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.

Inside the object.dart file we can find a mesh object, which is defined in the mesh.dart file. This mesh object contains a list of vertices, polygons, normals and other information about the model, used to visualize it. A minor but important detail is that there is a function, that duplicates vertices that have multiple texture coordinates, so that each vertex has only one texture coordinate. We need to keep this in mind when we later want to change the vertices of the model.

The Cube package also provides us with the transformation matrices, namely the model, view and projection matrices. We need to access these matrices later, so knowing that they can be found in scene.dart simplifies our later work.

3.4. Modification of the 3D Model through PCA

3.4.1. Principal Component Analysis

3.4.2. Region specific modification

3.5. Model modification through Correction UI

3.5.1. Problem specification

3.5.2. Methods

3.5.3. Newton's method

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.