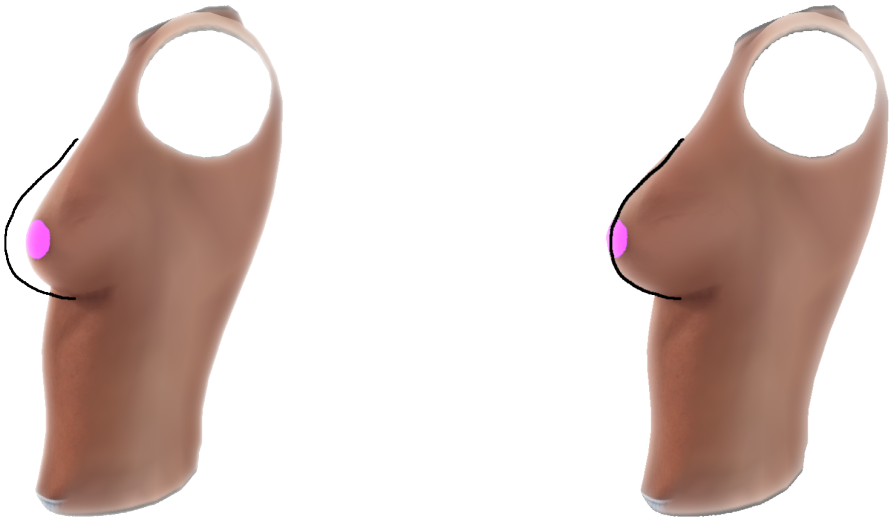
Cross-platform Real-time Visualization and Editing Interface of Breast 3D meshes



Luca Sichi

Bachelor Thesis
June 2023

Dr. Endri Dibra, Dr. Beren Kaul, Prof. Dr. Markus Gross

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

cgl
computer graphics laboratory

Abstract

This thesis employs computer graphics and computer vision techniques to capture images of the torso, enabling users to modify the appearance of the breasts and evaluate the potential outcomes of a surgical procedure. The objective is to develop a unified codebase capable of running on multiple platforms. We further developed two methods that allow the user to modify and edit the breasts, one method includes sliders, the other one implements a correction UI, that allows the user to draw the desired breast shape. To achieve multi-platform capabilities we leverage the Flutter framework, which allows for easy cross-platform compilation for our primary target platforms, the web and Android.

Zusammenfassung

In dieser Arbeit werden Computergrafik- und Computer-Vision-Techniken eingesetzt, um Bilder des Oberkörpers zu erfassen, die es dem Benutzer ermöglichen, das Aussehen der Brüste zu verändern und die möglichen Ergebnisse eines chirurgischen Eingriffs zu bewerten. Ziel ist es, eine einheitliche Codebasis zu entwickeln, die auf mehreren Plattformen laufen kann. Darüber hinaus haben wir zwei Methoden entwickelt, die es dem Benutzer ermöglichen, die Brüste zu modifizieren und zu bearbeiten; eine Methode umfasst Schieberegler, die andere implementiert eine Korrektur-UI, die es dem Benutzer ermöglicht, die gewünschte Brustform zu zeichnen. Um Multiplattformfähigkeit zu erreichen, nutzen wir das Flutter-Framework, das eine einfache plattformübergreifende Kompilierung ermöglicht für unsere primären Zielplattformen, das Web und Android.

Contents

List of Figures

List of Tables

1

Introduction

Visualization of a potential outcome is a crucial aspect of any surgical procedure, especially if the procedure is cosmetic in nature. Through the use of 3D models, surgeons can provide their patients with a visual representation of the potential outcome of a surgical procedure. This thesis is done in collaboration with Arbrea Labs, a company that specializes in the development of software solutions for plastic surgeons. Arbrea Labs has developed a pipeline that allows surgeons to create 3D models of their patients' breasts and torso. The dataset of 3D models used in this thesis is provided by Arbrea Labs.

The main goal of this thesis is to develop a cross-platform application that allows users to modify the appearance of the breasts and visualize the potential outcomes of a surgical procedure. We want to implement a way to change the patient's breast with the use of simple UI elements like sliders, but also with a more sophisticated method, which allows the user to change the breast by drawing a line where he wants the breast to be.

The aim is to establish a unified codebase capable of operating on various platforms, with a particular focus on the web and Android.

This thesis is structured in the following chapters:

- Chapter 2 gives an overview of the background and some related work.
- Chapter 3 discusses in-depth all the methods used in this thesis.
- Chapter 4 gives an experimental evaluation and compares the methods used in this thesis.
- Chapter 5 summarizes the results and gives an outlook on future work, solving some of the problems that occurred during the development of this thesis.

2

Background and Related Work

In this chapter we will discuss the background of this thesis and related work. We give a brief introduction to the topic of 3D models and how we can modify them. We will also discuss the Flutter framework, which we use to build our application.

2.1. Displaying 3D Models

A central aspect of this thesis revolves around the presentation of 3D models. Given the intended deployment of the application across multiple platforms, the implementation requires a cross-platform approach. To address this requirement, we use the Cube Flutter package, which we have adapted to our specific needs. The Cube package uses Wavefront's obj data format to read 3D models.

This data format is a simple text-based format that stores 3D models in a human-readable form. In our use case it saves vertex coordinates, vertex normals, vertex texture and faces. Since we later want to change the shape of the 3D model the vertex coordinates are most interesting. In section 5.2 we will discuss possible future work, in which we need access to the other model attributes for finding a silhouette.

2.2. Modification of 3D Models

As mentioned above, we only want to change the shape of the 3D model. This corresponds to changing the vertex coordinates. This change then automatically changes the faces, and thus the shape of the 3D model. To facilitate the alteration of the 3D models, an adapted approach is employed, drawing inspiration from the methodology presented in the scholarly work by

[RSM08]

We will discuss our approach in extensively in section 3.4.

2.3. Flutter Framework

Since our goal is to write a codebase for multiple target platforms, we decided to use the Flutter framework, which allows compilation to multiple target platforms. In our use case our primary focus is the web and then Android. In theory Flutter supports multiple other platforms, like iOS, Windows and so on.

Flutter uses as programming language Dart, which is an object-oriented language with C-style syntax.

Flutter also allows us to write platform specific code. For the web this is JavaScript, for Android this is Java. Although we didn't use this functionality in this work, this might be a desired for any future work.

Another important aspect of Flutter is the widget system. Flutter uses a widget system to build the user interface and to handle user input. This means that we need to build our UI once, and it will look similar on all platforms.

Flutter further comes with a package manager called pub. Pub allows us to use third party packages by simply adding them to the pubspec.yaml file. This allows us to get things like third party widgets, like for example a 3D model viewer, or libraries for mathematical operations that allow us to do quick matrix multiplication.

3

Methods

This chapter provides a detailed description of our approach, outlining the sequential steps we undertook to accomplish our goals. We present a thorough analysis of the challenges we encountered during the development process and offer comprehensive explanations for the decisions made in our approach.

We begin by explaining the rationale behind our chosen methodology and highlight the specific objectives we aimed to achieve. Furthermore, we then proceed to discuss the individual steps involved in our approach, providing a comprehensive overview of each stage's purpose and significance.

Throughout this chapter, we delve into the problems we encountered during the implementation of our approach. We discuss the reasoning behind the strategies adopted to address these challenges, providing insights into the decision-making process.

3.1. Image Acquisition

The initial phase involves the acquisition of patient data. In our case this is a triad of torso images. These images consist of a frontal view, a left lateral view, and a right lateral view. They provide comprehensive visual information required for the analysis and augmentation processes. Given our objective of maximizing the versatility of our application, we aim to enable the utilization of images from diverse sources. This means providing users with the flexibility to capture patient images using the device's camera or select existing images from the gallery. By incorporating such functionality, our application adapts to varying user preferences and facilitates seamless integration with different image acquisition methods, enhancing the overall usability and accessibility of the app.

Within the Flutter framework, a big range of image acquisition widgets is available, includ-

ing the ImagePicker and Camera widgets. The ImagePicker widget empowers users to select an image from the gallery of the device, while the Camera widget allows for real-time image capture utilizing the device's integrated camera or an external camera like a webcam. These widgets serve as essential components within the Flutter ecosystem, offering intuitive and efficient means of acquiring images from distinct sources.

At this stage, we encounter the need to write platform-specific code. The web application, being browser-based, presents a hurdle in terms of accessing the device's memory, which poses difficulties. In contrast, the Android platform offers seamless access to the device's memory without encountering any obstacles. This distinction underscores the need for devising a solution that accommodates the limitations imposed by web-based environments, ensuring compatibility and functionality across platforms.

3.1.1. Image Acquisition on Android

Since the Android platform offers direct access to the device's memory, the implementation of the image acquisition process is relatively straightforward. Utilizing the Camera widget, we facilitate real-time image capture, subsequently storing the captured images within the device's memory.

Given the requirements of Arbrea's established pipeline, which necessitates three torso images encompassing a frontal view, a right-side view, and a left-side view, the user is instructed to capture and retain three distinct pictures. We then save these images on the device's memory and subsequently pass them to the next stage of the pipeline.

3.1.2. Image Acquisition on Web

On the web platform, our implementation deviates slightly from the Android counterpart due to the restrictions on accessing the device's memory. To circumvent these limitations, we rely on methods inherent to web development practices. Specifically, images captured using the camera are stored within the browser environment as Binary Large Objects (BLOBs). This storage mechanism allows for efficient management and retrieval of image data, enabling seamless processing and integration within the web-based application.

A Blob represents a file-like object of immutable, raw data. Blobs represent data that isn't necessarily in a JavaScript-native format.

In our web-based implementation, the Camera widget generates Binary Large Objects (BLOBs) automatically for the captured images, providing us with URLs instead of conventional file paths for subsequent referencing. Similar to the Android implementation, we guide the user to capture three torso images. Given that a device may possess multiple cameras or webcams, we incorporate functionality that enables the user to select their preferred camera. The Flutter framework proves invaluable in this regard, as it handles the management of diverse camera and webcam options available on the device, ensuring seamless compatibility and optimal utilization.

In addition to capturing images through the device's camera, we offer users the alternative option of selecting images from their device's gallery. This feature proves particularly useful

for users operating desktop computers, where capturing the necessary images with a webcam can be challenging and inconvenient. Given the recurring constraint of limited access to the device's memory, we rely on the Flutter framework to present a viable solution. Leveraging the ImagePicker widget within Flutter, users can seamlessly select or drag and drop images from their devices and upload them to the web application. This functionality parallels the familiar concept of file uploads on websites, a ubiquitous feature found across numerous online platforms.

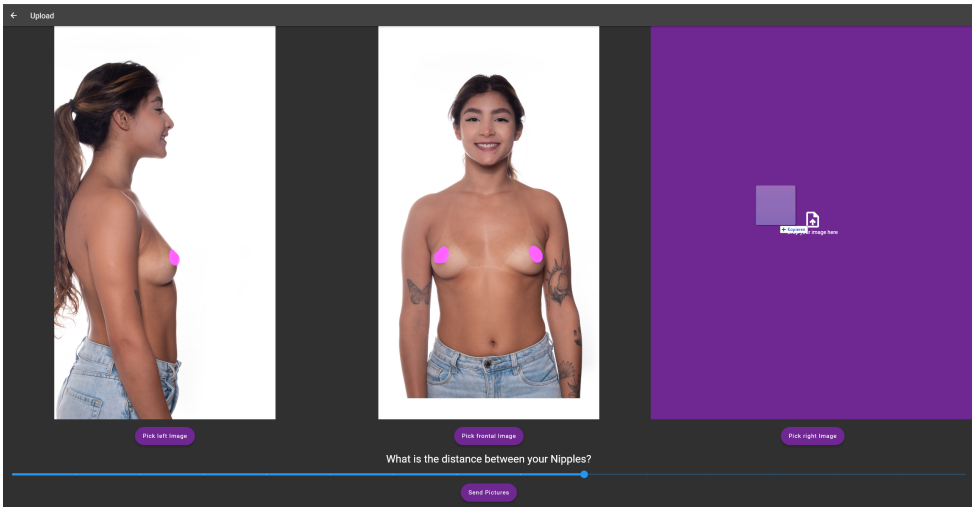


Figure 3.1.: *Upload Screen*

3.2. Communication with the backend Pipeline

At this juncture, we find ourselves faced with the task of obtaining a 3D model from the captured torso images of the patient. To accomplish this, we adopt the existing pipeline developed by Arbrea Labs, which serves as the foundation for generating the requisite 3D model. However, to initiate this process, connecting to a dedicated server hosting the pipeline becomes imperative. Subsequently, the client transmits the acquired images alongside additional pertinent data to the server. The server, in turn, calls Arbrea's pipeline with the received images, initiating the creation of the 3D model. Once generated, the resulting model is transmitted back to the client, where it is stored for subsequent stages of processing, augmentation, and visualization.

In light of the straightforward requirements for the server component, we have opted to utilize a basic Python server that leverages Python's built-in http.server implementation. This minimalistic server implementation provides the necessary functionality to handle HTTP requests and responses without the need for complex external dependencies.

3.2.1. Python

Within the server-side architecture of our application, the primary objective is to receive three images from the client. Furthermore, considering the pipeline's reliance on the distance be-

tween the patient's nipples, we require the client to transmit this essential information alongside a unique identifier. When a POST request is received, the server anticipates a JSON object comprising the three images, with the frontal image listed first, followed by the right-side image, and finally the left-side image. The JSON object also encompasses the distance between the nipples and the unique identifier.

Subsequently, the server proceeds to create a dedicated directory named after the unique identifier, serving as a repository for storing the received images. Given that the incoming images are not in a standard image file format, a conversion process becomes necessary. To accomplish this, we employ the Python Imaging Library (PIL), a powerful image processing library. By leveraging the functionalities offered by PIL, we can effectively convert the received image data into a compatible image file format that can be readily utilized within the pipeline.

Additionally, a text file is generated within this directory, containing the recorded distance between the nipples. This organization ensures efficient and easy to understand data management and facilitates seamless integration within the subsequent stages of processing.

Upon completion of the image processing pipeline developed by Arbrea, the resulting 3D model is saved within the same directory as the original images. Our application employs the widely used Wavefront OBJ file format for storing the 3D model, utilizing three distinct files. The primary file, denoted with the .obj extension, contains the actual geometric data of the 3D model. Additionally, the .mtl file is employed to store material-related information, while the .png file serves as a repository for texture information. These three files constitute the output generated by the pipeline.

Given that both the .obj and .mtl files are text-based, they can be easily processed by reading and transmitting them as strings to the client. However, the texture file, being an image file, necessitates encoding prior to transmission over the HTTP connection. Within our implementation, we have incorporated the functionality to perform individual GET requests to the server for each file separately. This enables the precise retrieval of the required files from the server in a granular manner.

3.2.2. Flutter

On the client-side of the application, an initial step involves initiating a POST request to the server. To facilitate this communication process, we employ the Dio package, a robust HTTP client specifically designed for Dart programming. This package provides comprehensive functionalities for handling HTTP requests and responses efficiently and effectively.

As previously mentioned, the server expects a JSON object as part of the POST request, comprising the three torso images in the following order: frontal, right-side, and left-side images. Additionally, the JSON object should include the distance between the patient's nipples and a unique identifier. Before transmitting the images to the server, it is necessary to encode them appropriately. To achieve this, we utilize another package, specifically the http package, which offers a convenient means of encoding images using the local URL of the Binary Large Object (BLOB).

It is worth noting that in scenarios where the image is uploaded or when operating within the

Android environment, direct access to the encoded image data is available, thus bypassing the need for additional encoding steps. Following this, the application prompts the user to input the distance between the patient's nipples, subsequently generating a unique identifier. Finally, utilizing the Dio package, the POST request is dispatched to the server, facilitating the seamless transmission of the required data.

To obtain the 3D model from the server, we employ a GET request, utilizing the Dio package once again to manage the communication process. Our application necessitates two versions of the 3D model: one that allows modifications and augmentations to the breast area and another that remains unaltered. Consequently, we perform two separate GET requests to retrieve these models.

As our visualization method is based on the Flutter Package Cube, we require three GET requests per model: one for the .obj file, one for the .mtl file, and one for the .png file. To accommodate this requirement, we have made adaptations to the existing Cube parser to accept a URL instead of a local file path. The parser is then responsible for performing a GET request for each file from the server and subsequently parsing the received data.

It is important to note that Dart employs base 16 encoding for strings, unlike Python's base 64 encoding. Consequently, when working with raw data, particularly when performing a GET request for the .png file, caution must be exercised. We must first decode and re-encode the data appropriately before utilizing it within the Cube parser. This ensures compatibility and proper handling of the data within the application.

3.3. Visualization of the 3D Model

As previously discussed, the visualization of the 3D model is primarily done by the integration of the Flutter package Cube. This package presents us with a 3D model viewer, which uses a standardized implementation of a computer graphics pipeline. In order to effectively render the 3D models, Cube relies on an internal representation of these models, which is described and structured by the object.dart file. Understanding this representation is crucial for our application, because we want to be able to change vertices of the model, and as discussed above, we have a different method of initializing the model.

It is worth mentioning that in cases where we have two objects within the scene, one loaded from local memory and the other fetched from the server, there may be a noticeable distinction in their appearance, manifesting as a bluish hue. This discrepancy arises due to the contrasting color descriptions utilized, specifically a mixture of RGB (Red, Green, Blue) and RGBA (Red, Green, Blue, Alpha) color formats. While this discrepancy surfaced unexpectedly during the debugging phase, it is essential to note that it does not significantly impact our application since we rely on the server as the source of our models.

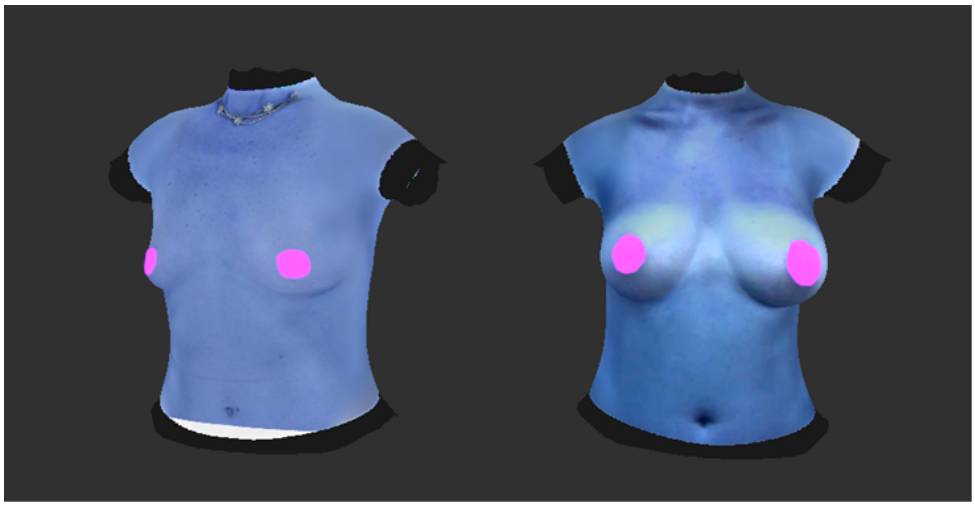


Figure 3.2.: *Screenshot of an early prototype displaying a bluish hue. Note that the nipple covers were painted on the image after it was taken.*

In our implementation we decided on having two screens, which display the 3D models. In the first screen we display the 3D model for breast modification. We implement three different views, one from the front, where the model can rotate around the y-axis:

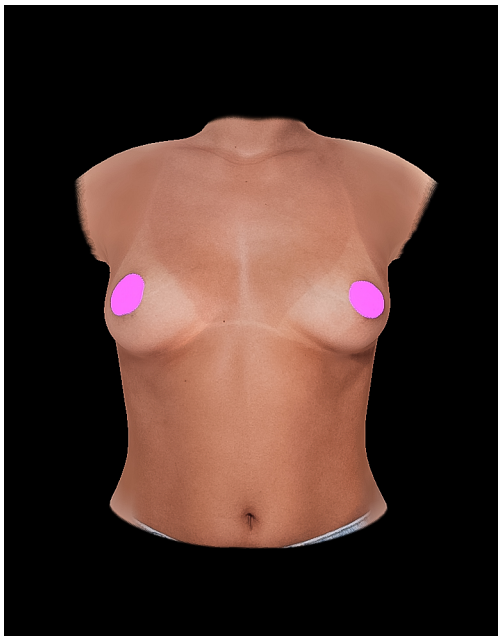


Figure 3.3.: *Frontal view of the 3D model*

One from the top, where the model can slightly rotate around the x-axis:

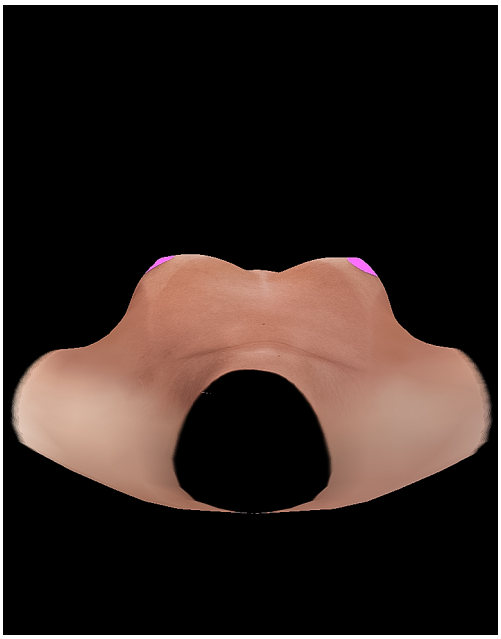


Figure 3.4.: *Top-down view of the 3D model*

And finally a view from below, in which the user can also slightly rotate the model around the x-axis:

Figure 3.5.: *Bottom up view of the 3D model*

The second screen displays the modified model next to the unmodified model, where the user can compare the two models.

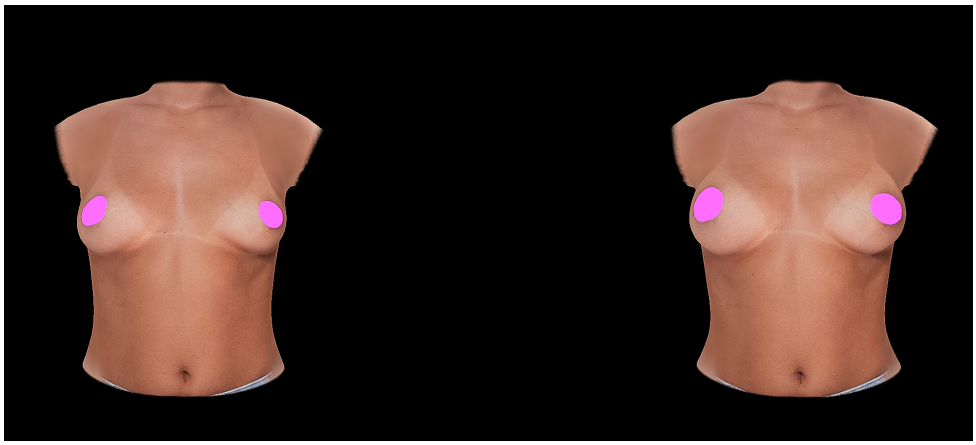


Figure 3.6.: *Comparison Screen*

Within the object.dart file, one can locate a mesh object, as explicitly defined in the mesh.dart file. This particular mesh object encapsulates an array of vertices, polygons, normals, and various other essential details pertaining to the model, which collectively contribute to its visual representation. It is worth noting that a function is employed to duplicate vertices that possess multiple texture coordinates. This duplication process ensures that each vertex within the mesh object maintains a singular texture coordinate. We needed to keep this in mind for later, because when we want to change the vertices of the model we can not simply take these vertices, as they are not the same after the duplication.

The Cube package provides us with the essential transformation matrices, specifically the model, view, and projection matrices. These matrices are of great importance as they enable us to perform various transformations on the 3D model and facilitate its visualization. The presence of these matrices within the scene.dart file streamlines our subsequent tasks, as we can readily access and utilize them when required.

3.4. Modification of the 3D Model through PCA

Our primary objective revolves around the ability to manipulate the 3D breast model to simulate the outcomes of a breast augmentation surgery. There exist many options to achieve this goal. However, our chosen approach entails employing the Principal Component Analysis (PCA) method, a statistical procedure that facilitates a substantial reduction in the dimensionality of our data. This methodology aligns with the techniques employed by Arbrea, thereby enabling potential future enhancements to be implemented in a manner consistent with those utilized in the Lind App.

3.4.1. Principal Component Analysis

Principal Component Analysis (PCA) is a statistical technique used to analyze and reduce the dimensionality of a dataset while preserving its essential characteristics. It identifies the principal components, which are linear combinations of the original variables that capture the maximum

amount of variance in the data. By projecting the data onto these principal components, PCA enables us to interpret complex datasets like for example 3D models of the female torso. PCA aids in uncovering patterns, relationships, and underlying structures in high-dimensional data, making it a valuable tool for dimensionality reduction and data compression.

Due to the convenience and functionality provided by Python for performing Principal Component Analysis (PCA), we opted to conduct the PCA fitting and initial analysis in Python. Python offers comprehensive libraries and tools, such as Scikit-learn, that allow an easy implementation of PCA and provide extensive support for data analysis and manipulation. By leveraging Python's capabilities, we can efficiently extract the principal components from our dataset and explore their significance and contribution to the variance. Finally, we can utilize the obtained PCA results in our Flutter application for further processing and visualization of the modified 3D models.

Our initial step involved acquiring a dataset from Arbrea, which consisted of 266 3D models. Since our objective was to modify only the vertices of the 3D models, we extracted the vertex information exclusively from the dataset. This resulted in a modified dataset comprising 266 3D models, with each model consisting of 3,354 vertices, each defined by three coordinates (x, y, and z). Consequently, the dimensionality of each 3D model was 10,062, representing a vector of length 10,062.

To perform the Principal Component Analysis (PCA), we utilized the PCA implementation provided by the Scikit-learn library (SciKit). By fitting our dataset to the PCA, we obtained the mean vector μ (also of length 10,062) and the covariance matrix Σ (a 10,062 x 10,062 matrix). To reduce the dimensionality of the dataset and analyze its principal components, we utilized the numpy.linalg.eig function in numpy, which computes the eigenvalues $\lambda_1 ... \lambda_n$ and the corresponding eigenvectors $v_1 ... v_n$ of the covariance matrix Σ. This decomposition process took approximately four minutes to complete. Subsequently, we saved the mean vector, standard deviations, and eigen vectors to be used for later reconstruction of the 3D models.

At this stage, we saved all the eigen vectors, as the number of principal components to retain was not yet determined. Theoretically, only the first k eigen vectors would be necessary, where k represents the desired number of principal components. However, to allow flexibility in selecting the number of principal components, we preserved all of them in our saved results.

To get the k eigenvalues λ_k corresponding to a specific model w we use the following formula:

$$\lambda_k = U_k * (w - \mu) \tag{3.1}$$

Where U_k are the first k eigenvectors $v_1 ... v_k$ arranged in a matrix, and μ is the mean vector, both determined by the principal component analysis we did earlier.

Through the Principal Component Analysis (PCA), we have successfully achieved a significant reduction in dimensionality. Initially, our input vector w, had a high dimensionality of 10,062. However, after applying the PCA, we obtained λ_k with a reduced dimensionality of k. This reduction in dimensionality allows processing of the data through capturing of the most important information through the principal components.

We can now use λ_k to reconstruct our 3D model $\hat{w}$, by using the following formula:

$$\hat{w} = U_k * \lambda_k + \mu \tag{3.2}$$

The next step was to analyze the principal components, to give them a meaning and understand their contribution to the model. This was done by setting all but one eigenvalue λ_i in λ_k to zero, and setting λ_i to the standard deviation σ_i of the corresponding eigenvalue.:

$$\bar{\lambda_k} = [0, ... , 0, \sigma_i, 0, ... , 0]^T \tag{3.3}$$

We proceeded to reconstruct our model $\hat{w_k}$ by setting λ_k to (3.3) in (3.2). Applying this reconstruction, we obtained a modified version of the model that showcased the effects of manipulating a specific eigenvalue.

To visualize the reconstructed model, we displayed it alongside the original model, enabling a direct comparison between the two. This allowed us to observe and understand the impact of altering a specific eigenvalue on the appearance and shape of the model. This process provided valuable insights into the interpretation and significance of individual eigenvalues.

Deconstruction and Reconstruction in Flutter

After conducting the Principal Component Analysis (PCA) in Python, we aimed to integrate this functionality into our Flutter application. To achieve this, we stored the calculated mean vector μ, standard deviations $\sigma_1, \ldots, \sigma_k$, and eigenvectors $v_1, \ldots, v_k$ in separate files. In our application, we utilized these files by reading them into memory for subsequent operations.

To enable real-time deconstruction and reconstruction of 3D models, we required efficient matrix multiplication capabilities within the Flutter environment. To address this, we leveraged the ml_linalg library, which offers matrix and vector classes, along with support for SIMD (Single Instruction, Multiple Data) instructions. This allowed us to perform fast matrix multiplications by relying on the ml_linalg library.

After receiving a 3D model from the server, we initiated the deconstruction process to obtain the first 30 principal components, as defined in equation (3.1). We saved the resulting vector λ_k twice: one for reconstructing modified models and another as a reference for reconstructing the original model. We then proceeded to reconstruct the model $\hat{w}$ based on (3.2), utilizing the obtained λ_k vector.

To enable user interaction and model modification, we implemented sliders that allow users to adjust the values of specific principal components. The range of adjustment was set to $\pm 3\sigma_i$, which covers approximately 99.73% of the data based on the PCA analysis performed on our dataset. Each time a user modifies a slider value, we recompute the model reconstruction using the above-mentioned process, resulting in an updated model representation.

3.4.2. Analysis of the principal components

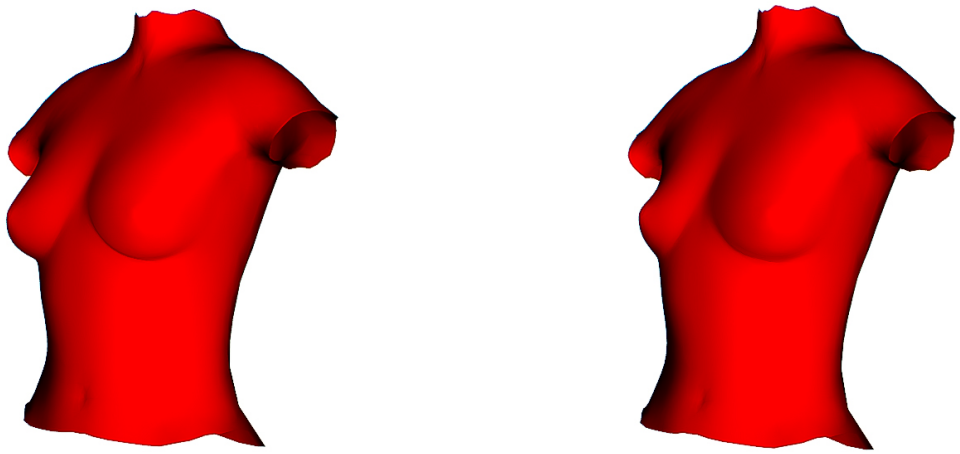
First principal component

After analyzing the principal components, we observed that the first principal component primarily represented the general form of the model, indicating the overall size of the patient. After

careful consideration, we concluded that modifying this principal component to alter the breast would not be advisable for two main reasons.

Firstly, changing the first principal component would result in a reconstructed model that deviates from anatomical correctness. Since the first principal component predominantly captures the overall size and structure of the patient, manipulating it to modify the breast would lead to unrealistic proportions and potentially compromise the integrity of the model.

Secondly, altering the first principal component may result in a reconstructed model that does not resemble the actual patient. Modifying it extensively could lead to a reconstructed model that does not accurately reflect the patient's unique features, thereby rendering it unsuitable for our purposes.



(a) (b)

Figure 3.7.: *First principal component.* (a) *Original model.* (b) *Model reconstructed increasing the first principal component by* $2 \cdot \sigma_1$.

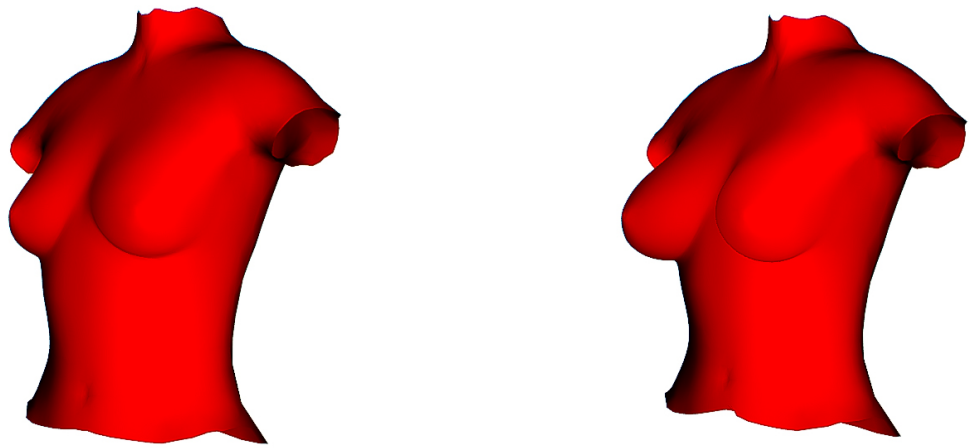
Second principal component

Upon analyzing the second principal component, we discovered that it primarily captured the size of the breast. This finding was highly significant for our breast augmentation application, as breast size is a crucial factor in the surgical procedure.

An intriguing observation was that the influence of the second principal component on other aspects of the breast was relatively minimal. This implied that modifying the breast size using this principal component would have a localized effect, primarily altering the size of the breast

while leaving other aspects of the breast area largely unaffected.

This insight was particularly valuable as it allowed us to focus on breast size modification without compromising the overall structure and characteristics of the breast. By selectively adjusting the second principal component, we could achieve targeted changes in breast size, enabling patients to visualize and assess the potential outcomes of breast augmentation surgery accurately.



(a) (b)

Figure 3.8.: *Second principal component.* (a) *Original model.* (b) *Model reconstructed increasing the second principal component by* $2 \cdot \sigma_2$.

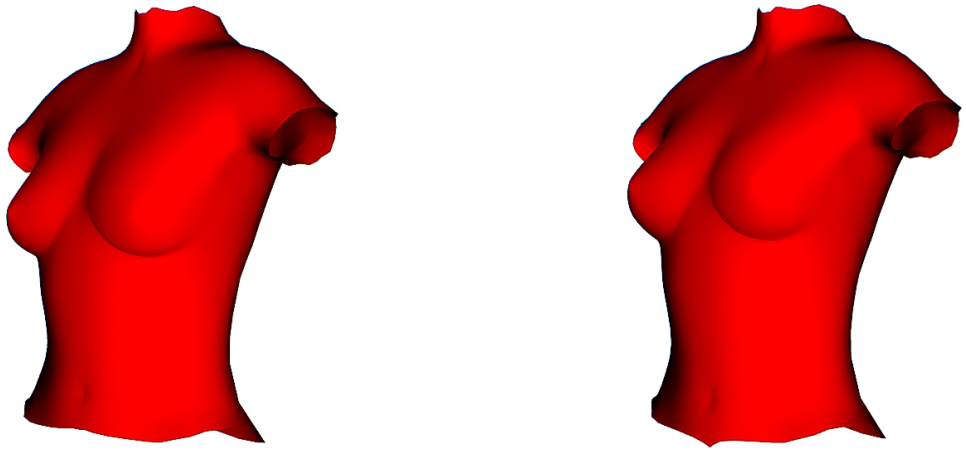
Third principal component

After analyzing the third principal component, we discovered that it predominantly represented the vertical lift of the breast. This finding was highly relevant for our breast augmentation application, as breast lift (also known as mastopexy) is a meaningful factor in the surgical procedure.

Similar to the second principal component, we observed that the influence of the third principal component on other aspects of the breast was relatively limited. This suggested that modifying the breast lift could be accomplished by selectively adjusting the third principal component while maintaining the integrity of other breast attributes.

This insight enabled us to focus on enhancing the breast lift aspect without compromising the overall structure and characteristics of the breast. By manipulating the third principal compo-

nent, patients could visualize and assess the potential outcomes of breast lift surgery accurately.



(a) (b)

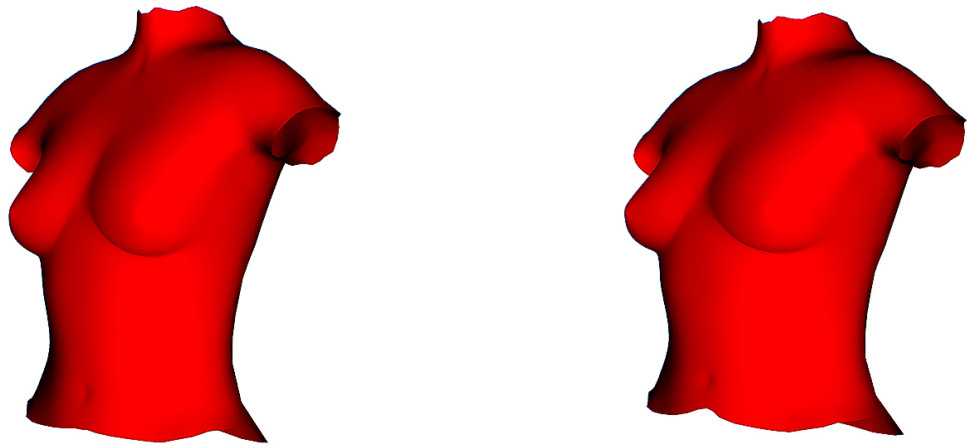
Figure 3.9.: *Third principal component.* (a) *Original model.* (b) *Model reconstructed increasing the third principal component by* $2 \cdot \sigma_3$.

Fourth principal component

Our findings of the fourth principal component showed us that it is primarily responsible for the cleavage width of the breast. This finding was interesting to us, since also this is a factor that we want to be able to change.

Similar to our previous findings, we noted that manipulating the fourth principal component had minimal impact on other aspects of the breast. This implied that altering the cleavage width could be achieved by selectively adjusting the fourth principal component while preserving the overall appearance of the breast.

The ability to isolate and modify the cleavage width component independently offered considerable flexibility in tailoring the breast augmentation process to achieve desired aesthetic outcomes.



(a) (b)

Figure 3.10.: *Fourth principal component.* (a) *Original model.* (b) *Model reconstructed increasing the fourth principal component by* $2 \cdot \sigma_4$.

Remaining principal components

After analyzing the remaining principal components beyond the fourth component, we did not identify any significant or relevant meanings that could be attributed to them in the context of our breast augmentation application. This outcome was not unexpected, given that the standard deviations associated with these components were relatively small.

The relatively small standard deviations indicated that these components were less likely to capture notable variations or distinct characteristics of the breast shape within our dataset. Thus, the probability of a patient's breast exhibiting significant features associated with these components was deemed to be low. Consequently, we did not implement functionality in our software to modify these components.

However, as the project progressed, we gained further insights into the meaning of some of these remaining principal components. While these components were found to contribute to the visualization quality, they were not deemed necessary to be modified for our application's purposes.

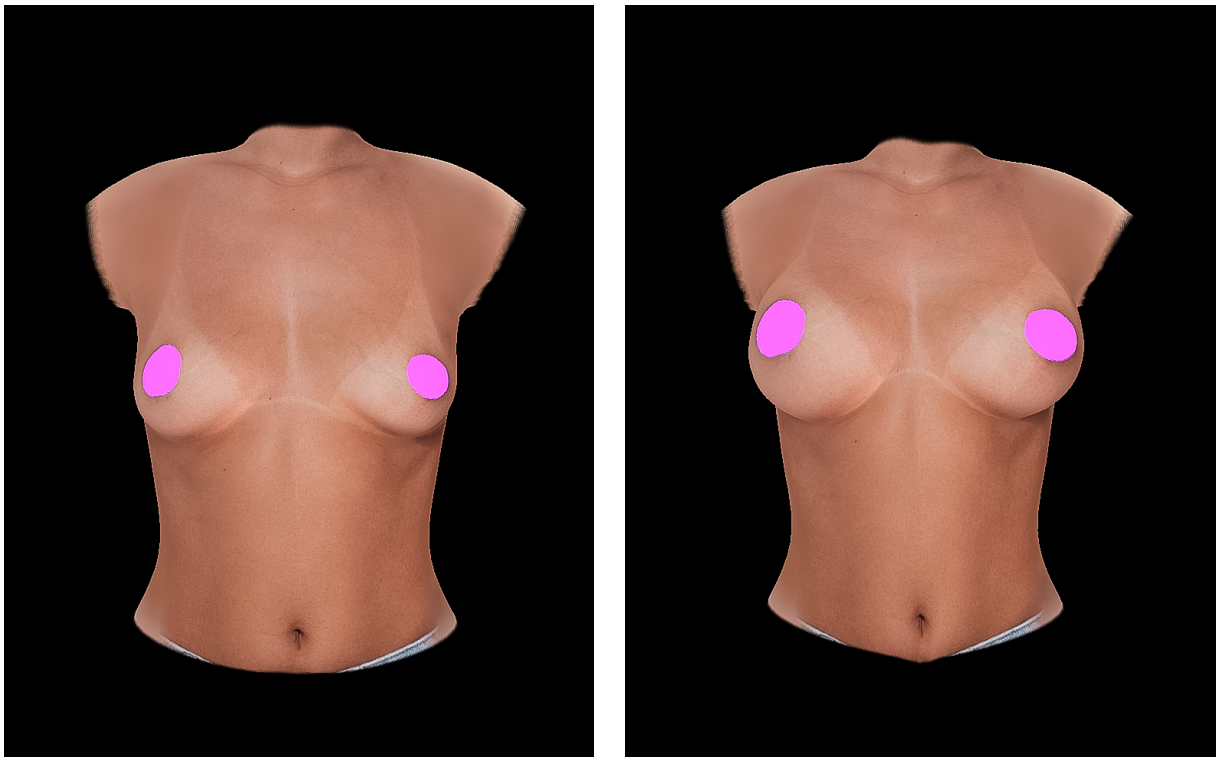
It is worth noting that despite not being directly modified, these remaining principal components still contributed to capturing and displaying various characteristics of the breast shape when using the first 30 principal components. This allowed for a comprehensive representation of the breast shape while focusing on the key components relevant to breast augmentation. We

provide an analysis of the reconstruction accuracy in Section 4.1.

3.4.3. Region specific modification

As previously mentioned, the second to fourth principal components primarily affect the breast area while having minimal impact on other aspects of the breast. However, it is important to note that these effects are specific to the breast area and do not extend to the rest of the torso or body.

When modifying these components, we observed that the changes in breast size, lift, and cleavage width were isolated to the breast region. However, the rest of the torso, including surrounding areas, underwent deformation as well. This unintended deformation of the torso was deemed undesirable as it compromised the overall resemblance of the augmented model to the patient's actual body.



(a) (b)

Figure 3.11.: *Deformation of the model through changes in principal components.* (a) *Original model.* (b) *Model reconstructed changing some principal components.*

Maintaining a realistic representation of the patient's body is crucial for a successful breast augmentation simulation. Thus, it was necessary to ensure that changes to the breast area did not result in significant distortions or alterations to the rest of the torso.

There are two potential approaches to address the issue of undesired changes in the torso when modifying the breast area. The first approach would involve modifying all the principal com-

ponents that affect the torso in order to counteract the unintended changes caused by desired modifications in the breast area. However, this approach contradicts the purpose of using PCA, as it aims to enable targeted modifications to specific aspects of the breast while preserving the overall body shape.

The second approach, which we adopted, focuses on refining the selection and manipulation of principal components related to the breast area while minimizing the impact on the rest of the torso. This approach acknowledges the trade-off between realistic breast modifications and maintaining the overall resemblance to the patient's body shape. By carefully selecting and manipulating specific principal components, we can achieve desired breast modifications while minimizing distortions in the surrounding areas.

In the 3D model obtained from the pipeline, each vertex retains its original position, meaning that a vertex located in the nipple area will remain in the nipple area throughout the modifications. Therefore, to isolate and manipulate the breast area, it is sufficient to know the indices of the vertices that correspond to this specific region.

Fortunately, Arbrea Labs has already performed the necessary work of identifying and indexing the vertices that constitute the breast area within the 3D model. This information is provided to us, saving us the effort of manually determining these vertex indices.

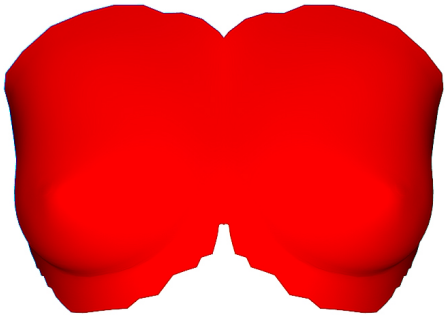


Figure 3.12.: *Model corresponding to breast area vertex indices.*

After observing the modified model, we noticed the presence of sharp and unnatural transitions between the breast area and the surrounding torso. This abrupt change occurs because the boundary region between the breast and the rest of the torso abruptly shifts from the original model $\hat{w}$ to the modified model $\hat{w}'$. To mitigate this issue and create a more visually appealing result, we opted to implement interpolation between the two models in the boundary region.

In our implementation, we adopted a two-step process to handle the interpolation of the modified model $\hat{w}'$ with the original model $\hat{w}$ in the boundary region. Firstly, we identified all the

vertices located outside the breast region and assigned them the corresponding vertices from the original model $\hat{w}$. This step ensures that the geometry outside the breast area remains unaltered.

Next, we focused on the vertices inside the breast region that lie outside a certain distance threshold d from the boundary. For these vertices, we replaced them with the corresponding vertices from the modified model $\hat{w}'$, preserving the desired modifications within the breast area. The distance threshold d used in determining the extent of modifications applied to the breast area must be chosen carefully. By adjusting the value of d, we can regulate the range over which the interpolated modifications take effect. A smaller value of d confines the modifications closer to the boundary region, resulting in a more localized, steeper change. Conversely, a larger value of d leads to a more conservative change.

For the remaining vertices lying between the breast region and the boundary, we computed their nearest distances to the boundary vertices, denoted as d_i. Based on these distances, we performed a weighted average between the original model $\hat{w}$ and the modified model $\hat{w}'$ for each vertex v_i. The weight assigned to each model was determined by the distance d_i, where vertices further away of the boundary were more influenced by the modified model $\hat{w}'$, while vertices closer to the boundary retained a stronger resemblance to the original model $\hat{w}$.

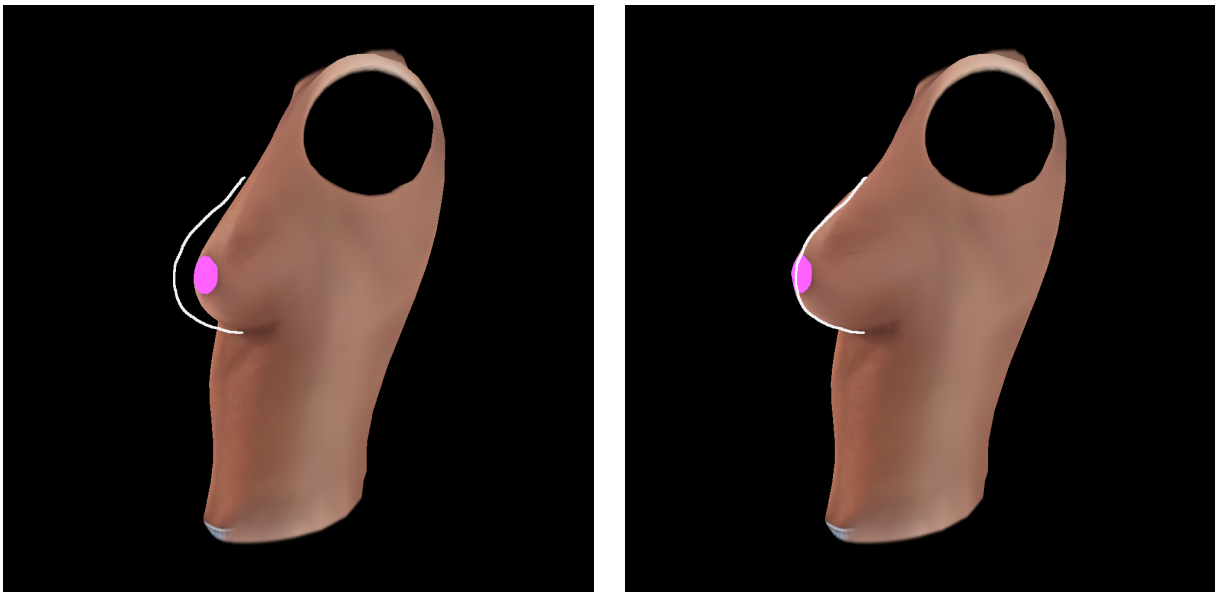
$$v_i = \frac{d_i}{d}\hat{w_i}' + \left(1 - \frac{d_i}{d}\right)\hat{w_i}$$

By employing this weighted interpolation strategy, we achieved a smooth and gradual transition between the modified and original models, effectively mitigating the sharp changes observed in the boundary region. This approach ensures a more natural and visually pleasing result, where the modifications within the breast area blend seamlessly with the surrounding geometry.

3.5. Model modification through Correction UI

Another objective of this thesis is to develop an interactive user interface that enables users to modify the breast model in a user-friendly and intuitive manner. A key feature we aim to incorporate is the ability for users to draw a line on the breast model, indicating their desired breast shape. The application then utilizes the principal components λ_k to modify the breast model in an attempt to align it as closely as possible with the drawn line. By manipulating the principal components λ_k, we ensure that the resulting breast model maintains a realistic appearance.

It is important to note that achieving an exact fit between the drawn line and the modified breast model may not be feasible. Instead, our approach focuses on minimizing the distance between the drawn line and the modified breast model to achieve the best possible alignment. This approach strikes a balance between user customization and maintaining the natural aesthetics of the breast model. The real-time nature of our application further enhances the user experience, allowing users to observe the changes to the breast model in immediate response to their input.



(a) (b)

Figure 3.13.: *The desired result of the correction UI is to modify the breast model to align with the user-drawn line.* (a) *Original model with user-drawn line.* (b) *Result of the correction process.*

3.5.1. Problem specification

To obtain the desired functionality, it was necessary to formulate the problem mathematically. The most straightforward approach was to treat the problem as a minimization problem, aiming to minimize the distance between the user-drawn line and the modified breast model. This problem can be conceptualized as follows:

$$\arg\min_{\lambda_k} = \|f(line) - s(g(\lambda_k))\| \tag{3.4}$$

Where $f(fline)$ represents a function that takes a user-drawn line and returns a corresponding plane in 3D space, representing the user-drawn line. $g(\lambda_k)$ denotes a function that takes the principal components λ_k and performs the reconstruction of the breast model. $s(\hat{w})$ represents a function that takes the reconstructed breast model $\hat{w}$ and returns the silhouette of the breast area.

However, there are several aspects that need to be addressed in order to minimize this distance effectively. To accomplish this, we need to determine the functions f, g, and s and obtain the necessary inputs.

Fortunately, we already have knowledge of how to reconstruct the 3D model $\hat{w}$ from the PCA parameters λ_k, which provides us with the function $g(\lambda_k)$.

Obtaining the user-drawn line, denoted as $line$, is also feasible within the Flutter framework. By utilizing the Gesture Detector widget, we can access the coordinates of the latest touch or

mouse events, which can be stored in a list for subsequent use.

For the rest of the functions we decided to simplify our problem. The way in which we simplify our problem is by assuming the silhouette of the 3D model stays on a constant x coordinate. This means that the line defined by the silhouette is the intersection of the breast model with a plane parallel to the yz plane defined by a constant x coordinate. We further simplify our problem by taking the vertices closest to this intersection, and using them to define the line this means we have now defined the function s. This simplification also means that we have a significantly easier solution to our function f, As we don't need to take the distance from the silhouette to a plane, but instead we can just take the distance from the silhouette to a list of points.

3.5.2. Getting a line in 3D from 2D

As we now have a simpler function f to find, we can start by again looking at the implementation of the Cube package.

Our approach involves the reverse projection of 2D points from image space to a line in 3D space. By intersecting these lines with the same yz plane as the silhouette, we obtain a list of points in 3D space. This allows us to reconstruct the breast model based on the user-drawn line and the corresponding 3D points.

The Cube implementation uses a standard graphics pipeline to project the 3D model coordinates to 2D image space. This means that first the 3D model coordinates get transformed by the model transform, then they get transformed by a view transform, and finally they get transformed by a projection transform:

$$I = T \cdot w$$

where

$$T = P \cdot V \cdot M$$

Where I is the 2D image space coordinates, P is the projection transform, V is the view transform, M is the model transform and w is the 3D model coordinates.

It is important to know that the projection transform is a perspective projection, which means that the 3D model coordinates (x, y, z) get projected to the $[-1, 1] \times [-1, 1] \times [0, 1]$ device coordinates corresponding to the view frustum of the camera. Cube then uses $(x, y) \in [-1, 1] \times [-1, 1]$ coordinates to draw the 3D model on the screen, and saves the z coordinate for later use.

We can now modify the Cube package, so it gives us the transformation matrix T. We can then use T^{-1} to do the back projection from 2D image space to 3D model space.

Assuming we are given the coordinates of a drawn pixel on the screen (u, v), we then need to scale these coordinates to the $[-1, 1] \times [-1, 1]$ range. We do this in the following way:

$$p_x = \frac{u}{\frac{S_x}{2}} - 1, p_y = -\frac{v}{\frac{S_y}{2}} - 1$$

where S_x and S_y are the width and height of the screen in pixels. Note that p_y is negated, because the point $(0, 0)$ lies in the top left corner. We now have a 2D point $p_I = (p_x, p_y)$ in image space, we can now use this point to get a line in 3D space. We do this by first defining two points in homogeneous coordinates:

$$p_{I1} \coloneqq \begin{bmatrix} p_x \\ p_y \\ 0 \\ 1 \end{bmatrix}, p_{I2} \coloneqq \begin{bmatrix} p_x \\ p_y \\ 1 \\ 1 \end{bmatrix}$$

We now have two points, p_{I1} at the near plane of the frustum, and p_{I2} at the far plane of the frustum. Furthermore, we now use the inverse transform T^{-1} to get the 3D coordinates of the points in model space:

$$p_{M1} = T^{-1} \cdot p_{I1}$$
$$p_{M2} = T^{-1} \cdot p_{I2}$$

In a next step we need to normalize the homogeneous coordinates of p_{M1} and p_{M2} to normal 3D coordinates, let's define $s \coloneqq p_{M1}$ and $t \coloneqq p_{M2}$, then:

$$q = \begin{bmatrix} s_x/s_w \\ s_y/s_w \\ s_z/s_w \end{bmatrix}, r = \begin{bmatrix} t_x/t_w \\ t_y/t_w \\ t_z/t_w \end{bmatrix}$$

We now connect q and r with a line and intersect this line with the yz plane defined by a constant x coordinate. This is done by calculating the vector v that points from q to r:

$$v = q - r$$

Then we solve for p, the point where the line intersects the yz plane:

$$p = q + v * \alpha$$

Where

$$\alpha = \frac{x - q_x}{v_x}$$

Note that x is the constant x coordinate of the yz plane.

This then gives us the point p in model space. We can now repeat this process for all the points in the list of points we got from the Gesture Detector widget. This gives us our line, and thus this is our function f.

3.5.3. Methods

We now need to solve the minimization problem from (3.4). As this is a non-linear optimization problem, we have multiple different methods to solve this. In the following section we will discuss some of the methods we considered using.

Newton's Method

Newton's method is a root finding algorithm, which tries to find a root of a function f. It uses an iterative approach in conjunction with the function's derivative f'.

The idea is to start with an initial guess x_0 and then approximate the function f by its tangent line at x_0. Then the algorithm calculates the x-intercept of this tangent line, and uses this as the next guess x_1. The algorithm uses the following formula to calculate the next guess:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Newton's method has quadratic convergence under certain assumptions. This means that if the assumptions don't hold, the algorithm might not converge at all, or be in a local minimum. This can be due to multiple factors, we list some of them here:

- The function f has a stationary point. This means that while calculating the next guess, the algorithm will divide by zero, and thus crash or terminate.
- The function f is not differentiable
- For certain functions f and certain initial guesses x_0 Newton's method might also overshoot the root, and diverge away from the root.

In practice, we can solve some of these problems by using different initial guesses x_0, and lower the step size m of the algorithm. This means that the algorithm will make smaller steps, and we can even lower the step size in each iteration, so we make large steps in the beginning and smaller steps when we get closer to the root. This variation is called damped Newton's method. The formula for calculating the next guess in damped Newton's method is as follows:

$$x_{n+1} = x_n - m \cdot \frac{f(x_n)}{f'(x_n)}$$

Quasi-Newton Method

The quasi Newton method is a variation of Newton's method. Essentially it approximates the derivative f' of f in some form. This is particularly useful if the derivative f' is hard to calculate, or unavailable at all.

Limited Memory BFGS

The Limited Memory BFGS method (L-BFGS) is a quasi Newton method, which uses an estimate of the inverse Hessian matrix to navigate through the variable space.

This algorithm was used by us to do a first implementation of our minimization problem. We used the L-BFGS-B implementation from the scipy.optimize package in Python. This implementation didn't require any information on the derivative or the Hessian matrix of the function f. This was useful for us, as we didn't have to calculate the derivative of f by hand.

Our function $f(x)$ that is to be minimized first calculated the 3D model of through PCA reconstruction as in (3.2). Then it calculated the closest distance between each predefined vertex on the breast and the vertices on the line. It then summed up all these distances and returned the sum. As one can see it looks similar to (3.4)

$$f(x) = \sum_{v \in V} \min_{p \in L} \|p - v\|$$

Where $v \in V$ are the vertices on the breast, and $p \in L$ are the points on the line.

Since the algorithm tries to estimate the inverse Hessian matrix, it uses many evaluations of the function f. This of course is very time intensive and undesirable for a real-time use case. We then decided to use a different implementation of this algorithm, which uses parallelism to speed up the minimization process. We used the method shown in [Ger20] and [GF19] We further simplified our function $f(x)$ by only considering principal components 2 to 8 as input to our function f. This was done, because on our testing machine we had 8 cores, and this implementation gives us a speedup of $1 + p$, where p is the number of parameters, so our principal components, and $1 + p$ cores are available. Furthermore, we tested different amount of iterations to get an idea on how many iterations we need to get a good result. Please see the evaluation section for more information on this.

Deep Reinforcement Learning

As another idea we had Deep Reinforcement Learning in our mind. This of course is a very different approach since it uses a neural network. We thought that this method is a bit overkill for our problem, but we want to keep it in mind for future applications. Especially when we want to optimize the unsimplified model (3.4) it might be worth to look into this method. Creating the dataset shouldn't be a problem, as we can just use our reconstruction function (3.2) with arbitrary λ_k and some noise to generate a large dataset.

3.5.4. Our Implementation

In the end we decided to use a simple Newton method. This is because we needed to do our computation in dart, and not in Python. This is a problem because while there are many libraries in python, there is next to nothing for dart, so we need to do everything by hand. As the Newton method is a very simple algorithm we decided to implement it ourselves.

Here the main problem was to get the derivative. We had to further approximate our function f by treating the vertices of the line and the breast as constant in one iteration. This of course does not reflect reality, as a fixed vertex of the 3D model can move and thus change the point on the line that it is closest to. But this approximation allowed us to simply calculate the derivative of f by hand. Let's consider our new loss function l:

$$l(x) = \|x_i - y_j\|^2$$

where

$$x = \hat{p} \cdot U_k + \mu$$

where $\hat{p}$ are the principal components, the x_i are the vertices of the approximated silhouette of the breast, and y_i are the corresponding closest points on the line. We want the following derivation:

$$\frac{\partial l}{\partial \hat{p}}$$

This gives us

$$\sum_i 2 * d_i * (v_{ix} + v_{iy} + v_{iz})$$

where

$$d_i = \|x_i - y_j\|$$

and v_{ix}, v_{iy} and v_{iz} are the components corresponding to the x, y, z coordinate of the i-th vertex of the breast, given by the PCA. This then gives us an approximation of the derivative of f at x. We can now use this to calculate the next guess x_{n+1} in the Newton method.

As we can never reach a loss of zero, the method will never converge. This is because the line we draw is most certainly not a line that we can represent with reconstructing the 3D model through 30 principal components. This means we could either say we terminate the method after the change between iterations is below a certain threshold ϵ

$$|l_{n+1} - l_n| < \epsilon$$

or after a certain amount of iterations is reached. We chose the second option, a limited amount of iterations, because then we can limit the time it takes to calculate a fit of the 3D model to the drawn line.

It is important to note that we limited the number of principal components the algorithm changes, because we would get unrealistic results. We decided on which components to change by trying out different combinations. Since we draw the line only for one breast, but change both of them at the same time, it is important to change them in the same way. Our tool would be close to useless if it would change one breast to a different size than the other. This is the main reason why we can't change all principal components, even if our loss l would be less.

By trial and error we found out, that changing the fifth principal component leads exactly to unproportional change in one breast but not the other. We further decided to change the first eight principal components. To achieve this we simply set the desired vector components of the derivative to zero.

4

Evaluation

In this section, we will analyze and evaluate the results of our methods, considering their quality and effectiveness. To provide a comprehensive understanding, we will present various plots and visualizations to support our findings and observations. These plots will serve as quantitative and visual evidence of the outcomes achieved through our approach.

4.1. Reconstruction Accuracy

Our initial experiment aims to assess the reconstruction accuracy achieved through the application of Principal Component Analysis (PCA). To conduct this experiment, we compare the reconstructed 3D models generated using varying numbers of principal components to the original input models. We calculate and analyze the reconstruction error, which quantifies the dissimilarity between the original models and their reconstructions.

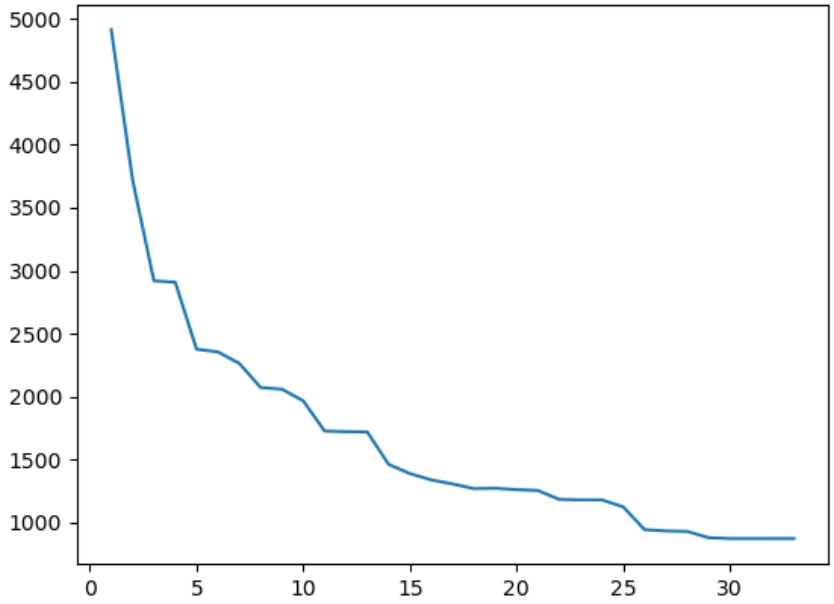


Figure 4.1.: *Reconstruction Accuracy*

Figure 4.1 shows hat we can clearly see the first few principal components are the most important ones. It is important to note that the method of calculating the dissimilarity is done by summing up the euclidean distances between the vertices of the original and reconstructed models:

$$\sum \|v_i - u_i\|$$

where v_i is the ith vertex of the original model and u_i is the ith vertex of the reconstructed model. Given the non-normalized nature of the vertex coordinates, we interpret this distance as a form of relative distance rather than an absolute measurement. Yet figure 4.1 still shows relevant information.

4.2. Correction UI

4.2.1. L-BFGS-B

Here we discuss the implementation of SciPy's L-BFGS-B minimization algorithm. SciPy's implementation of scipy.minimize() allows to choose many minimization algorithms. Because we have a parallel implementation of the L-BFGS-B algorithm, we compare it to the serial implementation of the L-BFGS-B algorithm.

The principal parameters in which we are interested in are the number of iteration and the number of principal components changed. We will show the effect of these parameters on the execution time and the loss.

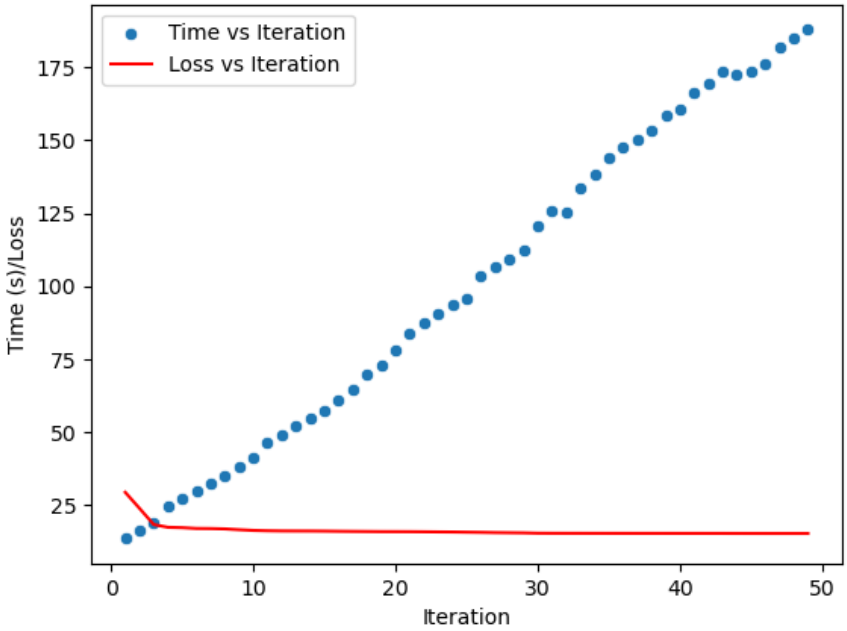


Figure 4.2.: *Plot of the serial quasi Newton method showing execution time and loss vs the number of iterations*

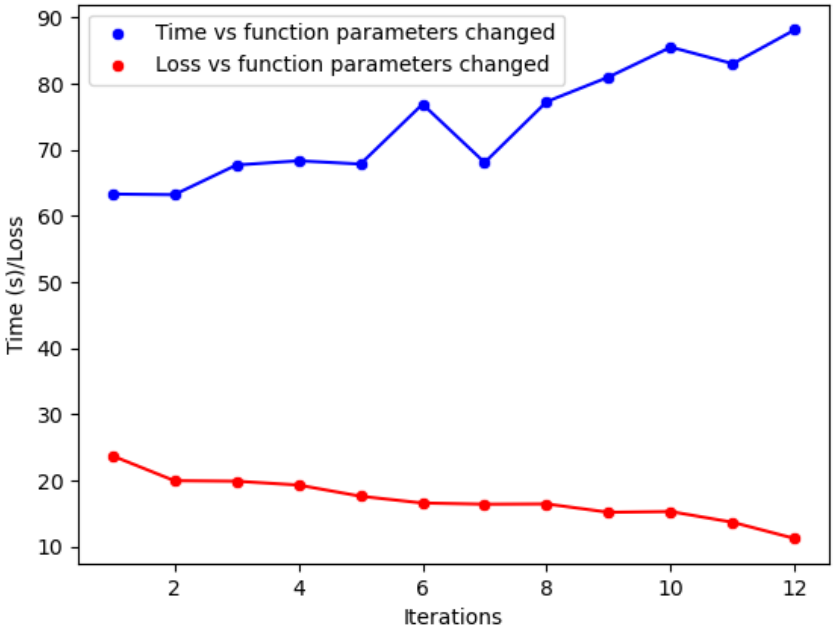


Figure 4.3.: *Plot of the serial quasi Newton method showing execution time and loss vs the number of principal components changed*

Figure 4.2 shows the execution time and loss of the serial Quasi Newton method vs the number of iterations. We see a linear increase in the execution time, while after a certain number of iterations the loss does only decrease slightly. This means that if we wanted to implement this method, we would have to stop the optimization after a few number of iterations.

Figure 4.3 shows the execution time and loss of the serial Quasi Newton method vs the number of principal components changed. We kept the number of iterations constant to 10, in contrast to the findings in figure 4.5, so we can not directly compare the results. We interpret increase in execution time as linear, while we interpret the decrease in loss also as linear. This is in contrast to the parallel implementation, which we discuss later.

4.2.2. Parallel L-BFGS-B

In this part we analyze the results of our implementation and experimentation with the parallel L-BFGS-B algorithm. As discussed in 3.5.3, we have multiple options in implementing this method. We show the effect of a number of changes in the implementation of the algorithm.

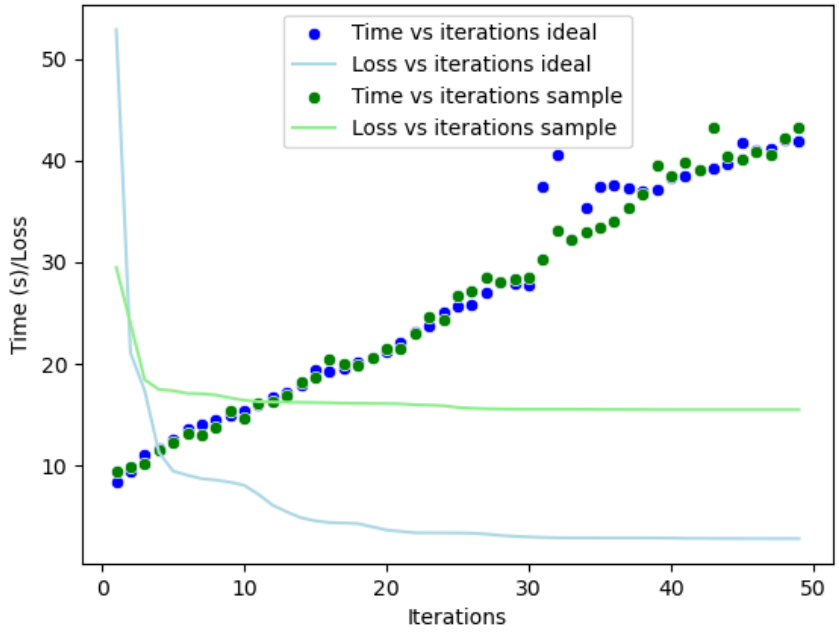


Figure 4.4.: *Plot of the parallel Quasi Newton method showing execution time and loss vs the number of iterations*

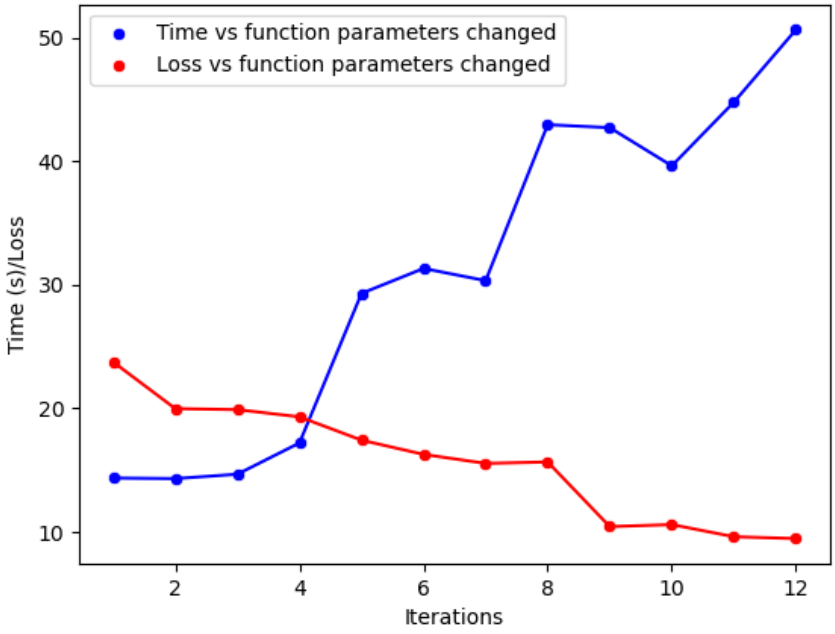


Figure 4.5.: *Plot of the Quasi Newton method showing execution time and loss vs the number of principal components changed*

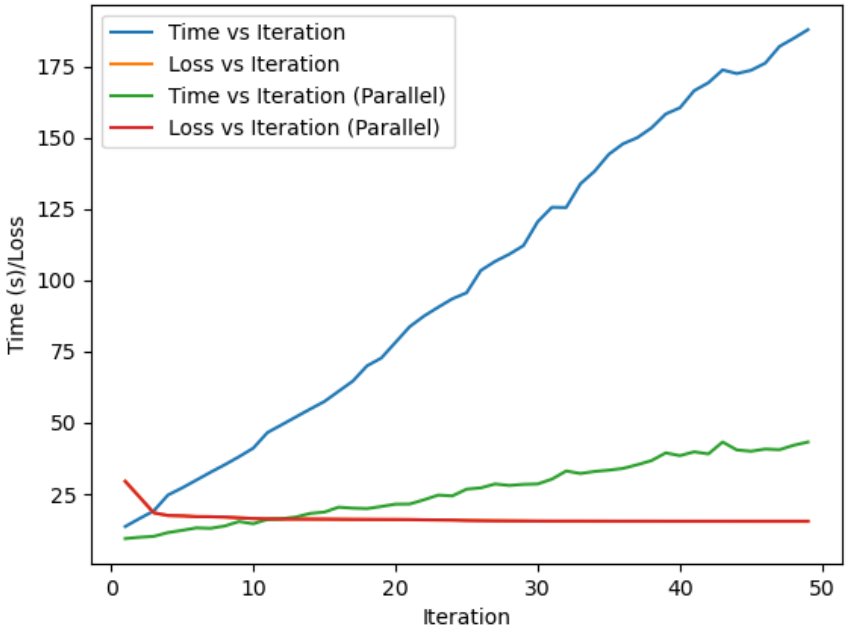


Figure 4.6.: *Plot of the Quasi Newton method showing execution time and loss vs the number of principal components changed in the serial and parallel implementation*

Figure 4.4 shows the execution time and loss of the parallel Quasi Newton method vs the number of iterations. We also have two models and corresponding objective lines to show the difference between a real world application, meaning a user drawn line, and a line which is a perfect fit to the model. 3.13 shows the exact model and line we used for our measurements. We see a very steep drop in the loss in the first few iterations, which is to be expected. This means that if we had used this implementation we would have been able to achieve a very reasonable result with just a few iterations.

Figure 4.5 shows the execution time and loss of the parallel Quasi Newton method vs the number of principal components changed. We kept the number of iterations constant to 30 As expected, the execution time goes up, the more principal components we change. This is because the number of parameters we have to optimize increases, and thus the number of function evaluations increases. The loss also decreases as we increase the number of principal components changed, which is to be expected.

In contrast to 4.3, the way how the execution time increases is not as clear to see. But what we can see is that we hit a plateau when we change seven parameters. As discussed in 3.5.3, this is expected. If we wanted to implement our correction UI with the parallel L-BFGS-B algorithm, we would choose to change seven parameters.

Figure 4.6 shows the execution time and loss of the parallel and serial Quasi Newton method vs the number of iterations in the parallel and the serial implementation. We can clearly notice, that the difference between the losses is very small, while the difference in execution time is very large. This is not very surprising, as we expect a constant speedup, as mentioned in 3.5.3.

4.2.3. Newton's Method

Now we discuss our actual implementation, which uses Newton's method. We primarily focus on the number of iterations in conjunction with the loss, meaning the function to minimize. For an explanation of our implementation refer to 3.5.3 and 3.5.4. We note here that the execution time of our Newton method is very small compared to the execution time of the parallel L-BFGS-B algorithm.

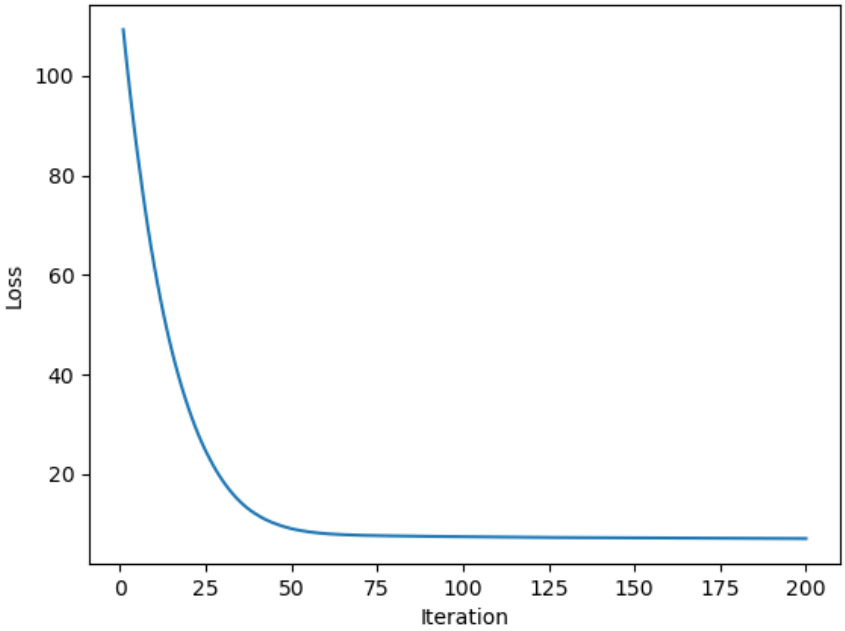


Figure 4.7.: *Plot showing losses vs iterations*

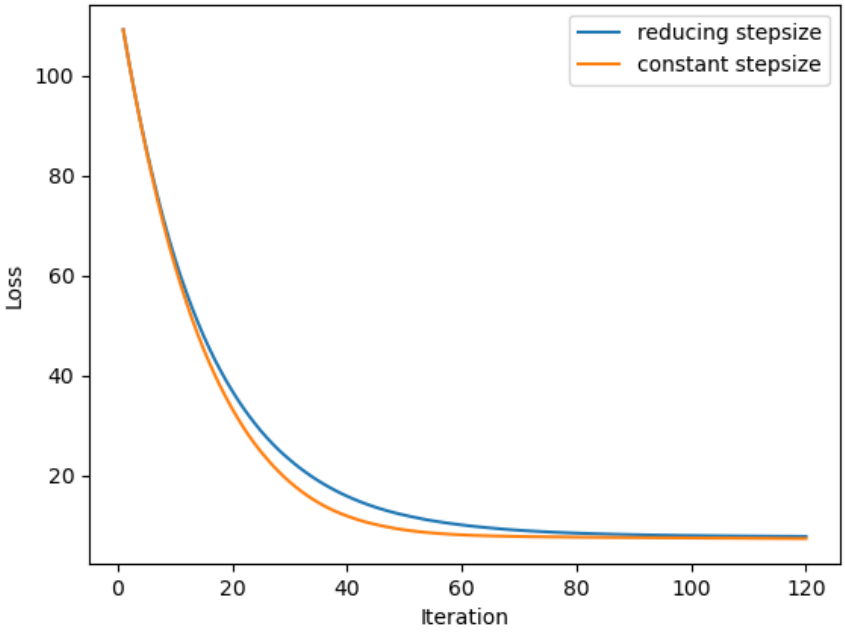


Figure 4.8.: *Plot showing losses vs iterations with and without a reduction of the step size*

Figure 4.7 shows the loss vs the number of iterations of our implementation of Newton's method. We see the loss decreasing very quickly in the first few iterations. It is important

to note that here a single iteration only takes a few milliseconds, so here 200 iterations are a lot quicker than 50 iterations of the parallel L-BFGS-B algorithm. In our implementation we chose to stop the optimization after 100 iterations.

In figure 4.8 we show the loss vs the number of iterations with and without a reduction of the step size. Here the step size is multiplied by 0.99 after every iteration, and starts with one, meaning in the first iteration we go in the negative direction of the gradient equal to the gradient. We observe that the loss decreases slower than without the reduction of the step size, but after 90 iterations we have more or less the same loss again. This is why we didn't use a reduction of the step size in our implementation.

Execution Time of our Implementation

As described in 3.5.4 the execution time does not depend on the number of parameters we change, but only on the number of iterations, and the time it takes to calculate the minimal difference between the breast and the line. Since the number of iterations is constant, we only have to look at the time it takes to calculate the difference. Here the more point the line has, meaning the more point the line has, the longer it takes to calculate the difference.

In figure 4.9 one can see the execution time vs the length of the list. Here we clearly see a linear behavior. What also can be seen is that the execution time is very small as displayed in Table 4.1.

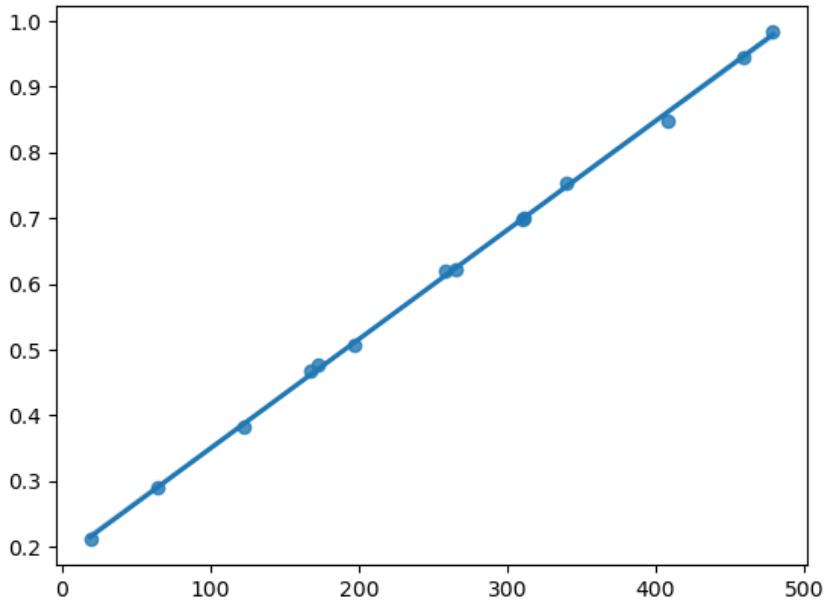


Figure 4.9.: *Plot showing the execution time vs the length of the line*

Length of the line	Execution time in seconds
19	0.211901
64	0.29
123	0.3821
167	0.4668
172	0.4764
197	0.507099
258	0.6202
265	0.623
310	0.6987
311	0.7006
340	0.7527
408	0.8482
459	0.9456
479	0.9836

Table 4.1.: *Execution time of our implementation of Newton's method and the length of the line*

5

Conclusion and Outlook

5.1. Conclusion

This thesis has introduced a methodology for the alteration of 3D models within the realm of cross-platform applications. Through the development of a prototype application, we have successfully showcased the practicality and effectiveness of our approach in enabling users to manipulate 3D models through various means.

Additionally, we introduce a breast modification technique utilizing a corrective user interface (UI), designed to facilitate effortless and anatomically accurate adjustments to a 3D model. This approach caters to inexperienced users, providing them with an intuitive interface for modifying the model in a manner that adheres to anatomical principles.

By harnessing techniques from the domains of computer graphics and data processing, and leveraging the capabilities of the Flutter framework, in combination with the existing pipeline developed by Arbrea Labs, we have successfully developed and presented our prototype application.

5.2. Outlook and Future Work

The prototype application presented in this thesis is a proof of concept, and as such, there is room for improvement and further development.

In a future work we would like to implement a method, which allows a user to change each breast individually.

As discussed in 3.5 we needed to simplify the problem to get to our solution. In a future work one can implement a more sophisticated method to get the silhouette of the breast. This

approach would necessitate more changes to the whole minimization process, which would undo our simplifications. One might also need to change the algorithm used for minimizing our loss function, because this loss function would look different.

These changes could further allow the user of our applications to not only change the breast from the side view, but also from every possible view angle, since we now don't rely on predefined vertices anymore. Such a feature rich implementation would certainly be a desired feature for any surgeon and patient.

Another desired feature could be a complete offline version of the application, eliminating the need for a server. This could be done by implementing the whole pipeline on the device itself. How useful this would be isn't part of this work, but it would be a considerable topic for a future work.

A

Appendix

A.1. Libraries and existing Code

A.1.1. Flutter

These are all Flutter packages that were used in the project.

- **camera** version: 0.10.3+2
  `https://pub.dev/packages/camera`
- **camera_web** version: 0.3.1+2
  `https://pub.dev/packages/camera_web`
- **dio** version: 5.0.3
  `https://pub.dev/packages/dio`
- **http_parser** version 4.0.2
  `https://pub.dev/packages/http_parser`
- **image** version: 4.0.15
  `https://pub.dev/packages/image`
- **image_utils_class** version: 2.0.0
  `https://pub.dev/packages/image_utils_class`
- **ml_linalg** version: 13.11.30
  `https://pub.dev/packages/ml_linalg`
- **file_picker** version: 5.2.11
  `https://pub.dev/packages/file_picker`

- **dart_numerics** version: 0.0.6
  https://pub.dev/packages/dart_numerics
- **flutter_dropzone** version: 3.0.5
  https://pub.dev/packages/flutter_dropzone
- **universal_html** version: 2.2.2
  https://pub.dev/packages/universal_html

As mentioned through this thesis we tailored the Cube package to our needs.

- **flutter_cube** version: 0.1.1
  https://pub.dev/packages/flutter_cube

A.1.2. Python

These are all Python packages that were used in the project.

- **numpy** version: 1.21.2
  https://pypi.org/project/numpy/
- **seaborn** version: 0.12.2
  https://pypi.org/project/seaborn/
- **optimparallel** version: 0.1.2
  Refer to [Ger20] and [GF19] or
  https://pypi.org/project/optimparallel/
- **pandas** version: 1.5.3
  https://pandas.pydata.org/
- **scipy** version: 1.10.0
  https://scipy.org/
- **sklearn** version: 1.2.1
  https://scikit-learn.org/stable/
- **vedo** version: 2023.4.4
  https://vedo.embl.es/
- **h5py** version: 3.7.0
  https://www.h5py.org/
- **PIL** version: 9.4.0
  https://pypi.org/project/Pillow/

A.2. Performance Tests

All experiments were conducted on a machine running Windows 10 with 32 GB RAM, an Intel® Core™ i7-9700K CPU and an NVIDIA GeForce RTX 2070.

Bibliography

[Ger20] Florian Gerber. florafauna/optimparallel-python: test zenodo, June 2020.

[GF19] Florian Gerber and Reinhard Furrer. optimParallel: An R Package Providing a Parallel Version of the L-BFGS-B Optimization Method. *The R Journal*, 11(1):352–358, 2019.

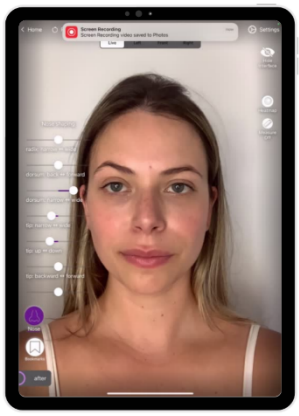
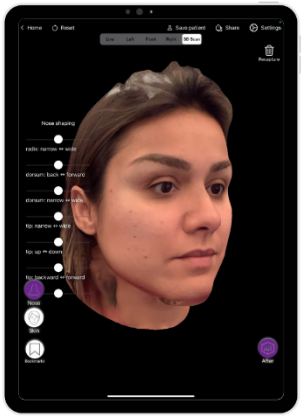
[RSM08] Michal Rychlik, Witold Stankiewicz, and Marek Morzyński. Applications of 3d pca method for extraction of mean shape and geometrical features of biological objects set. *Mathematical Modelling and Analysis*, 13(3):413–420, 2008.

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

computer graphics laboratory

Bachelor Thesis

Cross-platform Real-time Visualization and Editing Interface of Breast 3D meshes

Introduction

Arbrea Labs builds **AR&3D** surgery simulators that have revolutionized patient-surgeon visual communication during consultations, and are a game-changer in aesthetic medicine and plastic surgery. Integral technologies to such tools vary from 3D Reconstruction and AR/VR to Neural Rendering and Physic Simulations. In this thesis we will focus on visualization and editing outside of iOS devices.

Task Description

The bachelor thesis consists of the following steps:

- Get acquainted with the relevant literature
- Render a mesh on the web and in an app with flutter
- Implement PCA and blend shape based editing of the meshes
    - Train a PCA model with the data provided by Arbrea
    - Analyze the params and extract a direction related to the breast size
- Build a visualization of the differences between two meshes
- Link the app to a backend “pipeline” that builds the mesh from a set of input images
- Build a fit correction UI that lets the user edit masks and breastlines to get a new 3D mesh and do a usability test/apply to various cases
- (Bonus) Implement AR rendering

Remarks

A written report and an oral presentation conclude the thesis. The thesis will be overseen by Prof. Markus Gross and supervised by Dr Endri Dibra and Dr Beren Kaul.

Contact

For further information, please contact (endri.dibra@arbrea-labs.com) or
the CGL thesis coordinator (cgl-thesis@inf.ethz.ch)

Arbrea Labs

ETH

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

Cross-platform Real-time Visualization and Editing Interface of Breast 3D meshes

Authored by (in block letters):

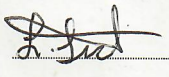
For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**
Sichi	Luca

With my signature I confirm that

- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**
Brugg, 01.03.2023	

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.