/include/librealsense2/h/rs_types.h#L55

$$
\begin{aligned}
x' &= \frac{X}{Z} \\
y' &= \frac{Y}{Z} \\
u &= x' \cdot f_x + ppx \\
v &= y' \cdot f_y + ppy
\end{aligned}
\tag{2.13}
$$

On the other hand, the problem of de-projecting pixel coordinates into the 3D coordinate space to obtain a 3D point is formulated as follows. Given a pixel coordinates $P(u, v)$ and depth information $d = Z$ alongside the camera parameter set mentioned previously with no distortion introduced, compute the corresponding $P_{3D}(X, Y, Z)$ point in the 3D space. Then $P_{3D}(X, Y, Z)$ coordinates are being calculated as follows:

$$
\begin{aligned}
x' &= \frac{u - ppx}{fx} \\
y' &= \frac{v - ppy}{fy} \\
X &= d \cdot x' \\
Y &= d \cdot y' \\
Z &= d
\end{aligned}
\tag{2.14}
$$

Both techniques we have discussed are already implemented in the camera SDK for us, therefore, throughout the project, we make use of those default methods provided by the SDK.

3
Methods

In this chapter, our main contributions are elaborated. We explain in detail our proposed method of the fitting pipeline, including its key components that we have put together to build the module based coherent and reliable system. The overview diagram of the system we have implemented can be seen in Figure 3.1. The system makes sure that the data from the camera is seamlessly delivered to the fitting pipeline in the correct format and being utilized efficiently. Further, in this chapter, we describe each of the key components of the system that could be split into three distinct modules:

- Module 1: takes care of the camera configuration, data capturing and preprocessing procedure (Figure 3.1 **Server** block). Among others, the data capturing procedure includes the acquisition of 3D landmarks, which introduces an additional technical challenge since they are not directly obtainable from the camera.
- Module 2: receives the data from the server, processes it, and performs the shape fitting (Figure 3.1 **Client & Shape Fitting** blocks[9]). Besides the more important shape fitting procedure, this module also takes care of minor but nevertheless important aspects of the module, such as data conversion and cleaning.
- Module 3: receives the best shape fit produced by the shape fitting module, alongside other necessary data, and performs the color fitting (Figure 3.1 **Color Fitting** block). The color fitting performed in this module is very much like the standard fitting approach, except small modifications as we discuss them in the following chapters.

In this way, it is very easy to control and use those modules for specific applications, like only producing the data and using it for other purposes, or only obtaining the shape fit where the color fit is not required.

We first explain in detail the camera calibration and configuration procedure. Then we introduce the software development kit that allows us to communicate with the camera and perform the data acquisition procedure. Further, the client-server API is introduced.

[9] In practice, these two modules are merged into one, we split them here to make procedures clear.

Based on this API data is seamlessly being delivered to the module that utilizes it. We also describe the data capturing procedure alongside the client-server interaction. Finally, and most importantly both of our fitting modules are elaborated, including the description of the proposals and evaluators used.

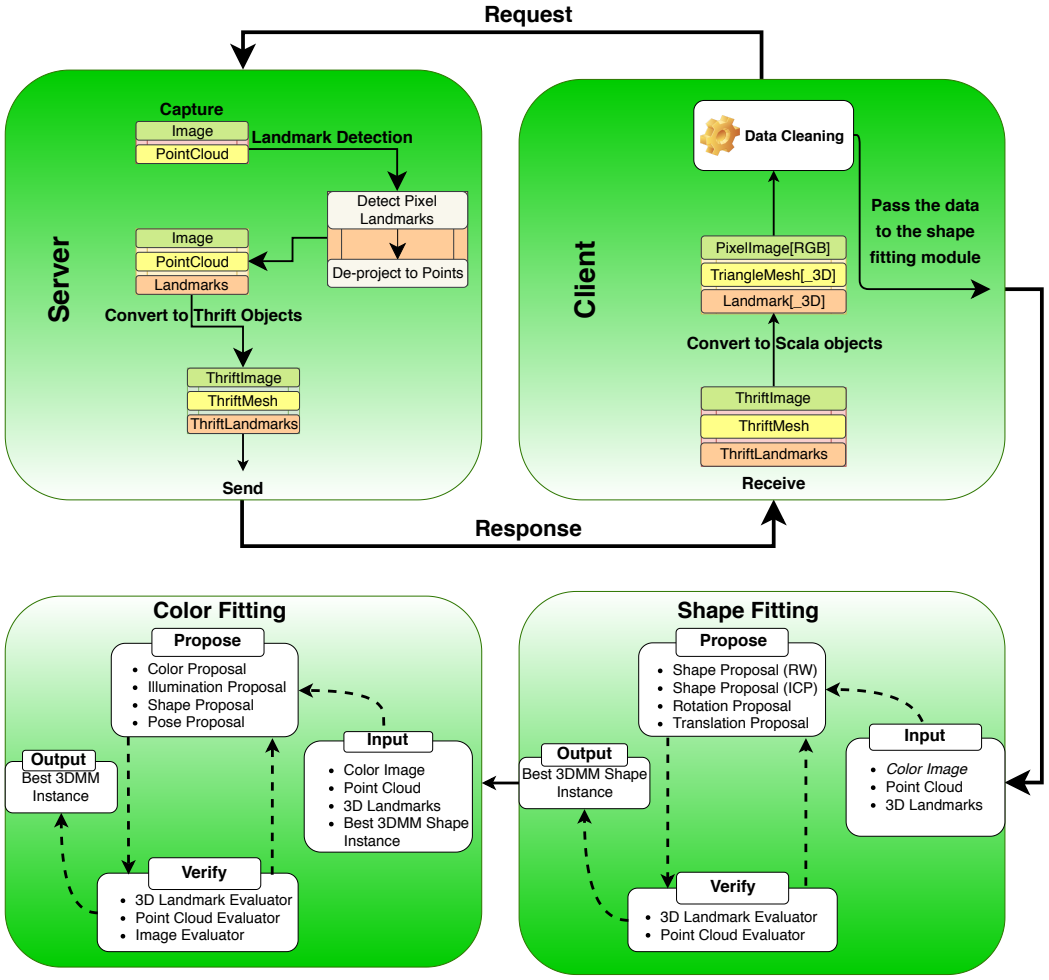


Figure 3.1: An overview of our system with its key modules and procedures.

3.1 Camera

Before we start working on our project is it essential to configure and calibrate the camera in order to maximize its performance. As we have mentioned previously, our camera of choice is the Intel® RealSense™ D415 depth camera. Thanks to its rolling shutter and narrow field of view, the camera is focused on accuracy and is especially well suited for near and still object scanning applications.

3.1.1 Camera Calibration

To calibrate the camera we use Intel's Dynamic Calibrator tool. The camera can be calibrated by either displaying a calibration texture pattern on a mobile device screen or printing it on a white paper and following the Calibrator tool instructions (Figure 3.2). Dynamic calibration tool optimizes the cameras' extrinsic parameters with regard to the main axis, consisting of rotation and translation parameters[18].

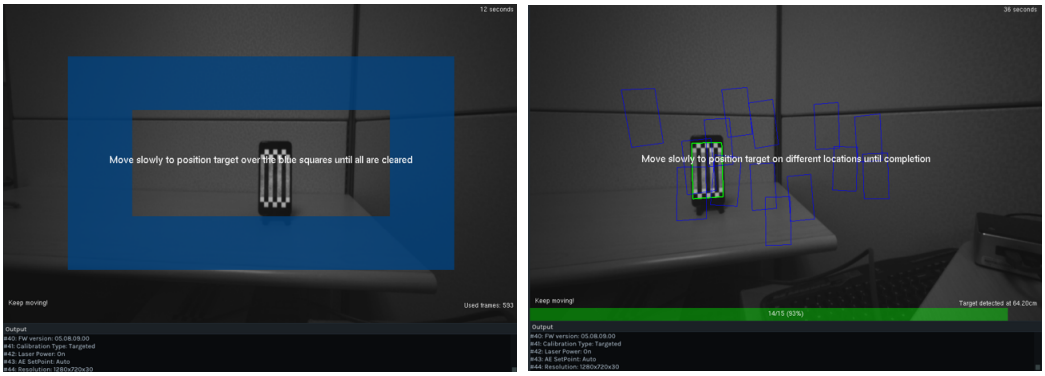


Figure 3.2: Left: Rectification phase, Right: Scale phase. Source: Intel

To evaluate the performance of our camera after the calibration procedure we use Depth Quality tool[10] (Figure 3.3).

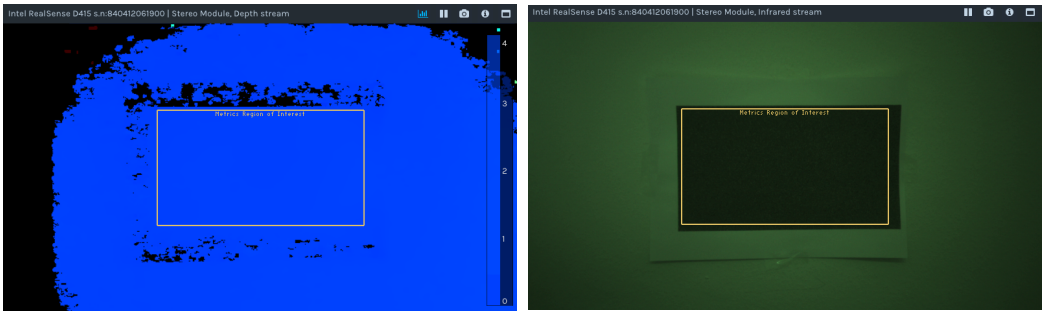


Figure 3.3: Depth quality tool using Intel's recommended textured wall test method to assess performance, displaying depth (left) and infrared (right) stream view, alongside with an ROI area marked with a yellow rectangle.

By running a series of recommended tests we obtain a set of important quality measurements presented in Table 3.1. **Fill rate** measures the percentage of pixels with valid depth value within the Region-Of-Interest (ROI). **Z-accuracy** indicates the depth accuracy given the Ground Truth (GT) as a percentage of the range, where positive value means that the depth plane fit is behind the ground truth (overshot), and a negative value indicates that the depth plane fit is in front of ground truth (undershot). **Plane Fit RMSE** calculates the Root-Mean-Squared (RMS) error of Z-error (Spatial noise). **Sub-pixel RMSE** provides sub-pixel RMS error.[11]

[10] Depth Quality Tool — https://github.com/IntelRealSense/librealsense/tree/master/tools/depth-quality
[11] This information is taken from the depth quality tool itself, refer to footnote link above for more details.

Measurement	Result	ROI	Resolution
Fill Rate	100%	40%	1280x720
Z-Accuracy	0.24%	40%	1280x720
Plane Fit RMSE	0.24%	40%	1280x720
Sub-pixel RMSE	0.13%	40%	1280x720

Table 3.1: Calibration test results.

3.1.2 Camera Configuration Parameters

After the camera calibration, we proceed with configuring the camera parameters including resolution, stream profiles (Color and Depth stream separately), measurement distance, filters and a long set of other depth configuration parameters. If not stated otherwise we do not modify any sensitive parameter and use Intel's recommended settings as of [19]. Although Intel recommends to use 1280x720 image resolution for an optimal image quality and depth accuracy, we have decided to use 640x480 resolution to right away reduce the number of calculations needed to do any sort pixel-wise or point-wise operations and yet maintain a decent image quality. 1280x720 resolution yields 900k pixels/points as opposed to 300k in the case of 640x480. When it comes to camera stream formats we use traditional *z16* format for the depth stream and *rgb8* format for the color stream. We enable auto exposure setting which allows the camera to dynamically adjust exposure for the various light and illumination environments. Since our application only requires a still relatively close-up distance head shots we set the minimum depth distance to 0.2 meter and maximum distance to 1 meter as well as the ROI parameter to 40%. This is because we are not interested in the entire scene captured by the camera but rather the face area. As of filters, we did not make use of them because we opted to work with raw data without various filters influencing its quality. The final note we would like to make regarding the camera configuration is that we align color and depth stream frames. The reason is straightforward, we would like to have the data that have the same view-point — color and depth frames should be in correspondence. If we do not align them, they will be viewed from slightly different angles, this is due to spatial differences between physical camera sensors. This process will become more obvious in Section 3.3.2 were we describe the process of capturing 3D landmarks.

3.1.3 SDK

To take advantage of the camera hardware and to fit the capture procedure to our needs we have to use Intel's *librealsense*[12] SDK. The reason why it is necessary to communicate with the camera through the API instead of using Intel's RealSense Viewer (Figure 3.4) is that firstly, it does not provide all the required data out-of-the-box and secondly, the data cannot be captured all at once. As we are designing our system to work in a variety of application scenarios including a web environment, it is crucial to automate all parts and run the system with minimal to no human interaction.

[12] Intel® RealSense™ SDK — https://github.com/IntelRealSense/librealsense

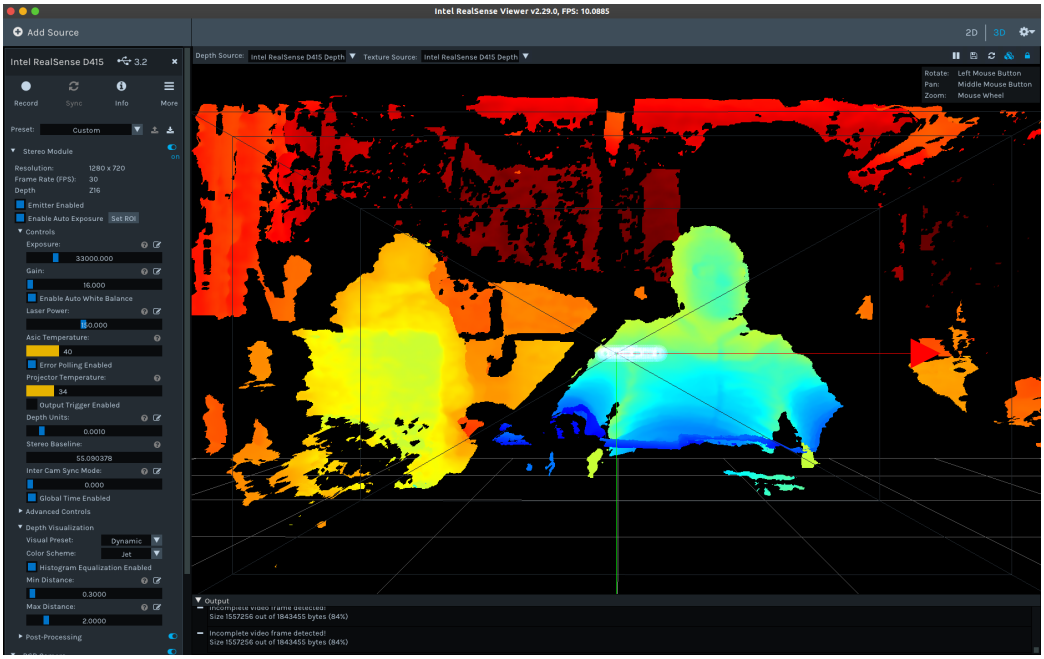


Figure 3.4: Intel RealSense Viewer — 3D Depth View with camera configuration parameters on the left side. Blue color indicates near points, red — far.

While being written in C++ the SDK provides wrappers for the majority of the popular programming languages[13]. Unfortunately, Java Virtual Machine (JVM) based languages (one of which, namely Scala we are going to use in the main part of the project) are not officially supported. Therefore, we had to choose an alternative lightweight and fast language, flexible enough to seamlessly run on any machine and operating system. We have decided to use Python because of its powerful companion frameworks, ease of use and also because of *pyrealsense2* — librealsense's Python wrapper that is stable and relatively fully featured comparing to others. We hereby note that upon investigating the camera SDK we have discovered[14] that it does not have a dedicated computer vision middleware module it used to have in the previous generation [7] used. The module used to offer relevant features for our application such as face scanning and landmark detection. Not having a direct, native access to these features introduces additional technical challenge. Since the performance of or application crucially depends on a set of facial landmarks we had to find a way to detect them. We present our solution to this problem in Section 3.3 where we describe a complete data acquisition procedure. The one last important notice we would like to make regarding the choice we have made in this section is that obviously, we cannot simply communicate between Python and Scala code base. To deal with cross-language modules we rely on interface definition language framework — Apache Thrift[15].

[13] Not all wrappers support full features offered by the native SDK
[14] GitHub Issue — https://github.com/IntelRealSense/librealsense/issues/3716
[15] Apache Thrift™ — https://thrift.apache.org

3.2 Cross-Language API

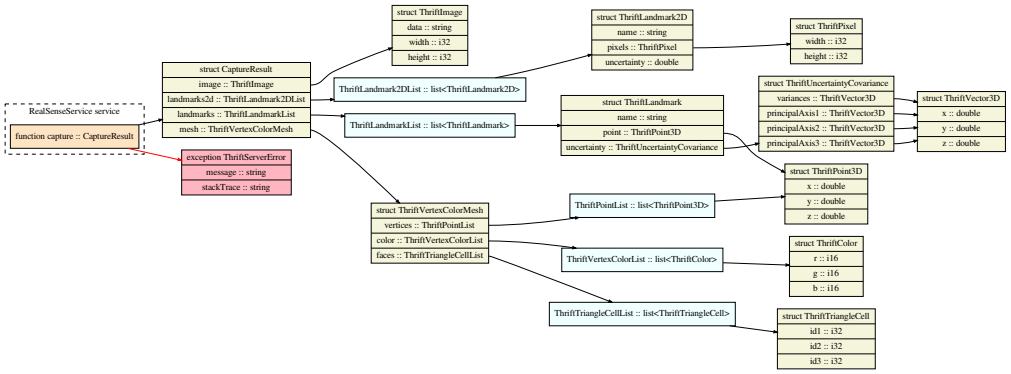


Figure 3.5: Diagram of our API. *For better resolution please refer to Appendix A.1

Thrift is an interface definition language designed to solve the problem of sharing the code base of cross-language modules. Specifying the signature of all the important types, methods and exceptions in the thrift interface — the framework generates an identical API for any major programming languages. By doing so, Thrift takes care of all the language specific details such as syntax, type variations, exceptions, etc. and makes sure that the code generated for the different languages are usability-wise identical from an API consumer's perspective. The diagram of our thrift interface can be seen in Figure 3.5. We define a set of relevant data structures such as *Pixel*, *Point*, *Vector*, *Image*, *Landmark*, *Triangle* and *Color* with respective lists that utilize some of those values *PointList*, *LandmarkList*, *ColorList*, and *TriangleList*. We also define a mesh structure with signature *TriangleMesh(PointList, ColorList, TriangleList)*, as well as, the main data holder object *CaptureResult(Image, Landmark2D, Landmark3D, TriangleMesh)*. *CaptureResult* is the object that is being passed from the server-side to the client-side. Every mentioned data type has its respective converter method implemented in both Python and Scala code base. We did not make use of 2D landmarks in this project, so we do not pay attention to it any further, however, if needed — its functionality is fully implemented in both Python and Scala code.

3.3 Capture Procedure (Server)

Upon receiving a request our server (Figure 3.6) launches a pre-configured camera stream into the OpenCV[16] window (Figure 3.7) that is split into two sub-windows, one displaying an RGB color stream and the other depth stream. Red color indicates pixels that are closer to the camera and blue that are farther. Note that the server does not execute capturing once it receives the request, instead it runs the camera stream and waits for additional user input (for example pressing the "D" key on the keyboard). This is because otherwise, the user will not be able to adjust their position. Obviously, it is possible to add timeout before capturing but having a visual tool is still preferred. Once the server receives a call to run

[16] Open Source Computer Vision Library — https://opencv.org/

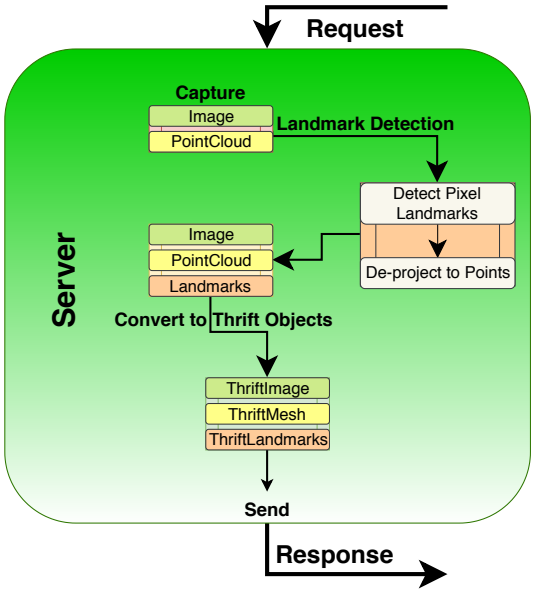


Figure 3.6: The server diagram.

the capturing script, it captures the required raw data and starts processing it. For the successful pipeline run, as we have mentioned previously, our modified fitting pipeline requires the following data: **RGB Color Image**, **A Set of 3D Landmarks** and **Point Cloud Mesh**.

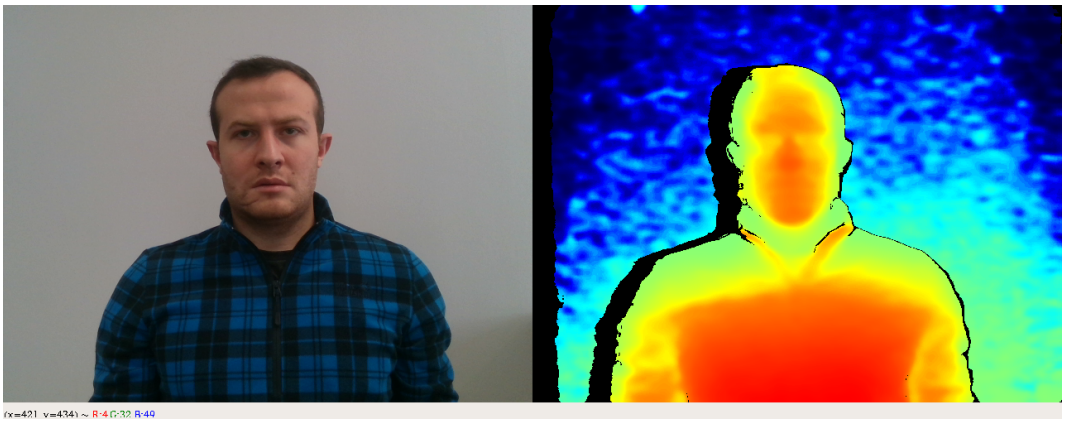


Figure 3.7: The OpenCV window displaying an RGB color image (left) and the depth image (right) visualized using JET color map.

3.3.1 RGB Color Image

Capturing a color image is not complicated, and we do not do any post-processing to it. After converting it to the Thrift image type using the corresponding converter method we directly pass it to the *CaptureResult* object described above.

3.3.2 3D Landmarks

Obtaining 3D landmarks is one of the important data requirements for this project. The problem arose due to the lack of 3D landmark availability in the SDK. The solution to this problem we have implemented is to detect 2D (pixel) landmarks on the color image using Dlib[17] landmark detection library (Figure 3.8). These landmarks are then de-projected into 3D landmark points using the camera SDKs de-projection capabilities we discussed previously. The de-projection is based on camera settings for the specific session (shot).

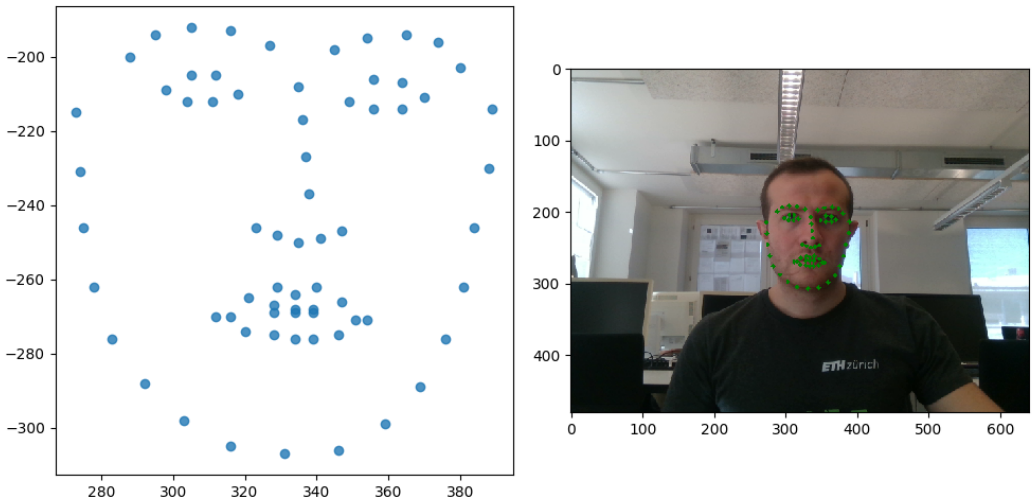


Figure 3.8: Facial landmark detection using OpenCV and Dlib libraries visualized over D415's color stream.

Dlib landmarks usually have corresponding landmark ID assigned to it, based on those IDs we match each 2D landmark to the fitting pipeline landmarks by the name. This way, after performing de-projection we create the landmark structure that corresponds to the structure defined in the fitting pipeline $\{"landmarkID": [LM_x, LM_y, LM_z]\}$. Sometimes detected pixel landmarks do not have corresponding 3D point due to missing depth information, in situations like this 3D landmark points will be filled with 0.0 values. We will deal with these landmarks on the client-side where we determine which landmarks to use. During landmark de-projection, it is essential to have color and depth frames in correspondence (aligned). Otherwise, when examining point cloud, landmarks will not appear in locations where they supposed to, because point cloud mesh will be captured based on depth stream and landmarks will be obtained via de-projecting color stream pixels to 3D points. We have already emphasized the importance of 3D landmarks in our application, therefore, we place a hard constraint over this block of information. If Dlib for some reason fails to detect the face in the image, the capturing process throws an exception. Furthermore, when there is more than one face detected in the image, we also break the process, because the pipeline would not be able to figure out which face we are trying to reconstruct.

[17] Dlib C++ library — http://Dlib.net

3.3.3 Point Cloud

The camera SDK does provide raw point cloud access out-of-the-box, however, the operations that are defined on top of point cloud objects are limited to accessing the point information, saving them as a structured mesh file, obtaining texture coordinates and getting the size of a point cloud. Saving the point cloud as a mesh file is the feature we need, however, we have to modify it, because we do not save the mesh file anywhere, instead serialize and sent it through the network. We have implemented an overridden version of the mesh formation in order to satisfy this requirement.

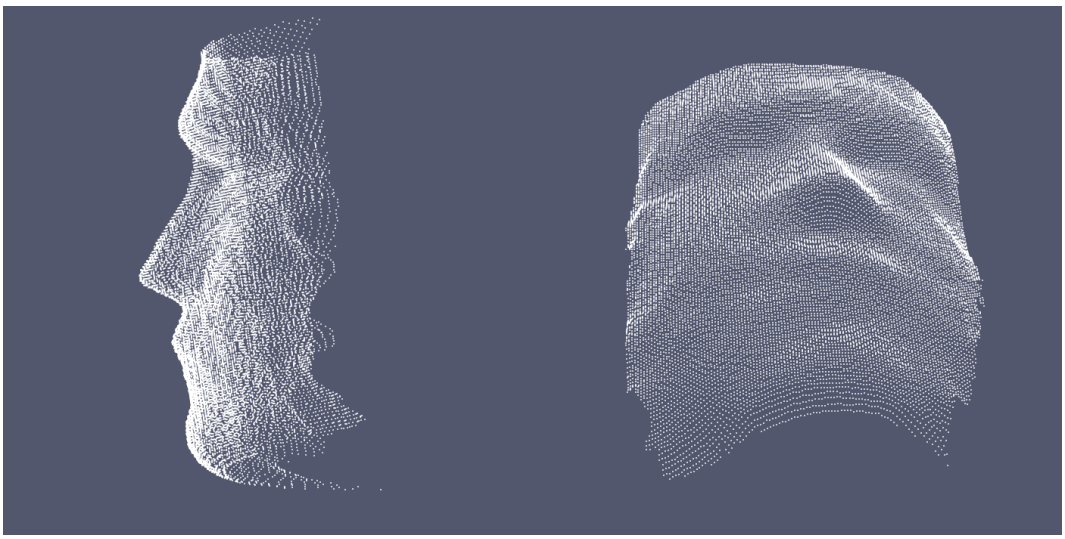
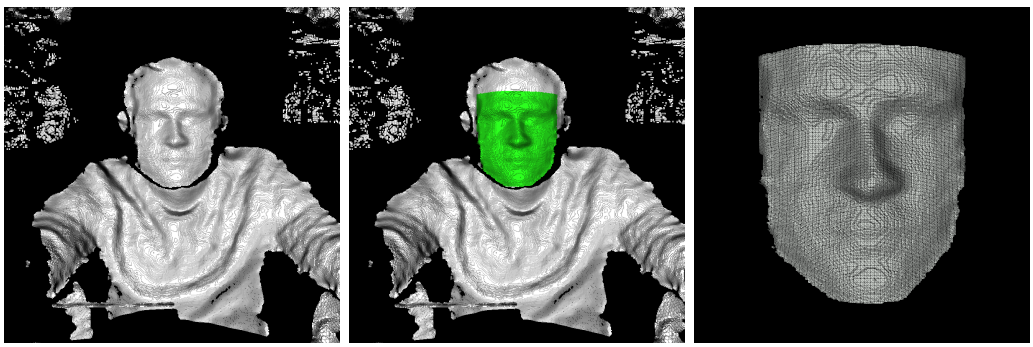


Figure 3.9: Obtained point cloud clipped at face region.

The method takes point cloud provided from the SDK and creates triangles between neighboring points. Triangle formation itself is based on the same algorithm Intel uses to save the mesh as a file. That is, we only modify the usability of the method but the algorithm is unchanged.



(a) (b) (c)

Figure 3.10: (a) Full scene point cloud mesh. (b) The same scene as (a) but with region of interest highlighted in green. (c) Clipped face region (target mesh)

The point cloud obtained from the SDK contains points which are describing the entire

scene most of which we are not interested in, and more importantly having unnecessary points introduces additional computational overhead during the fitting. Therefore, we consider clipping the point cloud in the face region and discard the rest of the points. This process happens on the client-side which we yet to describe. The basic flow of mesh clipping operation is shown in Figure 3.10. We will describe this process further in Section 3.4.1.

3.4 Fitting Pipeline (Client)

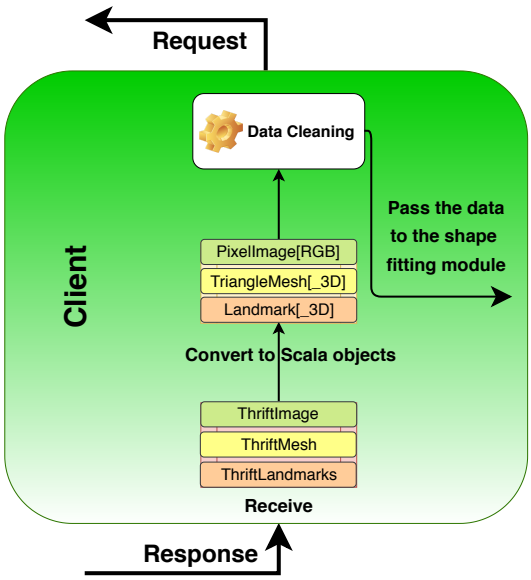


Figure 3.11: The client diagram.

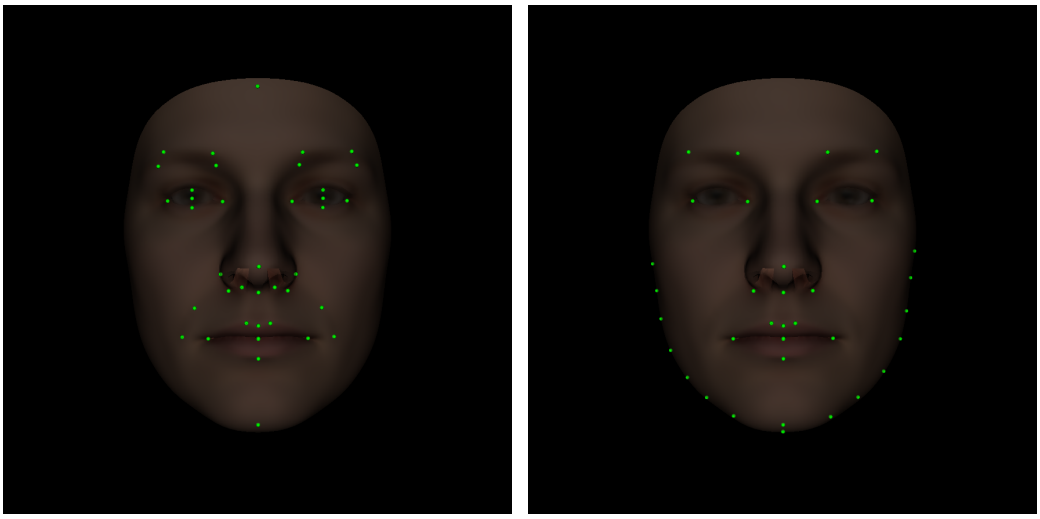
As described previously, the fitting pipeline is split into two parts. The first part (Figure 3.11) is responsible for receiving the data from the server, converting it to respective Scalismo[18] objects (*PixelImage, Landmark3D, TriangleMesh*), performing the necessary data cleansing procedure, and finally starting the shape fitting process. The shape fitting utilizes additional point cloud information by fitting the 3DMM onto the point cloud to obtain better pose and shape model parameters. We describe this process in Section 3.4.2. The second part of our proposed fitting pipeline is the color fitting stage. It is a continuation of the shape fitting in the sense that it takes the result of the shape fitting as an input, and runs the slightly modified standard color fitting. This process is described in Section 3.4.3.

3.4.1 Data Processing

Before shape fitting starts it is necessary to process and clean the data received from the server. The more important data cleansing happening with *Landmark3D* and *TriangleMesh* structures. Before we do that, we augment the current model with additional chin landmarks that we are getting from the Dlib, and remove some which we do not have in Dlib.

[18] Scalismo — Scalable Image Analysis and Shape Modelling https://scalismo.org/

To make this process better understandable we visualize landmark differences in Figure 3.12. Where the figure on the left shows the original landmarks that are available in the model, and the figure on the right shows our version of it. It should be noted that the pose variation in the image tends to affect obtained chin landmarks significantly, and the pixel-to-point de-projection we perform further increases the uncertainty of those landmarks. Therefore, to deal with this problem we assign higher uncertainty to chin landmarks that are being received from the server.



Original model landmarks Landmarks we use

Figure 3.12: Visualization of the differences between the original model landmarks and the landmarks we use.

Once we have our model ready we discard landmarks that contain 0.0 values, which were produced by de-projection due to unavailable depth information. We also discard landmark points that are being de-projected wrongly due to the occlusion or frame misalignment. This is done by evaluating each landmarks' z coordinate which indicates its distance from the camera and discarding landmarks that are farther than predetermined capture distance. Additionally, we remove model landmarks that are not part of our target landmarks. Based on cleaned up target landmarks we perform mesh clipping procedure.

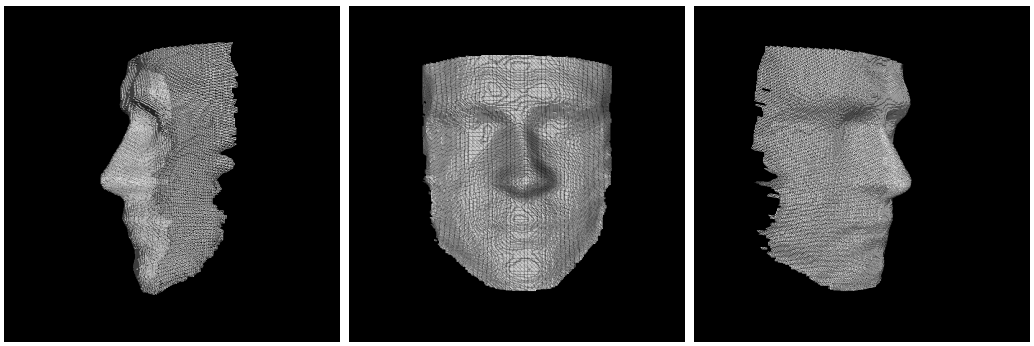


Figure 3.13: Clipped sample target mesh from different angles.

The basic idea is to create a box region covering the face by determining the outer most landmark positions for each dimension and ignoring all the points that lie outside this region. The result of this procedure can be seen in Figure 3.13. The only issue with this approach is that determining outer most landmarks and slicing based on their value will slice up some part of the face if the pose is too extreme. This is because slicing operation is happening parallel to basis axes. Hence, we recommend to work with no more than 30° degree yaw rotations and for optimal performance frontal shots where occlusion and frame misalignment are not affecting to the landmarks. The final result of the mesh slicing and landmark filtering is shown in Figure 3.14.

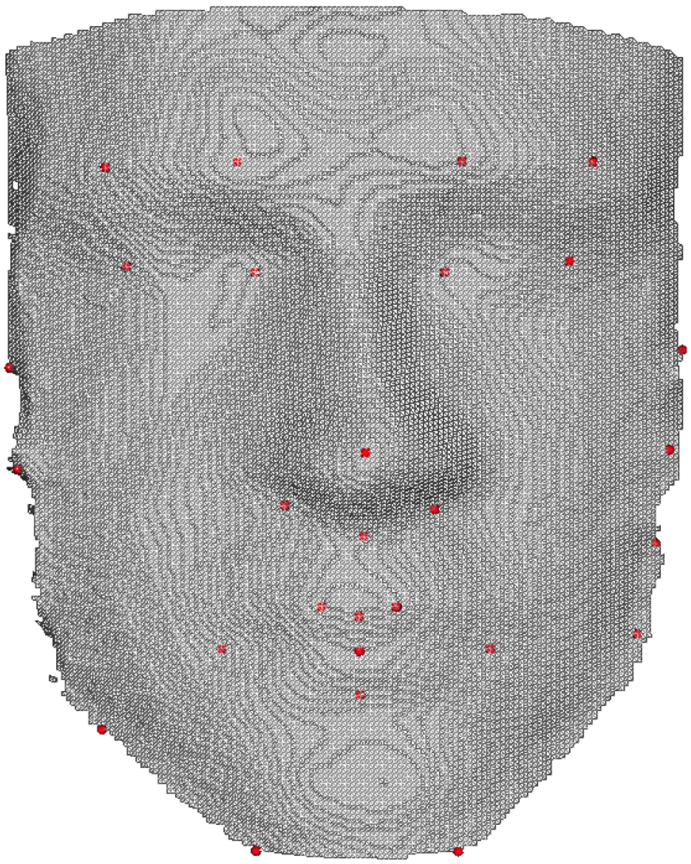


Figure 3.14: Sample target mesh with filtered landmarks.

Note that there are fewer chin landmarks available on the target mesh due to the landmark filtering process described previously. This can be explained by taking into account what we have described in Section 3.3.2. The pixel-to-point de-projection is successful, if and only if, the camera detects the respective depth information for the specific pixel we are trying to de-project. Since chin pixels commonly suffer from occlusion, camera often is not able to detect the depth information. Hence, we have some 3D landmarks that are missing from the target mesh.

3.4.2 Shape Fitting

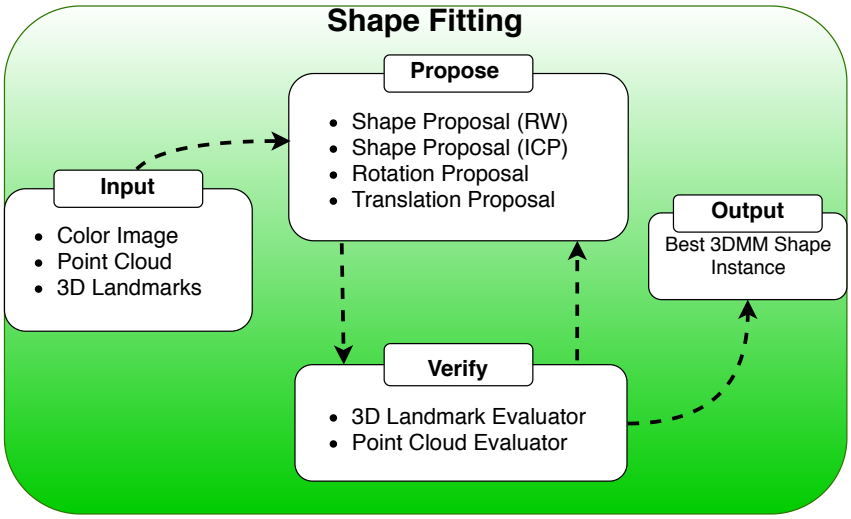


Figure 3.15: The shape fitting diagram.

Before we start the shape fitting process (Figure 3.15) itself, we first align the model and the target mesh rigidly based on landmark correspondences using rigid 3D landmark registration method. The method, which is already implemented in Scalismo, performs generalized Procrustes analysis[20] to obtain the translation and rotation parameters, which we then use to give the model a good starting position parameters. This step is required because the model is centered at the origin of the 3D coordinate system, whereas the target mesh location is determined by how far the face was from the camera when the shot was made[19]. A visual example of this scenario can be seen in Figure 3.16.

Figure 3.16: Initial location of the model (right) and the target (left).

The process alternatively can be performed during the fitting process itself, but we opted to obtain necessary parameters using the rigid alignment because it is usually very fast and leads the model to a correct initial position.

Shape fitting boils down to optimizing the set of parameters $\theta = \{\theta_S, \theta_R, \theta_T\}$ referring to

[19] The camera center is located at (0, 0, 0)

the model's *shape*, *rotation*, and *translation* parameters respectively. The shape parameter θ_S is a vector of model coefficients $\theta_S = (\alpha_0, \ldots, \alpha_{199})$. The rotation parameter θ_R consists of Euler rotation angles $\theta_R = (r_\phi, r_\psi, r_\omega)$ and the translation parameter θ_T is a vector holding 3D translation information $\theta_T = (t_x, t_y, t_z)$. As we have elucidated previously, the initial rotation and translation parameters are set to be an output of rigid landmark registration procedure, which leads to the state shown in Figure 3.17.

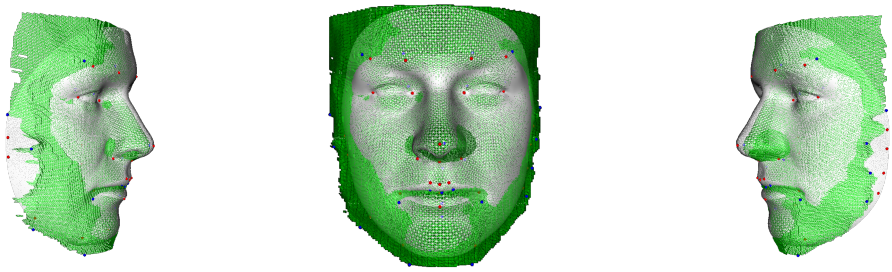


Figure 3.17: The model instance and the target mesh alignment after applying the initial θ.

Proposal Architecture

The proposal distribution that will be used in the Metropolis-Hastings algorithm for the shape fitting is constructed similarly to the mixture proposal distribution described previously according to Equation 2.6. We create a mixture of three separate proposal distributions each responsible for sampling shape, translation and rotation parameters. Separating proposals from each other not only gives us better control over the individual proposals, but it also simplifies the analysis of the process as a whole. For example, this allows us to specify the step size (standard deviation) of each proposal separately. The rotation and translation proposals both are simple Random Walk proposals with isotropic Gaussian perturbation distribution. For the shape proposal though, instead of using just Random Walk proposal we use a combination of Random Walk and Iterative Closest Point (ICP) proposals. Due to fundamental characteristics of the Random Walk proposals, that is, proposing samples without having any specific target direction, they tend to require more iterations, and therefore more time for convergence. Hence, by plugging in the ICP proposal that always moves towards the target, we reduce the number of iterations and accordingly the execution time by an order of magnitude. The final mixture proposal distribution then is as follows:

$$Q(\theta'|\theta) = \frac{2}{10}Q_R(\theta'|\theta) + \frac{2}{10}Q_T(\theta'|\theta) + \frac{3}{10}Q_{S(RW)}(\theta'|\theta) + \frac{3}{10}Q_{S(ICP)}(\theta'|\theta). \tag{3.1}$$

Where Q_R is the rotation proposal, Q_T translation proposal, $Q_{S(RW)}$ random walk based shape proposal and $Q_{S(ICP)}$ ICP based shape proposal. The coefficients of the proposals are c_i from Equation 2.6 indicating the probability of drawing samples from the specific proposals.

Evaluators

The main purpose of the evaluators is to evaluate the proposed sample θ and assess whether it should be accepted or rejected based on its posterior value $P(\theta|D) \propto P(\theta)P(D|\theta)$ as described in Section 2.2.1. We introduce two distinct likelihood evaluators under the assumption that they are conditionally independent. Based on this assumption we combine conditionally independent likelihoods using product likelihood alongside with prior evaluator. The first evaluator we model is for the landmark correspondences. Its purpose is to measure how plausible observed n landmark locations are under the influence of normal isotropic Gaussian noise assumption at every point with $\sigma = 3mm$ standard deviation:

$$P(D|\theta) = \mathcal{L}(\theta; \ell_1, \ldots, \ell_n) = \prod_i \mathcal{N}(\ell_i | \ell'_i(\theta), \sigma^2_{LM3D}). \tag{3.2}$$

Where ℓ_i are observed target landmarks and $\ell'_i(\theta)$ are i-th landmarks corresponding landmark on the model instance rendered based on θ parameters.

The second evaluator we introduce is an evaluator for uniformly sampled $n \leq model_{pts}$ points on the model instance under θ, which evaluates the distance between sampled points and the closest point on the target mesh surface. The Gaussian noise assumption is known to encounter problems when dealing with outliers. We observe quite a few outliers on our target mesh, therefore, we choose to use known to be more robust Cauchy[20] distribution[8] noise model with scale $\gamma = 0.5$:

$$\mathcal{L}(\theta; p'_1(\theta), \ldots, p'_n(\theta)) = \prod_i Cauchy(p'_i(\theta) | p_i, \gamma_p) \tag{3.3}$$

where $p'_i(\theta)$ are uniformly sampled points onto the model instance under θ parameters and p_i are its corresponding closest points on the target surface.

When it comes to a prior evaluator, we use zero-mean normal distribution for θ_T and θ_R. As for θ_S prior, we treat the model itself as a prior observation. [21]

Model Decimation

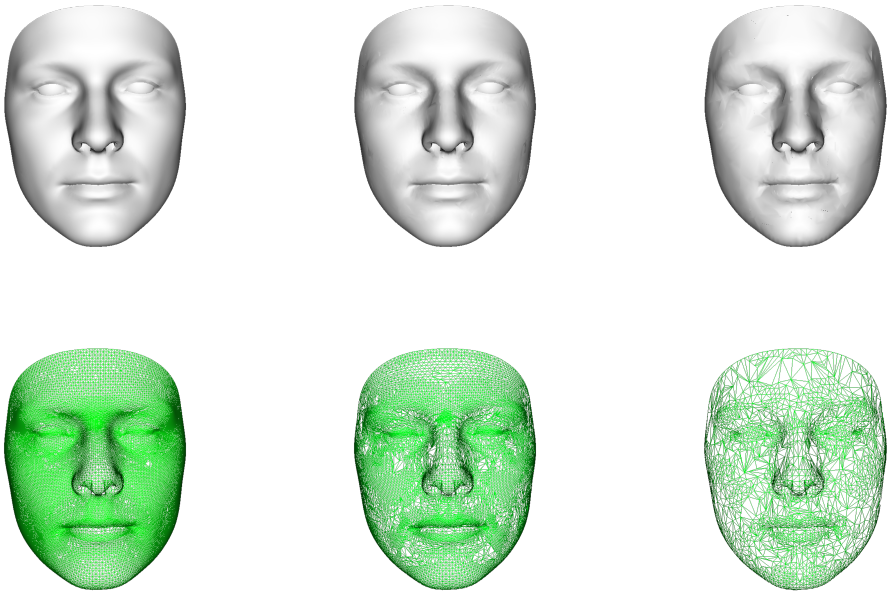
Before we put everything together and construct a Markov Chain through Metropolis-Hastings algorithm and do the inference, we have to make a small but important computationally beneficial adjustments. The model we are working on contains 28k points and 56k triangles. Every time we apply a parameter set θ to the model to obtain a model instance it has to calculate the new position for 28k points. This is a computationally rather heavy task, especially when considering the fact that we do not need to execute transformation over the full set of model points. The way to overcome this problem is to introduce the standard model decimation technique, which reduces the number of points without model losing its expressive shape characteristics. We will not go into details how the decimation process works, but rather report important differences between the full model and

[20] https://en.wikipedia.org/wiki/Cauchy_distribution

the decimated model. In Table 3.2 various decimation rates and the corresponding number of points and triangles are shown. Figure 3.18 shows how much the visual appearance changes after decimation.

Decimation Rate	Number of Points	Number of Triangles
0	28588	56572
0.1	25731	50913
0.3	20017	39600
0.6	11467	22628
0.9	2919	5657

Table 3.2: The number of points and triangles after decimation.



0.1 0.6 0.9

Figure 3.18: Various rates of model decimation and how they influence the appearance of the model.

We use the decimated model with 0.9 rate for both ICP shape proposal described previously and for the closest point evaluator (Equation 3.3).

Metropolis-Hastings

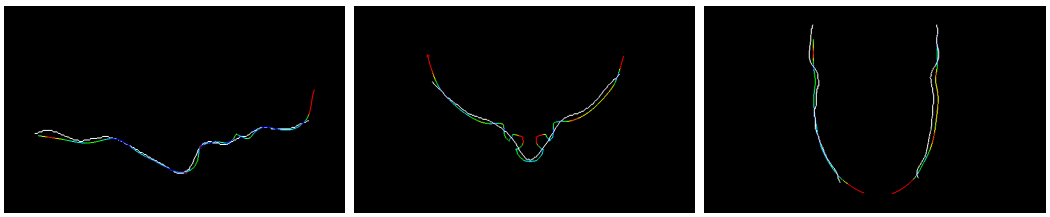
To recall, the Metropolis-Hastings algorithm is a method of the family of Markov Chain Monte Carlo algorithms, which is based on the acceptance-rejection sampling technique. As described in Section 2.2.1 it constructs a Markov Chain by accepting or rejecting samples drawn from a proposal distribution[2]. We now have defined both a proposal distribution Q (Equation 3.2) and a target distribution P which is a product of three previously men-

tioned evaluators(*Landmarks Evaluator*, *Closet Point Evaluator*, and *Prior Evaluator*). Having these two distributions is all that is needed for the Metropolis-Hastings algorithm to start the propose-and-verify cycle. Therefore, we simply pass our proposal and evaluator distributions to it. In Table 3.3 proposal acceptance ratios are shown after a test shape fitting run with 1000 iterations. As expected, our ICP proposal has higher acceptance ratios than any other proposal.

Proposal	Std. Deviation	Acceptance Ratio
ShapeProposalICP	0.1	0.6292
ShapeProposalRW	0.1	0.1909
RotationProposal	0.01	0.1428
TranslationProposal	1.0	0.0754

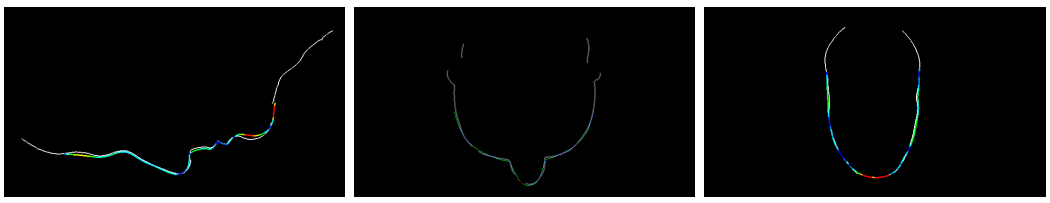
Table 3.3: Proposal acceptance ratios after 1000 sampling iterations.

The runtime of this specific run was *84s* on a standard laptop computer.[21] On a better performing machine, the execution time easily falls under *60s*. Decreasing the number of iterations from 1000 to 500 further reduces the time on earlier mentioned machine from *84s* to *56s* without sacrificing fit quality too much.[22] We will report detailed results in the next chapter but to make the above mentioned test more clear, we provide quantitative and qualitative assessments for that specific run.



X Slice Y Slice Z Slice

Figure 3.19: The white line is our target mesh, colored line is the best fit converted into the scalar mesh field with scalar distance range *0.0 mm* (blue) to *3.0 mm*+ (red).



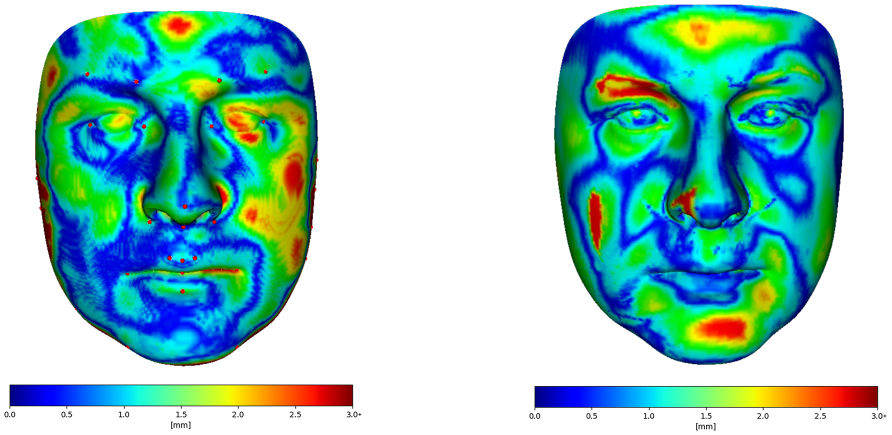
X Slice Y Slice Z Slice

Figure 3.20: The white line is the ground truth, colored line is the best fit converted into the scalar mesh field with scalar distance range *0.0 mm*(blue) to *3.0 mm*+(red).

The average mesh distance calculated point-wise between the best model fit and the target

[21] Intel® Core™ - i7-8550U, 32GB RAM, Intel® UHD Graphics 620
[22] Mentioned run times are only shape fitting times without client-server protocol

mesh shown in Figure 3.19 was ***1.26 mm***. The average mesh distance calculated between the same best model fit and the ground truth of the target person yielded ***0.82 mm***. Figure 3.20 shows the result of ground truth and best fit alignment with the same scalar distance range (*0.0 – 3.0 mm*). We hereby note that the red line visible on X and Z slices (better seen on Figure 3.21 right) that indicates *3.0 mm+* distance between the points are due to the ground truth scan having a gap in the mandible region as well as in the nose, cheek and eyebrow regions.



Target mesh vs. Best shape fit (*1.26 mm*) Best shape fit vs. Ground Truth (*0.82 mm*)

Figure 3.21: Mesh distances colored via scalar mesh field.

3.4.3 Color Fitting

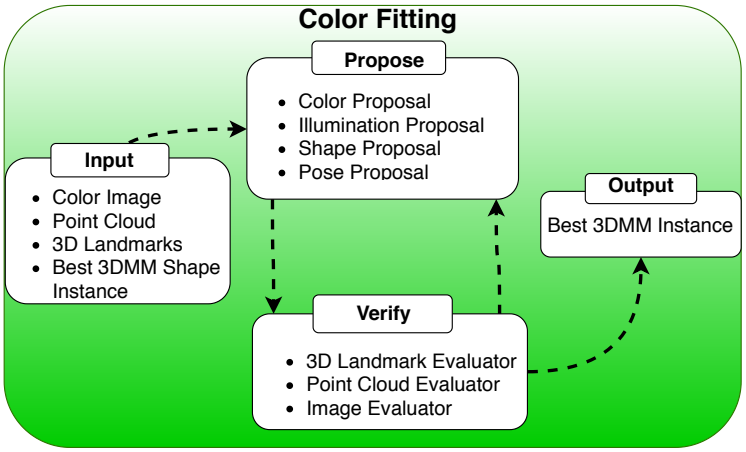


Figure 3.22: The color fitting diagram.

After the successful run of the shape fitting module, we collect the resulted data and pass

it to the color fitting module (Figure 3.22). We do not make too many changes in the color fitting part. If not stated otherwise we use standard color fitting settings from [2]. The most important data structure that is very important throughout the entire fitting pipeline is so called *RenderParameter*. It is an object which holds all the necessary parameters for the shape model to render reconstructions. It combines parameters of the model such as shape, color, expression, illumination, and camera parameters. Each of these parameters are updated via dedicated proposals. We go through proposals that we modify and describe what kind of changes, if any, we introduce.

Generally, standard image fitting proposals are constructed as a mixture of proposals with varying scales. This is because, when proposals are defined on a single scale, the samples they propose are commonly very similar in terms of their step size. Mixing varying scales introduces an additional feature of the proposals to make larger steps when possible or smaller in other cases. This process is especially important during the initial phase when it is possible to make bigger steps and by doing so move towards the target distribution faster.

Camera Proposals

Since we know an actual camera parameters for our D415 camera, which are fixed unless we change resolution, there is no need to infer them from the image. Therefore, we fix the camera parameters and make sure to omit proposals that try to modify this parameter. Such proposals are *DistanceProposal* and *ScalingProposal* both of which are part of *PoseProposal*.

Shape & Pose Proposals

Even though we already have a decent shape and pose parameters after the shape fitting, we still have to include these two proposals in the color fitting module. The change we make though is that we drop the highest step size proposals inside the mixture proposal and lower the rate of how often they propose. This is because during the shape fitting it is not possible to determine the key facial characteristics like eyebrows, lip borders, eye lines, etc. and if we do not allow slight shape and pose changes we will encounter anomalies like double eyebrows. The rest of the details are kept the same as standard fitting.

Evaluators

Besides default image evaluators proposed by [2], we make use of point cloud evaluator used during the shape fitting part (Equation 3.3), in addition to 3D landmark evaluator to filter our samples that are not making through it.

4
Results

To understand the process better, we have evaluated the shape fitting and the color fitting module separately. The dataset we evaluate is exclusively based on captures made by the D415 camera, it includes images with varying faces, poses and illumination conditions. Around 80% of the dataset consists of frontal shots, the rest of the images are varying between $\pm 30^{\circ}$ yaw and roll rotation angles. To replicate the real world scenarios as close as possible, we did not set up a specific camera rig for capturing procedure, rather, photos are taken arbitrarily. To compare our results to the previous results we have also executed a novel 2D fitting [2] over the same dataset. Unfortunately, we were unable to run [7] project and therefore the values that we list are directly taken from the report — those values are not tested on our dataset. Before we present our results, we shall define the evaluation metrics that we employ to assess the performance and quality of the final reconstructions.

4.1 Evaluation Metrics

The most obvious and straightforward way to evaluate the shape quality of 3D object reconstruction is to measure the distance between the target object and the reconstructed one. Thanks to the target mesh we have obtained from the camera and the reconstructed mesh produced after the 3DMM fitting, we have a possibility to work with 3D mesh objects and evaluate the distance between them. The more robust and meaningful distance measure is the average mesh distance. It measures the distance from every point on one mesh to the closest point on another mesh and averages them. This kind of evaluation metrics are known to work well when two objects that are being evaluated are symmetric. Unfortunately, in our case, the target mesh and the reconstructed meshes are far from symmetric and quite a lot of outlier points are observed. Therefore, to make the returned value reasonable we have made a small adjustment. We employ the simple boundary point ignorance technique, which ignores the distance entry for every point on one mesh that has its closest point located on the boundary of another mesh. This will ignore obvious outliers and give us a quantitative measure of how far our shape reconstruction is from the real shape. The method is robust enough and works well even when the mesh points are distributed non-evenly. When it comes to image evaluation, the best way to evaluate it is using human

perception. Therefore, we make sure to provide a set of sample reconstructions alongside numerical measurements.

4.2 Shape Fitting Module Evaluation

We first start listing results of shape fitting module and evaluating how well reconstructed shape fits to the target mesh. Table 4.1 shows the average mesh reconstruction accuracy in millimeters after 1000 shape fitting iterations.

Evaluated Against	#Iterations	Avg. Mesh Distance [mm]
Target Mesh	1000	**1.18**
Ground Truth	1000	**1.47**

Table 4.1: Shape fitting accuracy in mm.

The value for the ground truth alignment is obtained by aligning the best shape reconstruction over the ground truth using the method described previously, however, we do not perform any explicit alignment procedure for the shape fit and the target mesh because the shape fitting itself is an alignment problem. The latter also explains the difference between those two evaluations - shape fit is constantly getting evaluated against the target mesh so it is expected that the average value would be lower between the target mesh and the reconstructed mesh than the average value between the ground truth and the reconstructed mesh.

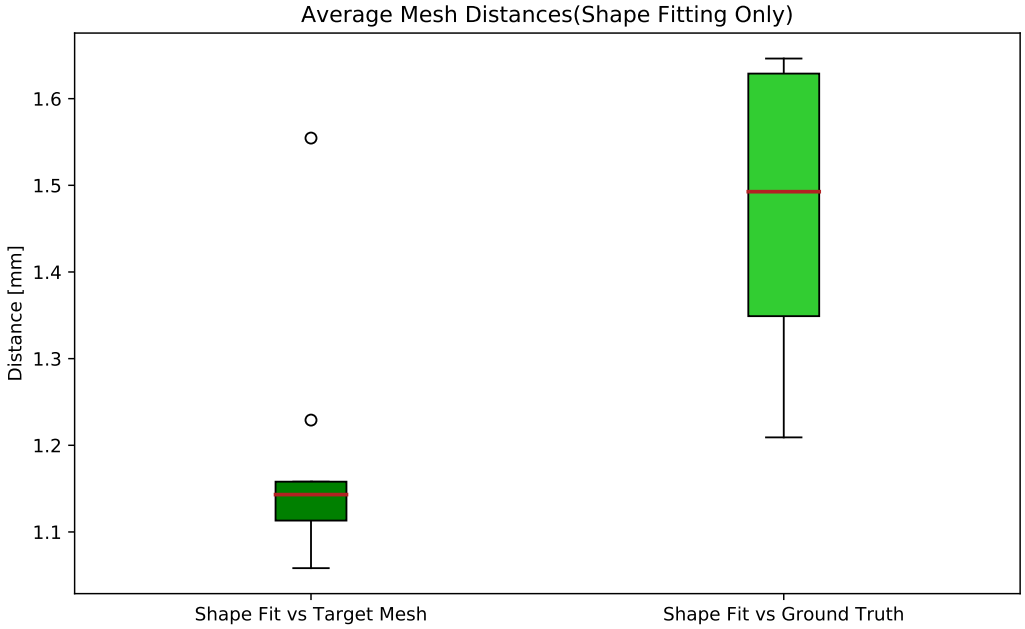


Figure 4.1: The first column of this box plot shows the average mesh distance between the best shape fit and the target mesh, the second column shows the distance between the best shape fit and the ground truth. The red line indicates the median (Table 4.1).

Figure 4.1 shows the distribution of the entire dataset. Outlier points in the first column are produced by $\pm 30^{\circ}$ yaw rotated shots. As we have emphasized previously our method performs noticeably better in the frontal shots and the worst in shots where the yaw rotation is introduced. The reason behind this goes back to the landmark detection procedure. Recall, we detect landmarks onto a color image and then de-project landmark pixels to points. The problem occurs due to the fact that whenever the Dlib landmark detection library detects a face, it provides the full set of landmarks without respecting the pose. If the part of the face is not visible, landmarks that are located onto the occluded side of the face are still present. Those landmarks are first not very accurate and secondly, they are getting grouped into one area and the distance between landmark points does not correspond to the actual distance. This either breaks our shape fitting part or produces bad reconstructions, because of this constraint majority of our dataset samples are frontal shots. In Figure 4.2 we show two sample shape reconstruction after 1000 iterations.

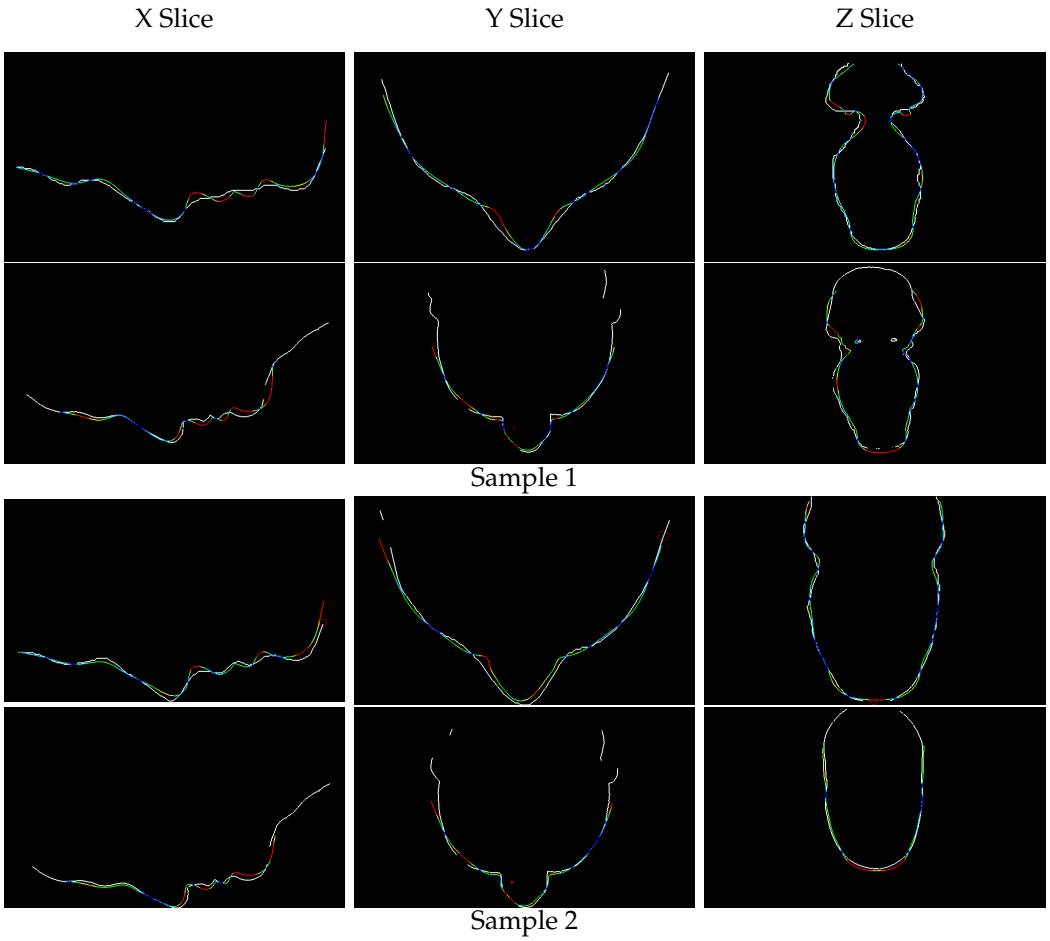


Figure 4.2: The first row of these two sample fits shows the target and the best shape fit alignment, the second ground truth and the best shape fit. In both cases, the colored line represents the best shape fit instance, where the blue color indicates 0.0 mm distance between the points and the red indicates 3.0 mm+ distance. Both shots are frontal shots to make the slice visualization easier to understand.

As we can clearly see from the sample slices above, the highest errors (colored in red) are being accumulated in places where there are a lot of curvature or acute angles. For example the lip details or nose angle. This is because the point cloud itself does not capture this kind of details, what we have instead is a rather smooth and generic shape without detailed facial features.

4.3 Full Fitting Evaluation (Shape Fitting + Color Fitting)

Once the shape fitting module ends its cycle the final reconstruction and other necessary parameters are then being passed to the color fitting module. It starts another proposal-and-verify cycle to approximate the color values from the target image. Figure 4.3 shows the comparison of the shape reconstruction quality after the shape fitting and after the full fitting.

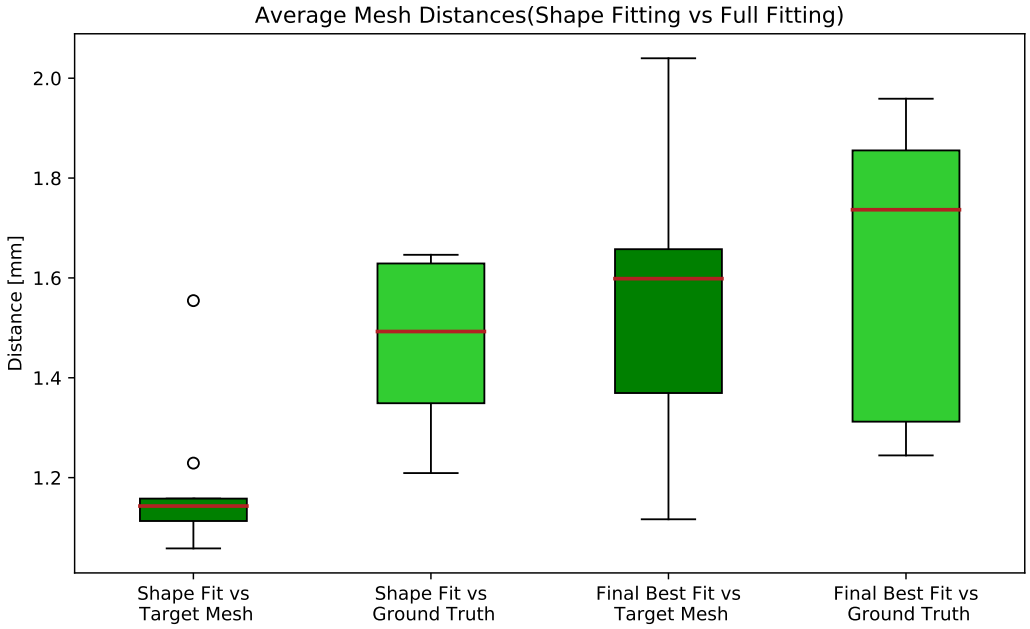


Figure 4.3: The average mesh distances after the shape fitting only (1st and 2nd column) and after the full fitting (3rd and 4th column). The first and third column shows the average mesh distances between the reconstructed mesh and the target mesh, the second and fourth column shows the average mesh distances between the reconstructed mesh and the ground truth.

As stated previously, if we do not add shape or pose proposals to the color fitting module, obviously the average distance will not change. While numerically this would be perfect, visually we would not get satisfactory results. Because of this, we had to make a compromise and include shape and pose parameter proposals into the color fitting module. Even though we evaluate landmark positions and mirror the point cloud evaluator from the shape fitting module, shape still gets worst in comparison to the shape fitting results.

Alongside our fitting module, we have evaluated a novel 2D fitting approach[23]. To do so, we took our entire dataset images and obtained landmarks for each target image using the *landmarks-clicker*[24] tool. We did not make any changes to the standard fitting module, we simply pass our dataset images with corresponding landmarks to the script, and in the end, evaluate the final reconstructions. Table 4.2 shows the result of the final average shape reconstruction accuracy calculated against the ground truth.

Landmarks	Shape Information	Avg. Mesh Distance [mm]
2D	No	2.21 [2]
3D	Yes(Depth)	1.94 [7]*
3D	**Yes(Point Cloud)**	**1.61**

Table 4.2: The final shape fitting accuracy in mm obtained by aligning the final 3D reconstructions over the ground truth. *BFM dataset evaluation value; taken directly from the paper to have a rough idea how does our method compare to it — we did not manage to run the module to test it on our dataset.

Table 4.3 shows the average distance between the final shape fit and the target mesh. This value is obtained by aligning the final reconstruction and the corresponding target mesh.

Landmarks	Shape Information	Avg. Mesh Distance [mm]
2D	No	2.94 [2]
3D	**Yes(Point Cloud)**	**1.56**

Table 4.3: Final shape fitting accuracy in mm obtained by aligning the final 3D reconstructions over the target mesh.

Figure 4.4 shows the visual comparison of the target image, rendered model fit of our method and standard fitting method. Figure 4.5 shows the same results as Figure 4.4 but the final fit is overlaid to the target image to make the comparison easier. We argue that the reconstructions that are produced by our method are capturing a bit more details and facial characteristics, whereas the standard fitting results are more generic while still maintaining close-to-target state. We think that this is because our shape reconstruction quality is noticeably better and hence after applying the texture and illumination we get a bit more close to the target results. We once again emphasize and more closely observe(rows 3 and 4) the difficulties our method experiences when working with poses that have yaw rotation applied.

The box plot in Figure 4.6 shows the distribution of the average mesh distances of the entire dataset. Where the first and second columns indicate our final shape reconstruction accuracies, and the third and fourth column indicate the performance of the standard fitting method. Unlike results that we have listed for the shape fitting evaluation, this time, the final fits are being aligned both to the target mesh and the ground truth mesh.

[23] Basel Face Registration Pipeline — https://github.com/unibas-gravis/basel-face-pipeline/
[24] Landmarks Clicker — https://github.com/unibas-gravis/landmarks-clicker/

Figure 4.4: A visual comparison of our method and the standard 2D fitting results.

Figure 4.5: A visual comparison of the same fits from Figure 4.4 overlaid to the target image.

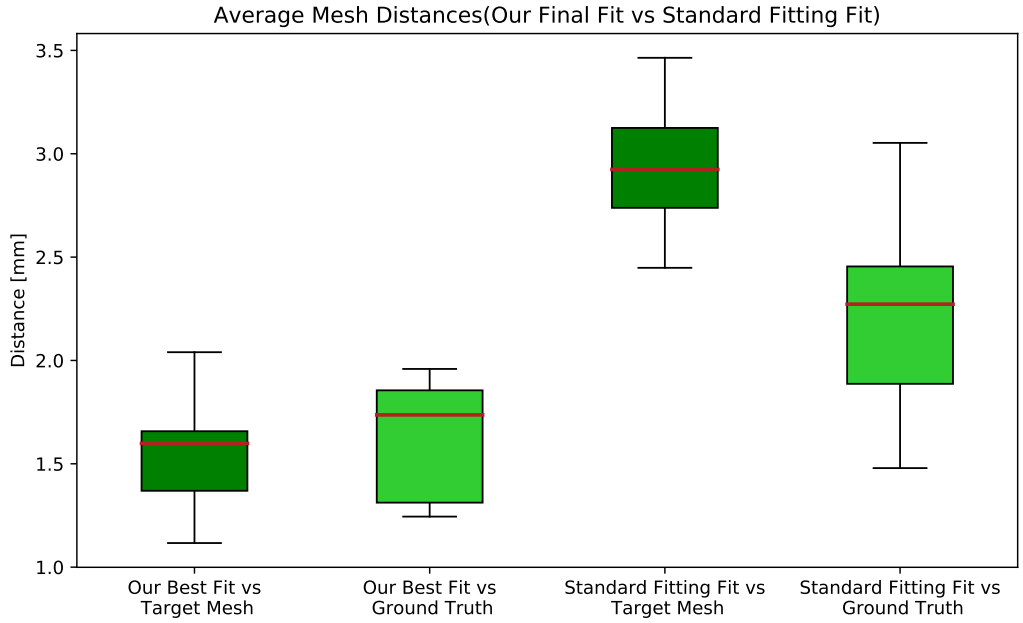


Figure 4.6: The average mesh distance comparison of our method(1st and 2nd column) and the standard fitting method(3rd and 4th column).

4.4 Execution Performance

In order for our system to be useful for demo applications the execution time of at least one module needs to be reduced down to seconds. While it was not possible for us to achieve a decent full fitting performance time-wise. We were able to execute shape fitting module within the reasonable time limits (bellow 30 seconds). We show average execution times for each module in Table 4.4.

Module	#Iterations	Avg. Execution time
Shape Fitting	500	$\sim$ 25 seconds
Shape Fitting	1000	$\sim$ 45 seconds
Color Fitting	5000	$\sim$ 10 minute
Color Fitting	10000	$\sim$ 18 minute

Table 4.4: Timing the execution times for shape fitting and color fitting module.

Web-Service Integration

Based on shown results, the shape fitting does have the potential of being used for demonstration purposes and it is possible to integrate into the Scalismo Face Morpher web-service. Obviously, listed times are directly dependent on sampling iterations, increasing them will increase the execution time and decreasing will reduce time alongside with quality of the reconstruction. To be able to integrate our proposed shape fitting procedure into the cur-

rent web client we have implemented a single intermediate Thrift server which will connect the already existing Face Morpher web module and proposed shape fitting module to each other. The flow of this system can be seen in Figure 4.7.

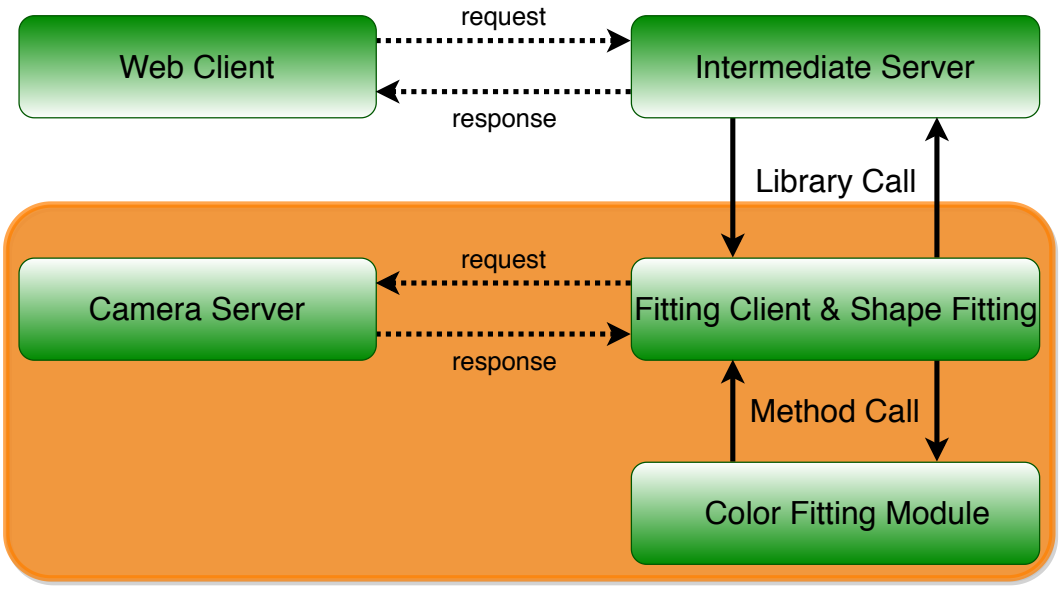


Figure 4.7: Connecting web client to our proposed fitting modules using the single intermediate server. Everything that is inside the orange box is part of our fitting pipeline.

Whenever a user triggers the starting sequence, the web client sends the request to the intermediate server. The server, that has our fitting pipeline embedded into it as a library, then calls a method from our pipeline which triggers the fitting client to send a request to the camera module in order to obtain the data and start the shape fitting procedure. Due to these multiple client-server interactions, the overall execution time will increase slightly but it will still stay in an acceptable region.

5
Conclusion

We have successfully implemented the complete, reliable and coherent system for more efficient face fitting pipeline. We have taken a modularized approach to tackle this problem. By separating data acquisition (including the module which handles all the camera related processes), shape fitting and color fitting modules from each other, we have made the system useful for any future application that will either make use of one of the modules or all of them together. We have introduced an alternative, more efficient way of a standard 3DMM face fitting by relying on a point cloud information obtained from the consumer depth camera. We have shown that our method produces especially good shape reconstructions in comparison to previous approaches. This was achieved by fitting the 3D Morphable model to the point cloud first to obtain an accurate shape and pose parameters. Those parameter sets are very important prerequisites for the color fitting module and directly influence the final reconstructions. The parameter set that is produced by the shape fitting module, is getting utilized by the color fitting module to obtain final face reconstruction. Based on more accurate shape reconstruction, our method also attempts to capture more specific details of the target face in the image.

We have presented an evaluation of our proposed shape and color fitting modules, as well as, comparison of our final results and the previous approaches. Our final reconstructions are noticeably better than previous methods. Especially stands out the shape reconstruction quality, which we measure based on average mesh distances. This was expected since, out method has an actual shape information to rely on during the fitting process. When comparing our shape and color fitting modules we have observed a slight decrease in the shape quality. This is because of the color fitting module requiring additional adjustments throughout the process to correctly infer the color information from the image. The decrease in shape quality also is due to increased difficulty of the fitting problem during the color fitting phase. Despite this decrease, final reconstructions that our method produces are still more accurate than previous method results. Our method performs noticeably better in full-frontal shots and suffers mainly when yaw rotation is introduced. This is due to the landmark detection algorithm onto the image, which provides all landmarks whether the landmark point is visible or not. This is not necessarily optimal for our application

because pixels that get de-projected incorrectly might cause problems for shape fitting.

We have also presented a demo implementation of our shape fitting module into the already existing Scalismo Face Morpher web service. Since our system is module based, this kind of implementations are straightforward and can be easily preformed for any future applications.

6
Future Work

In this chapter, we will be discussing potential improvements to our approach that were not possible to execute in our project due to time constraints.

6.1 Using Multiple Camera Setup

Figure 6.1: Multi-camera configuration examples. Source: Intel

As we have seen from the previous images we are only working on the face region instead of full head region (including ears). One of the reasons we opted to do so is that one single camera view is not able to capture some parts of the face that are occluded. However, if multiple camera angles will be used this problem will no longer persist and it would be possible to work with a full face model instead of the reduced model of only the face region. This would also improve the pose estimation process and will most likely produce

more accurate shape reconstructions. Thus, we think that it is worthwhile to investigate a multiple camera setup (Figure 6.1) and obtain a more accurate and well-structured mesh from a point cloud. The Intel RealSense SDK does provide support for multi-camera setup out of the box[22]. Especially useful would be, the inward facing camera configuration from Figure 6.1. It will capture the target object from four (or more) different angles and will likely produce full 360° scan.

Additionally, tweaking camera parameters and performing post-processing steps as per [19] could also improve the quality of the target mesh constructed from a point cloud. As we have mentioned in Section 3, there are a lot of configuration parameters available which we did not investigate in detail due to time limitations. Denoising the depth data is one of the obvious optimization steps that would be beneficial for the fitting pipeline[23]. As discussed previously, a point cloud that we obtain from the camera by default is very noisy and lacks the important details and facial characteristics. The higher quality point cloud would most likely produce final reconstructions that are capturing more details of the target face.

6.2 Better Landmark Filtering

The main reason for our method being very sensitive to the yaw rotation angle is due to Dlib landmarks not being filtered according to the pose. One possible fix of this issue would be to control the Dlib face box during the landmark detection procedure and by doing so only take into account landmark points that are actually visible. Recall, Dlib estimates a location of the landmarks that are not visible in the image and returns a full set of landmarks that are then being de-projected to obtain 3D landmark points.

Another work-around would be to assign a higher uncertainty to the landmarks that are being estimated. In our implementation, we have used higher landmark uncertainty for chin landmarks than for the rest of the landmarks. However, we have no way of dynamically detecting the pose variation in the image to ignore altogether or assign higher uncertainty to a subset of landmarks that latter cause problems for the fitting pipeline. We think this is a very important feature update for the entire module since both shape fitting and color fitting module crucially depend on those landmarks.

An alternative way to deal with this issue would be to implement a native landmark detection method instead of relying on 3rd party pixel landmark detection method and then de-projection which introduces an additional error.

6.3 Multi-Resolution Fitting

Unfortunately, due to time constraints, we were not able to deeply research the color fitting module and its speed up capabilities. Because of this, the web service integration was only possible for our shape fitting module. Therefore, we state this challenge as a future work, one possible solution of which could be a multi-resolution fitting. As the name

suggests, the basic idea is to perform the color fitting in multiple stages, starting from a low-resolution image and model, and gradually increase the resolution and the complexity of the problem [24–26].

Bibliography

[1] CHAPTER 4 - 3D MORPHABLE FACE MODEL, A UNIFIED APPROACH FOR ANALYSIS AND SYNTHESIS OF IMAGES. In Zhao, W. and Chellappa, R., editors, *Face Processing*, pages 127 – 158. Academic Press, Burlington (2006). URL http://www.sciencedirect.com/science/article/pii/B9780120884520500054.

[2] Schönborn, S., Egger, B., Morel-Forster, A., and Vetter, T. Markov Chain Monte Carlo for Automated Face Image Analysis. *International Journal of Computer Vision*, 123(2):160–183 (2017). URL https://doi.org/10.1007/s11263-016-0967-5.

[3] Schönborn, S., Forster, A., Egger, B., and Vetter, T. A Monte Carlo Strategy to Integrate Detection and Model-Based Face Analysis. In Weickert, J., Hein, M., and Schiele, B., editors, *Pattern Recognition*, pages 101–110. Springer Berlin Heidelberg, Berlin, Heidelberg (2013).

[4] Guo, Y., Zhang, J., Cai, J., Jiang, B., and Zheng, J. CNN-Based Real-Time Dense Face Reconstruction with Inverse-Rendered Photo-Realistic Face Images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 41(6):1294–1307 (2019). URL http://dx.doi.org/10.1109/tpami.2018.2837742.

[5] Tu, X., Zhao, J., Jiang, Z., Luo, Y., Xie, M., Zhao, Y., He, L., Ma, Z., and Feng, J. Joint 3D Face Reconstruction and Dense Face Alignment from A Single Image with 2D-Assisted Self-Supervised Learning. *arXiv preprint arXiv:1903.09359* (2019).

[6] Blanz, V. and Vetter, T. 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. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA (1999). URL http://dx.doi.org/10.1145/311535.311556.

[7] Betschard, C. Fitting a 3D Morphable Face Model to Captures from Consumer Depth Cameras. *University of Basel, Department of Mathematics and Computer Science, Master's Thesis* (2016).

[8] Schönborn, S. Markov Chain Monte Carlo for Integrated Face Image Analysis. *University of Basel, Department of Mathematics and Computer Science, PhD Thesis* (2014).

[9] Projection in RealSense SDK 2.0. https://dev.intelrealsense.com/docs/projection-in-intel-realsense-sdk-20#section-point-coordinates. Access Date: 2019-10-01.

[10] Gerig, T., Morel-Forster, A., Blumer, C., Egger, B., Luthi, M., Schoenborn, S., and Vetter, T. Morphable Face Models - An Open Framework. In *2018 13th IEEE International Conference on Automatic Face Gesture Recognition (FG 2018)*, pages 75–82 (2018).

[11] Egger, B., Schönborn, S., Blumer, C., and Vetter, T. Chapter 5 - Probabilistic Morphable Models. In Zheng, G., Li, S., and Székely, G., editors, *Statistical Shape and Deformation Analysis*, pages 115 – 135. Academic Press (2017). URL http://www.sciencedirect.com/science/article/pii/B9780128104934000067.

[12] Albrecht, T., Lüthi, M., Gerig, T., and Vetter, T. Posterior shape models. *Medical Image Analysis*, 17(8):959 – 973 (2013). URL http://www.sciencedirect.com/science/article/pii/S1361841513000844.

[13] Lüthi, M., Gerig, T., Jud, C., and Vetter, T. Gaussian Process Morphable Models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(8):1860–1873 (2018).

[14] Schönborn, S., Egger, B., Forster, A., and Vetter, T. Background Modeling for Generative Image Models. *Comput. Vis. Image Underst.*, 136(C):117–127 (2015). URL http://dx.doi.org/10.1016/j.cviu.2015.01.008.

[15] Dorodnicov, S. The basics of stereo depth vision. URL https://www.intelrealsense.com/stereo-depth-vision-basics/. Access Date: 2019-10-01.

[16] Keselman, L., Woodfill, J. I., Grunnet-Jepsen, A., and Bhowmik, A. Intel RealSense Stereoscopic Depth Cameras. *CoRR*, abs/1705.05548 (2017). URL http://arxiv.org/abs/1705.05548.

[17] Hartley, R. and Zisserman, A. *Multiple View Geometry in Computer Vision*. Cambridge University Press, New York, NY, USA, 2 edition (2003).

[18] Calibration Tools User Guide for Intel® RealSense™ D400 Series. URL https://dev.intelrealsense.com/docs/intel-realsensetm-d400-series-calibration-tools-user-guide. Access Date: 2019-10-01.

[19] Grunnet-Jepsen, A., Sweetser, J. N., and Woodfill, J. Best-Known-Methods for Tuning Intel® RealSense™ D400 Depth Cameras for Best Performance. URL https://dev.intelrealsense.com/docs/tuning-depth-cameras-for-best-performance. Access Date: 2019-10-01.

[20] Lorusso, A., Eggert, D. W., and Fisher, R. B. A Comparison of Four Algorithms for Estimating 3-D Rigid Transformations. In *Proceedings of the 1995 British Conference on Machine Vision (Vol. 1)*, BMVC '95, pages 237–246. BMVA Press, Surrey, UK, UK (1995). URL http://dl.acm.org/citation.cfm?id=236190.236213.

[21] Scalismo Tutorials. URL https://scalismo.org/tutorials. Access Date: 2019-10-01.

[22] Grunnet-Jepsen, A., Winer, P., Takagi, A., Sweetser, J., Zhao, K., Khuong, T., Nie, D., and Woodfill, J. Using the Intel® RealSense™ Depth cameras D4xx in Multi-Camera Configurations. URL https://dev.intelrealsense.com/docs/multiple-depth-cameras-configuration. Access Date: 2019-10-01.

[23] Ferrari., C., Berretti., S., Pala., P., and Bimbo., A. D. 3D Face Reconstruction from RGB-D Data by Morphable Model to Point Cloud Dense Fitting. In *Proceedings of the 8th International Conference on Pattern Recognition Applications and Methods - Volume 1: ICPRAM,*, pages 728–735. INSTICC, SciTePress (2019).

[24] Liu, C., Shum, H.-Y., and Zhang, C. Hierarchical Shape Modeling for Automatic Face Localization. In Heyden, A., Sparr, G., Nielsen, M., and Johansen, P., editors, *Computer Vision — ECCV 2002*, pages 687–703. Springer Berlin Heidelberg, Berlin, Heidelberg (2002).

[25] Huber, P., Hu, G., Tena, R., Mortazavian, P., Koppen, P., Christmas, W., Ratsch, M., and Kittler, J. A Multiresolution 3D Morphable Face Model and Fitting Framework. In *11th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications* (2016). URL http://epubs.surrey.ac.uk/809478/. Paper accepted for presentation at 11th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications, 27-29 February 2016. Full text may be available at a later date.

[26] Huber., P., Hu., G., Tena., R., Mortazavian., P., Koppen., W. P., Christmas., W. J., Rätsch., M., and Kittler., J. A Multiresolution 3D Morphable Face Model and Fitting Framework. In *Proceedings of the 11th Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications - Volume 4: VISAPP, (VISIGRAPP 2016)*, pages 79–86. INSTICC, SciTePress (2016).

A

Thrift API Scheme

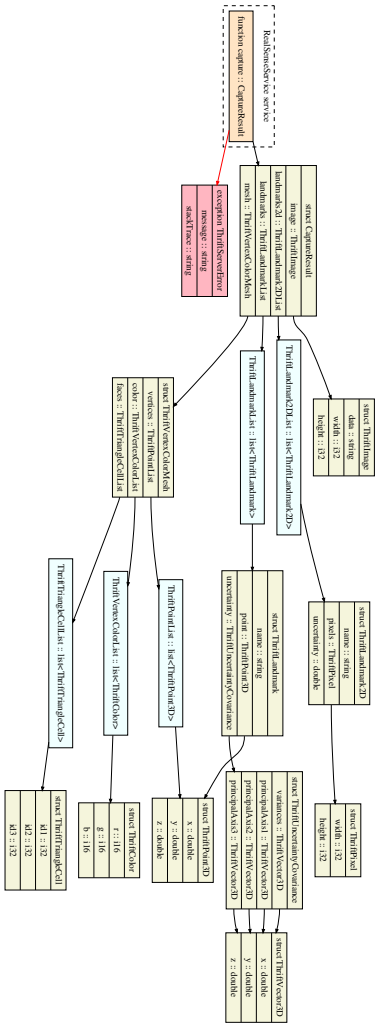


Figure A.1: The API scheme we have implemented to obtain the required data.

Declaration on Scientific Integrity
Erklärung zur wissenschaftlichen Redlichkeit

includes Declaration on Plagiarism and Fraud
beinhaltet Erklärung zu Plagiat und Betrug

Author — Autor
Giorgi Grigalashvili

Matriculation number — Matrikelnummer
12-059-754

Title of work — Titel der Arbeit
Efficient 3D Morphable Model Face Fitting using Depth Sensing Technologies

Type of work — Typ der Arbeit
Master's Thesis

Declaration — Erklärung
I hereby declare that this submission is my own work and that I have fully acknowledged the assistance received in completing this work and that it contains no material that has not been formally acknowledged. I have mentioned all source materials used and have cited these in accordance with recognised scientific rules.

Hiermit erkläre ich, dass mir bei der Abfassung dieser Arbeit nur die darin angegebene Hilfe zuteil wurde und dass ich sie nur mit den in der Arbeit angegebenen Hilfsmitteln verfasst habe. Ich habe sämtliche verwendeten Quellen erwähnt und gemäss anerkannten wissenschaftlichen Regeln zitiert.

Basel, 13.11.2019

Signature — Unterschrift