Thesis Title

Luca Sichi

Bachelor Thesis
June 2023

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. To achieve this, the Flutter framework is utilized, which facilitates cross-platform compilation. The primary target platforms for this application are web and Android.

Zusammenfassung

Diese Arbeit nutzt Computergrafik und Computer Vision, um Bilder des Oberkörpers zu erfassen, die es dem Benutzer ermöglichen, die Brüste zu verändern und das neue Modell zur Bewertung einer möglichen Operation anzuzeigen. Das Ziel besteht darin, einen Code-Basis für mehrere Plattformen zu verwenden. Hierfür wird das Flutter-Framework eingesetzt, das eine Kompilierung für verschiedene Plattformen ermöglicht. Die Hauptzielplattformen dieser Anwendung sind das Web und Android.

Contents

List of Figures

List of Tables

1 Introduction

1.1. Background

The main goal of this thesis is to develop a cross-platform application that allows users to modify the appearance of the breasts and evaluate the potential outcomes of a surgical procedure. The application is intended to be used by plastic surgeons and their patients.

One of the primary objectives is to incorporate similar functionalities as the **Lind App**, created by Arbrea Labs, into the application. The aim is to establish a unified codebase capable of operating on various platforms, with a particular focus on the web and Android.

1.2. Problem Statement

2

Related Work

Sample references are [ZRB+04] and [Alt89].

2.1. Displaying 3D Models

A pivotal aspect of this thesis revolves around the presentation of 3D models. Given the intended deployment of the application across diverse platforms, the implementation necessitates a cross-platform approach. To address this requirement, we derive inspiration from webGL and openGL, adapting their principles to develop a custom solution. Notably, a substantial portion of the rendering process is executed through the utilization of the Cube Flutter plugin, which we have tailored to align with our specific needs.

2.2. Modification of 3D Models

To facilitate the alteration of the 3D models, an adapted approach is employed, drawing inspiration from the methodology presented in the scholarly work by **Zwicker et al. (2004) titled "Perspective Shape from Shading."**

Wavefront OBJ files serve as the file format for storing the 3D models employed in this study. To facilitate their integration with the rendering engine, a parsing mechanism is implemented to extract the necessary geometric and texture information from the OBJ files. Subsequently, a conversion process is executed to transform the extracted data into a compatible format that can be seamlessly rendered by the rendering engine.

3

Methods

This chapter describes the

3.1. Image Acquisition

The initial phase involves the acquisition of patient data. For our specific objectives, a triad of torso images is essential, comprising a frontal view, a left lateral view, and a right lateral view. These images provide comprehensive visual information required for subsequent 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 entails 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 caters 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 comprehensive range of image acquisition widgets is available, including the ImagePicker and Camera widgets. The ImagePicker widget empowers users to select an image from the gallery of the device, while the Camera widget facilitates 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, thus enhancing the flexibility and functionality of our application.

At this stage, we confront a challenge pertaining to platform-specific code. The web application, being browser-based, presents a hurdle in terms of accessing the device's memory, which poses inherent difficulties. In contrast, the Android platform offers seamless access to the device's memory without encountering similar 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 divece'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 stringent 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 streamlines 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 satisfactory 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. **TODO: Add image of ImagePicker widget and drag and drop**

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, establishing a connection 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, employs the received data to invoke the pipeline, 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 between 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 concluding with 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. Visualusation of the 3D Model

As previously discussed, the visualization of the 3D model is primarily facilitated by the integration of the Flutter package Cube. This package encompasses a 3D model viewer, which incorporates 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 applycation, 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.

TODO: Add image of the two models

Within the object.dart file, one can locate the presence of 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 a significant but nuanced aspect, where 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. This particular detail warrants consideration when attempting to manipulate the vertices of the model at a later stage.

The Cube package furnishes us with the essential transformation matrices, specifically the model, view, and projection matrices. These matrices are of utmost 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. Numerous options exist 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 facilitate the 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. Subsequently, 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.:

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

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

TODO: This section needs to be rewritten.

After performing the Principal Component Analysis (PCA) in Python, we needed to bring this functionality to our Flutter application. For this we wrote the mean μ, standard deviations $\sigma_1 ... \sigma_k$ and eigen vectors $v_1 ... v_k$ to seperate files. As the number of principal components k we decided on the number 30. In our Flutter Application we then read these files into memory for further usage.

As we wanted to be able to deconstruct and reconstruct the 3D models in real time, we needed to be able to perform the operations in Flutter. To do this we first needed to be able to perform the matrix multiplications needed in reasonable time. For this we used the ml_linalg library, which provides a matrix and a vector class, as well as the support of SIMD instructions. This allowed us to rely on ml_linalg for fast matrix multiplication.

As a first step, after recieving the 3D model from the server, we deconstruct it to get the 30 principal components, according to (3.1). We then save the vector λ_k two times, one vector to be used for reconstructing a modified model, and one vector to be used for reconstructing the original model as a reference. we then proceed to reconstruct the model $\hat{w}$ according to (3.2), using λ_k.

For changing the model, we give the user the ability to use slider to change the value of the specific principal components explained below. We allow the user to change a eigenvalue λ_i by $\pm\ 3\sigma_i$, so we can cover 99.73% of the data, with respect to the PCA fitted on our data. Then every time the user changes a value with a slider, we reconstruct the model as described above.

TODO: Add images of sliders

3.4.2. Analysis of the principal components

First principal component

Upon 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.

TODO: Add images of the first principal component

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.

TODO: Add images of the second principal component

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

TODO: Add images of the third principal component

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.

TODO: Add images of the fourth principal component

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.

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.

TODO: Add images of the torso deformation

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 components 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.

TODO: Add images of the breast area

Upon 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 $\hat{w}$.

$$v_i = \frac{d_i}{d}\hat{w}' + \left(1 - \frac{d_i}{d}\right)\hat{w} \tag{3.4}$$

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.

3.5.1. Problem specification

To achieve the desired functionality, we first needed to define our problem in mathematical terms. The easiest way to achieve this is to consider the problem as a minimization problem, where we try to minimize the distance between the drawn line and the modified breast model. We knew the goal was to solve a problem similar to this:

$$\arg\min_{\lambda_k} = \|f(line) - s(g(\lambda_k))\| \tag{3.5}$$

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

But also here we stand before multiple questions. For minimizing this distance we need to know $line$ f, g and s. Since we know how to reconstruct the 3D model $\hat{w}$ from PCA parameters λ_k, we know $g(\lambda_k)$. Getting the $line$ drawn by the user also is not a problem, because Flutter provides us with the Gesture Detector widget, which provides us with the coordinates of the latest touch or mouse events. We can just save these coordinates in a list for later 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 shilouette of the 3D model stays on a constant x coordinate. This means that the line defined by the shilouette 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 shilouette to a plane, but instead we can just take the distance from the shilouette 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.

The idea behind our approach is to reverse project the 2D points in image space to a line in 3D space, and then intersect these lines with the same yz plane as the shilouette to get a list of points in 3D space.

The Cube implementation uses a standard graphics pipeline to project the 3D model coorinates 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 backprojection 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 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:

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

We now have two points, p_{I1} at the near plane of the frustum, and p_{I2} at the far plane of the frustum. 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, lets define $s := p_{M1}$ and $t := p_{M2}$, 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 (3.5). 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 a 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 algortithm 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 convergance 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 stepsize m of the algorithm. This means that the algorithm will make smaller steps and we can even lower the stepsize 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 you can see it looks similar to (3.5)

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

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 further simplified our function $f(x)$ by only considering principal components 2 to 8 as input to our function f. We did this 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. 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.

TODO: cite paper

Deep Reinformcement Learning

As another idea we had Deep Reinformcement Learning in our mind. This of course is a very diffe

3.5.4. Our Implementation

4

Evaluation

4.1. PCA

4.2. Correction UI

5

Conclusion and Outlook

5.1. Conclusion

5.2. Outlook and Future Work

A

Appendix

Nein, meine Texte les ich nicht, so nicht, stöhnte Oxmox. Er war mit Franklin, Rockwell und dem halbtaxgrauen Panther Weidemann in Memphis (Heartbreak Hotel) zugange. Sie warteten auf die fette Gill, um bei der Bank of Helvetica die Kapitälchen in Kapital umzuwandeln. Oxmox liess nicht locker. Ich fleh euch an, rettet meine Copy, gebt meinem Body nochn Durchschuss! Kein Problem, erbarmte sich Old Face Baskerville, streichelte seinen Hund, zog seine einspaltige Poppl, legte an und traf! (Zeidank nichts Ernstes — nurn bisschen Fraktur.) Oxmox: Danke, ist jetzt mit Abstand besser. Derweil jumpte der Fox leise over the Buhl, die sich mal wieder immerdar wie jedes Jahr gesellte. Diesmal war Guaredisch ihr Erwählter, weil seine Laufweite einem vollgetankten Bodoni entsprach und seine ungezügelte Unterlänge ihre Serifen so serafisch streifte, dass sie trotz Techtelmechtelei die magere Futura, jene zuverlässige und gern eingesetzte Langstreckenläuferin, rechtsbündig überholen konnten.

A.1. Foo Bar Baz

Nein, meine Texte les ich nicht, so nicht, stöhnte Oxmox. Er war mit Franklin, Rockwell und dem halbtaxgrauen Panther Weidemann in Memphis (Heartbreak Hotel) zugange. Sie warteten auf die fette Gill, um bei der Bank of Helvetica die Kapitälchen in Kapital umzuwandeln. Oxmox liess nicht locker. Ich fleh euch an, rettet meine Copy, gebt meinem Body nochn Durchschuss! Kein Problem, erbarmte sich Old Face Baskerville, streichelte seinen Hund, zog seine einspaltige Poppl, legte an und traf! (Zeidank nichts Ernstes — nurn bisschen Fraktur.) Oxmox: Danke, ist jetzt mit Abstand besser. Derweil jumpte der Fox leise over the Buhl, die sich mal wieder immerdar wie jedes Jahr gesellte. Diesmal war Guaredisch ihr Erwählter, weil seine Laufweite einem vollgetankten Bodoni entsprach und seine ungezügelte Unterlänge ihre Serifen so serafisch streifte, dass sie trotz Techtelmechtelei die magere Futura, jene zuverlässige und gern eingesetzte Langstreckenläuferin, rechtsbündig überholen konnten.

A.2. Barontes

Nein, meine Texte les ich nicht, so nicht, stöhnte Oxmox. Er war mit Franklin, Rockwell und dem halbtaxgrauen Panther Weidemann in Memphis (Heartbreak Hotel) zugange. Sie warteten auf die fette Gill, um bei der Bank of Helvetica die Kapitälchen in Kapital umzuwandeln. Oxmox liess nicht locker. Ich fleh euch an, rettet meine Copy, gebt meinem Body nochn Durchschuss! Kein Problem, erbarmte sich Old Face Baskerville, streichelte seinen Hund, zog seine einspaltige Poppl, legte an und traf! (Zeidank nichts Ernstes — nurn bisschen Fraktur.) Oxmox: Danke, ist jetzt mit Abstand besser. Derweil jumpte der Fox leise over the Buhl, die sich mal wieder immerdar wie jedes Jahr gesellte. Diesmal war Guaredisch ihr Erwählter, weil seine Laufweite einem vollgetankten Bodoni entsprach und seine ungezügelte Unterlänge ihre Serifen so serafisch streifte, dass sie trotz Techtelmechtelei die magere Futura, jene zuverlässige und gern eingesetzte Langstreckenläuferin, rechtsbündig überholen konnten.

A.3. A Long Table with Booktabs

Table A.1.: *A sample list of words.*

ID	Word	Word Length	WD	ETL	PTL	WDplus
1	Eis	3	4	0.42	1.83	0.19
2	Mai	3	5	0.49	1.92	0.19
3	Art	3	5	0.27	1.67	0.14
4	Uhr	3	5	0.57	1.87	0.36
5	Rat	3	5	0.36	1.71	0.14
6	weit	4	6	0.21	1.65	0.25
7	eins	4	6	0.38	1.79	0.26
8	Wort	4	6	0.30	1.62	0.20
9	Wolf	4	6	0.18	1.54	0.19
10	Wald	4	6	0.31	1.63	0.19
11	Amt	3	6	0.30	1.67	0.14
12	Wahl	4	7	0.36	1.77	0.42
13	Volk	4	7	0.45	1.81	0.20
14	Ziel	4	7	0.48	1.78	0.42
15	vier	4	7	0.38	1.81	0.42
16	Kreis	5	7	0.26	1.62	0.33
17	Preis	5	7	0.28	1.51	0.33
18	Re-de	4	7	0.22	1.56	0.33
19	Saal	4	7	0.75	2.10	0.43
20	voll	4	7	0.48	1.82	0.24
21	weiss	5	7	0.21	1.59	0.36
22	Är-ger	5	7	1.16	2.69	0.59

continued on next page

Table A.1.: *(Continued)*

ID	Word	Word Length	WD	ETL	PTL	WDplus
23	bald	4	7	0.18	1.56	0.19
24	hier	4	7	0.40	1.70	0.43
25	neun	4	7	0.17	1.52	0.26
26	sehr	4	7	0.36	1.85	0.43
27	Jahr	4	7	0.50	1.82	0.43
28	Gold	4	7	0.04	1.35	0.20
29	Tä-ter	5	8	0.15	1.39	0.59
30	Tei-le	5	8	0.30	1.71	0.46
31	Na-tur	5	8	0.18	1.59	0.41
32	Feu-er	5	8	0.30	1.71	0.45
33	Rol-le	5	8	0.15	1.46	0.45
34	Rock	4	8	0.29	1.68	0.25
35	Spass	5	8	0.28	1.64	0.32
36	Gäs-te	5	8	0.49	1.75	0.66
37	En-de	4	8	0.36	1.72	0.33
38	Kunst	5	8	0.26	1.59	0.35
39	Li-nie	5	8	0.45	1.88	0.63
40	Bäu-me	5	8	0.48	1.92	0.45
41	Büh-ne	5	9	0.94	2.48	0.62
42	Bahn	4	9	0.21	1.62	0.42
43	Bür-ger	6	9	0.38	1.70	0.65
44	Druck	5	9	0.60	2.03	0.31
45	zehn	4	9	0.41	1.84	0.42
46	Va-ter	5	9	0.36	1.78	0.40
47	Angst	5	9	0.29	1.56	0.35
48	lei-der	6	9	0.13	1.47	0.52
49	häu-fig	6	9	0.82	2.31	0.52
50	le-ben	5	9	0.38	1.85	0.40
51	aus-ser	6	9	1.20	2.26	0.57
52	be-vor	5	9	1.28	2.75	0.39
53	Kai-ser	6	9	0.92	2.37	0.53
54	Markt	5	9	0.23	1.58	0.28
55	Os-ten	5	9	0.21	1.54	0.48
56	Krieg	5	9	0.33	1.67	0.50
57	Mann	4	9	0.31	1.47	0.25
58	Hal-le	5	9	0.24	1.65	0.45
59	heu-te	5	9	0.44	1.87	0.46
60	in-nen	5	10	0.36	1.80	0.45
61	Na-men	5	10	0.28	1.72	0.41
62	jetzt	5	10	0.70	2.07	0.32
63	kei-ner	6	10	0.28	1.62	0.53

continued on next page

Table A.1.: *(Continued)*

ID	Word	Word Length	WD	ETL	PTL	WDplus
64	Schu-le	6	10	1.02	2.12	0.48
65	Ar-beit	6	10	0.34	1.70	0.52
66	An-teil	6	10	0.27	1.63	0.53
67	di-rekt	6	10	0.67	2.04	0.47
68	vor-her	6	10	0.78	2.25	0.47
69	wol-len	6	10	0.44	1.85	0.51
70	Kampf	5	10	0.70	1.96	0.27
71	än-dern	6	10	1.18	2.62	0.65
72	lau-fen	6	10	0.21	1.64	0.52
73	Eu-ro-pa	6	10	0.23	1.53	0.66
74	statt	5	10	1.61	2.86	0.39
75	Wes-ten	6	10	0.29	1.60	0.54

Bibliography

[Alt89] Simon L. Altman. Hamilton, grassmann, rodrigues, and the quaternion scandal—what went wrong with one of the major mathematical discoveries of the nineteenth century? *A Mathematical Association of America journal*, dec 1989.

[ZRB$^+$04] Matthias Zwicker, Jussi Räsänen, Mario Botsch, Carsten Dachsbacher, and Mark Pauly. Perspective accurate splatting. In *Proceedings of Graphics Interface*, pages 247–254, 2004.