Abteilung: Höhere Lehranstalt für Informatik

Diplomarbeit

Höhere Abteilung für Informatik

Thema:	AnthroMorph - App zur anthropometrischen Visualisierung
eingereicht von:	Johannes Schober <j.schober00@gmail.com> Witold Dobersberger <witold.dobersberger@gmx.at>
eingereicht am:	29. September 2016
Betreuer:	Prof. Dipl.-Ing. Dr. Michael Buchberger
In Zusammenarbeit mit:	Schober Stephan BSc

Eidesstattliche Erklärung

Die unterfertigten Kandidaten / Kandidatinnen haben gemäß § 34 (3) SchUG in Verbindung mit § 22 (1) Zi. 3 lit. b der Verordnung über die abschließenden Prüfungen in den berufsbildenden mittleren und höheren Schulen, BGBl. II Nr. 70 vom 24.02.2000 (Prüfungsordnung BMHS), die Ausarbeitung einer Diplomarbeit mit der umseitig angeführten Aufgabenstellung gewählt.

Die Kandidaten / Kandidatinnen nehmen zur Kenntnis, dass die Diplomarbeit in eigenständiger Weise und außerhalb des Unterrichtes zu bearbeiten und anzufertigen ist, wobei Ergebnisse des Unterrichtes mit einbezogen werden können.

Die Abgabe der Diplomarbeit hat bis spätestens 05.04.2017 beim zuständigen Betreuer / der zuständigen Betreuerin zu erfolgen.

Die Kandidaten / Kandidatinnen nehmen weiters zur Kenntnis, dass gemäß § 9 (6) der Prüfungsordnung BMHS nur der Schulleiter bis spätestens Ende des vorletzten Semesters den Abbruch einer Diplomarbeit anordnen kann, wenn diese aus nicht beim Prüfungskandidaten (bei den Prüfungskandidaten) gelegenen Gründen nicht fertiggestellt werden kann.

Kandidaten / Kandidatinnen inkl. Unterschrift:

Ort, Datum	Johannes Schober
Ort, Datum	Witold Dobersberger

Danksagung
Wir bedanken uns bei all jenen, die uns im Rahmen dieser Diplomarbeit zur Seite gestanden sind.

Ganz besonders möchten wir uns bei Herrn Prof. Dipl.-Ing. Dr. Michael Buchberger bedanken, der uns mit seiner fachlichen Unterstützung begleitet hat. Seine Erfahrung beim Schreiben von wissenschaftlichen Arbeiten hat uns sehr weitergeholfen.

Darüber hinaus bedanken wir uns bei Herrn Stephan Schober BSc, welcher uns mit seiner langjährigen Erfahrung als Entwickler von Android Anwendungen kompetent unterstützt hat und fast zu jeder Zeit des Tages Zeit hatte für unsere Fragen.

Kurzfassung

Dieses Projekt hat als Ziel eine benutzerfreundliche, mobile Applikation hervorzubringen. Sie soll sowohl die Möglichkeit einer aufregenderen Art der Erstellung einer Fotoserie sowie auch eine einfach zu bedienende Variante eines Morphing-Prozesses, der auch mehr als zwei Bilder verarbeiten kann, bieten. Es gibt bereits für verschiedene Plattformen mobile Anwendungen, die es dem Benutzer erlauben, Fotos aufzunehmen und diese in einer Fotoserie als Diashow oder Video hintereinander abzuspielen. Gleichzeitig dazu existieren bereits Webseiten, die es einer Person ermöglichen, ein Video oder ein GIF zu erstellen, bei dem ein Bild einen flüssigen, durch Verschmelzung gewährleisteten Übergang in ein anderes Bild durchgeht (Morphing-Prozess). Das Problem war nun einerseits, dass die mobilen Anwendungen zur Erstellung einer Fotoserie zwar benutzerfreundlich gestaltet waren, aber schnell den Reiz verloren, wenn man sie über längeren Zeitraum benutzt hat. Da die anwendende Person genau gewusst hat, was sie erwartet. Andererseits sind die Webseiten die einen Morphing-Prozess zur Verfügung stellen, meist schwer aufzufinden, da sie keine sehr sprechenden Namen besitzen und zusätzlich oft mühsam zu bedienen sind.

Abstract

This project aims to create a user-friendly mobile application, that grants the frequent ability to create a series of photos and additionally contributes an easy-to-use version of a morphing process that can process more than two photos. There are already existing mobile applications on different platforms for taking photos, creating series of photos and visualizing them in a video or a GIF. At the same time there are some websites providing the user the possibility to make a video where a picture smoothly transitions into another picture, guaranteed by warping the pictures (morphing process). The problem was that on the one hand the mobile applications, making it possible to create a series of photos, loose their appeal quickly when used over a long period of time, because the user already knows what he/she can expect from it. On the other hand the websites providing the morphing process are either not that easy to find because they have no descriptive names or are lacking an ease of use.

Inhaltsverzeichnis

1 Aufgabe

1.1 Aufgabenstellung und Erfüllung

Aufgabenstellung war es eine native, mobile Anwendung zu entwickeln, welche den Benutzer täglich erinnert ein Foto einer Person oder eines Gegenstands aufzunehmen, wobei die Kamerafunktion in der Anwendung integriert sein soll. Diese Fotoserien werden von der Anwendung gespeichert und verwaltet.

Fotos sollen nach der Aufnahme in sozialen Netzwerken oder in Chats geteilt werden können.

Die Anwendung soll aus den gespeicherten Fotoserien automatisch Videodateien erzeugen und als Morphingprozess darstellen können. Zusätzlich gibt es eine Kamerafunktion, welche den Umriss vom letzten Foto dieser Fotoserie transparent anzeigt, so dass am Ende die gewünschte Person oder der gewünschte Gegenstand immer an derselben Stelle im Video sein soll.

Ausgangslage

Für verschiedene mobile Plattformen existieren bereits Anwendungen mit denen Fotoserien aufgenommen werden können, diese enthalten aber keine Morphing Funktion.

Erfüllung der Aufgaben

Im Laufe der Diplomarbeit hat sich herausgestellt, dass die Morphing Funktion eine größere Herausforderung ist als geplant. Dadurch wurde die eingeplante Zeit für die iOS App für den Morphingprozess aufgewandt und dieser in eine Android App inkludiert, wobei die Android App alle gestellten Aufgaben erfüllt.

2 Grundlagen

2.1 Web mobile Anwendungen

Web Anwendungen werden mit den Technologien HTML5, JavaScript und CSS implementiert und wie native Anwendungen auf dem Gerät installiert. Wenn die App dann ausgeführt wird, startet im Hintergrund nicht eine kompilierte Anwendung, sondern der Webbrowser des Geräts. Da die Anwendung indirekt über den Webbrowser läuft, kann sie auf jedem beliebigen Betriebssystem ausgeführt werden. Die Entwicklungskosten sind niedriger als bei nativen Anwendungen, da der Quelltext nur einmal implementiert werden muss, um alle Plattformen abzudecken.

HTML5 ermöglicht auch eine Offline-Speicherung der Anwendung und damit den Gebrauch ohne verfügbaren Internetzugang. Die Veröffentlichung von Web Anwendungen geschieht über einen Web-Server und kann deshalb in sekundenschnelle und ohne das Prozedere, das sonst in einem App Store durchlaufen werden muss, von statten gehen.

2.2 Hybride mobile Anwendungen

Eine hybride Anwendung ist eine Web Anwendung welche mithilfe eines Frameworks, wie etwa Apache Cordova oder Corona, mit einen nativen Container ummantelt wird. Dabei muss die Anwendung auf dem Gerät installiert werden und verfügt dadurch über Gerätefunktionen, welche mit Web Anwendungen normalerweise nicht verwendet werden können. Außerdem muss die Anwendung nicht für jede Plattform extra implementiert werden, sondern sie läuft auf jedem Betriebssystem mithilfe von HTML5 und JavaScript. [1]

2.3 Native mobile Anwendungen

Native Anwendungen, auch 'Apps' genannt, werden in einer bestimmten Programmiersprache für ein bestimmtes Betriebssystem entwickelt z.B.: der Quelltext einer Anwendung für Android ist meistens in Java und das Layout mit XML implementiert. Der Quelltext kann allerdings auch in der Programmiersprache C oder C++ implementiert werden, wie es der Fall bei Android NDK ist[2]. Dadurch ergeben sich einige Vorteile gegenüber Web Apps und Hybriden Apps.

Zum Einen sind native Anwendungen für ein Betriebssystem optimiert und können daher die Hardware des Gerätes effektiv nutzen. Außerdem können native Anwendungen, wie auch Hybride, beliebig große Daten auf dem Gerät speichern und damit den reibungslosen Betrieb ohne aktiver Internetverbindung garantieren.

App Stores bieten nativen Anwendungen eine einfache Plattform zum Vertreiben der eigenen Anwendungen und bieten ein bewährtes Bewertungssystem und je nach Erfolg der Anwendung, auch einen Platz in Top-Listen der Stores und somit eine gratis Vermarktung. [3]

Die verwendete Form

Da der im selben Kapitel beschriebene Morphingprozess eine sehr hohe Rechenleistung erfordert, wurde hier entschieden die Lösung in Form einer nativen Anwendung zu realisieren. Außerdem verbrauchen Bilddateien immer mehr Speicher, da die Qualität der aufgenommenen Bilder immer besser wird. Daher werden beim Morphing Algorithmus große Datenmengen bearbeitet, was in einer nativen App schneller abgewickelt werden kann. Weiters startet eine native App schneller als Web- oder Hybride Apps. Dieser Aspekt ist sehr wichtig, da die Anwendung täglich für nur kurze Zeit verwendet wird und da fallen lange Ladezeiten mehr ins Gewicht.

2.4 Android

Zum Erstellen einer Android App wird der Quelltext in Java implementiert und zum Definieren der Layouts werden XML Dateien benutzt. Die entwickelte Android App, welche in Form eines APK (Android Package) installiert wird, ist mit Smartphones, welche mit der Android Version 4.4 oder höher ausgestattet sind, kompatibel.

Aufbau

Die wichtigsten Verzeichnisse in der Struktur eines Android Projektes sind src und res, wobei src den Java Quelltext der App enthält und res die Resourcen dazu, wie zum Beispiel Layouts der Views, Übersetzungen der Strings oder den Aufbau des Menüs. Daten des res Ordners werden vom Code aus über eine ID aufgerufen.

2.4.1 Fragments

Ein Fragment repräsentiert ein Verhalten oder einen Teil der Benutzeroberfläche in einer Activity. Mehrere Fragments können in einer einzigen Activity kombiniert werden, um eine Benutzeroberfläche mit mehreren Ansichten zu schaffen, dabei können auch einzelne Fragments in anderen Activities wieder verwertet werden. Fragments sind also eine Art Sub-Activities, welche ihren eigenen Lebenszyklus haben. Sie bekommen ihre eigenen Eingabe Events und man kann sie, während die Activity läuft, jederzeit hinzufügen oder wieder entfernen. Beim Hinzufügen zur Activity wird das Fragment in eine sogenannte ViewGroup im Layout der Activity eingebettet, in welchem das Fragment sein eigenes Layout definiert. Allerdings muss ein Fragment keine Benutzeroberfläche haben, sondern kann einfach Funktionen für die Activity bieten.

Fragments müssen immer in einer Activity eingebettet werden und werden dadurch direkt durch den Lebenszyklus dieser Activity beeinflusst. Allerdings können Fragments, während die Activity läuft, nach Belieben gesteuert werden, ohne die Activity zu beeinflussen. Wenn ein Fragment gestartet wird, kann es auf einen Backstack, welcher durch die Activity verwaltet wird, gelegt werden. Dies ist notwendig, um eine Navigation, beispielweise durch den Back-Button des Gerätes, zu ermöglichen.

Ein Vorteil der Benutzung von Fragments, ist das gleichzeitige Verwenden von Fragments bei Tablets. Während bei Smartphones das Display zu klein ist, um mehrere Ansichten auf einmal zu zeigen, haben Tablets meist viel ungenutzten Platz auf dem Bildschirm. Dieser kann durch mehrere Fragments gefüllt werden. Zum Beispiel kann eine Liste von Bildern auf der linken Seite des Bildschirm angezeigt werden, während auf der rechten Seite das aktuell ausgewählte Bild sichtbar ist. Beide Ansichten können separat, unumschränkt verwendet werden, da ein eigenes Fragment sie verwaltet, wobei wiederum beide Fragments durch eine Activity verwaltet werden. Das Beispiel ist in Abbildung 2.1 ersichtlich. [4]

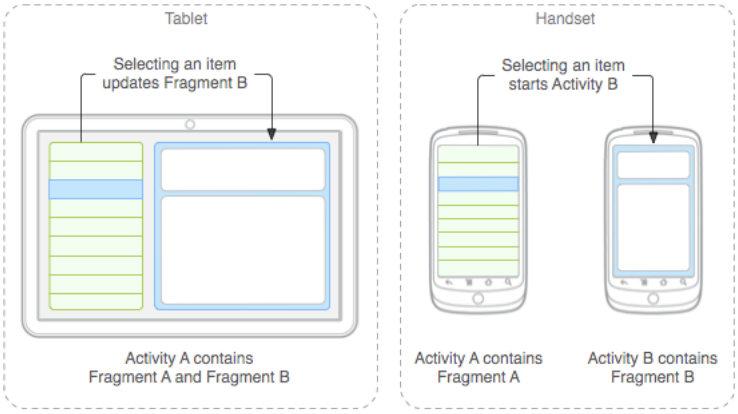


Abbildung 2.1: Ein Beispiel der Verwendung von zwei Fragments gleichzeitig auf einem Tablet und der separaten Ansicht auf einem Smartphone.

Android-Manifest

Jede Android App hat eine Datei namens *AndroidManifest.xml* in ihrem Wurzel Verzeichnis. Diese Datei enthält notwendige Informationen zur App, welche das Betriebssystem benötigt bevor die App ausgeführt werden kann. Eigenschaften der App wie zum Beispiel der Name, benötigte Zugriffsrechte und die Minimum SDK Version. Die Minimum SDK Version legt fest, welche Android Version das Gerät mindestens haben muss, um die App ausführen zu können.[5]

Gradle

Gradle ist ein Build-Management Tool, vergleichbar mit Apache Maven. Android Projekte sind normalerweise sehr umfangreich und würden zum Bauen viel Zeit beanspruchen. Daher unterstützt Gradle inkrementelle und parallel ablaufende Build-Prozesse, wobei ersteres nur veränderte Teile der Software baut und zweiteres ermöglicht es parallel auf mehreren Prozessoren bestimmte Tasks laufen zu lassen. Dadurch lässt sich eine wesentlich höhere Geschwindigkeit des Erstellprozesses erreichen.[6]

Intents

Sind asynchrone Mechanismen im Android Betriebssystem für den Austausch von Nachrichten. Diese können dazu benutzt werden, um Activities der App, des Betriebssystems aufzurufen oder einen Service zu starten. Sie ermöglichen also den flexiblen Wechsel zwischen verschiedenen Activities und Anwendungen.[7]

Snackbar

Eine Snackbar ist eine schmale Benachrichtigung in Form einer Leiste am unteren Bildschirmrand, welche dem Benutzer eine Information anzeigt und eine Rückmeldung einfordern kann. Die Snackbar besitzt nur eine Zeile Text und keinen, bis maximal einen Button. Zu einem Zeitpunkt kann immer nur eine Snackbar angezeigt werden, welche dann nach einiger Zeit oder nachdem sie auf die Seite gezogen wird, wieder verschwindet. [8]

2.5 Facemorphing

Facemorphing ist ein spezifischer Anwendungsfall von Morphing (siehe Kapitel 2.5.1), wobei die markanten Punkte, die ineinander übergehen sollen, die Gesichtszüge wie Augen, Nase, Mund und die groben Umrisse des Gesichts sind.

2.5.1 Morphing

Morphing ist ein Vorgang bei dem der Computer zwei Bilder nimmt und aus diesen Zwischenbilder generiert. Anstatt die beiden Bilder einfach zu überlappen, werden sie zusätzlich noch verzerrt, so dass markante Punkte des Bildes (bspw. Ränder von Objekten wie Pflanzen, Tiere, Gegenstände ...) fließend ineinander übergehen. Wenn man diese Zwischenbilder in einem Video abspielt, erwecken sie die Illusion einer stetigen Entwicklung der im Bild gezeigten Gegenstände ineinander. [9]

2.5.2 ARGB-Farbmodell

Das RGB-Farbmodell ist ein additives Modell zur Darstellung von Farben, das häufig in Computern und anderen elektronischen Geräten wie Fernseher und Kameras verwendet wird. Der Name stammt von den Anfangsbuchstaben der vier Komponenten, nämlich dem Alpha-Kanal, Rot, Grün und Blau. Diese vier Komponenten werden addiert, um einen gewissen Farbton darzustellen. Das Modell basiert auf der menschlichen Wahrnehmung von Farben, da das menschliche Auge, aufgrund des Aufbaus der Netzhaut, Farben als eine Kombination der Farben Rot, Grün und Blau wahrnimmt. Der Alpha-Kanal wird verwendet, um zusätzlich die Durchlässigkeit der Farbe zu definieren.[10]
Der Alpha-Kanal gibt die Trübung, das Gegenteil von Transparenz, der Farbe an, wobei ein niedriger Wert eine eher transparente oder durchlässige Farbe und ein hoher Wert eine satte und kräftige Farbe zeigt. Die Komponenten Rot, Grün und Blau geben an, wie stark der jeweilige Farbton in der resultierenden Farbe enthalten ist. Wobei ein niedriger Wert den Farbton schwächer und ein hoher Wert den Farbton stärker einwirken lässt.
Am Computer wird ein einzelner Bildpunkt, wie Abbildung 2.2 zeigt, als ein 32-Bit Datensatz gespeichert, was einem Spektrum von über vier Milliarden verschiedenen Farben bietet. Dabei werden 8 Bit und damit 256 verschiedene Stufen jeder einzelnen Komponente verfügbar. Im Computerspeicher kann somit ein Bildpunkt als unsigned integer gespeichert werden, bei dem die obersten acht Bit den Alpha-Kanal darstellen und die nachfolgenden die Rot, Grün und Blau Komponenten.

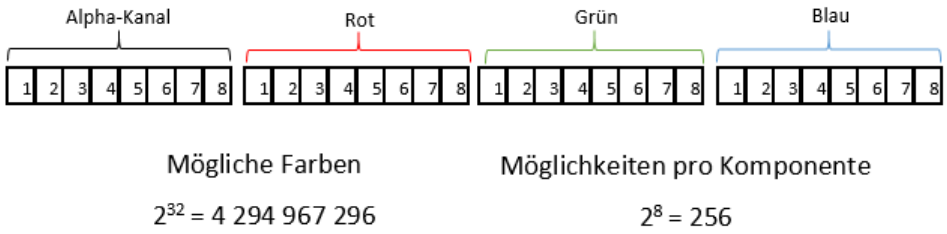


Abbildung 2.2: Bit-Darstellung eines Bildpunkte im ARGB-Modell

In Abbildung 2.3 sieht man zu Beginn, in Beispiel 1, die Farbe Grün, da nur die Komponente Grün verwendet wird und der Alpha-Kanal auf Maximalwert eingestellt ist. Mit einem stetig

sinkendem Alpha-Kanal, in den Beispielen 2 und 3, erscheint sie immer transparenter. Zum Schluss, in Beispiel 4, verblasst sie völlig, da der Alpha-Kanal auf Minimalwert eingestellt und daher eine 100%-ige Transparenz gegeben ist.

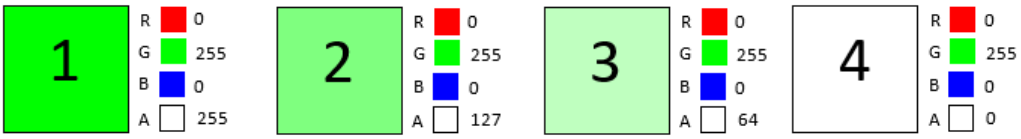


Abbildung 2.3: Auswirkung der Verringerung des Alpha-Kanals

In Abbildung 2.4 wird, in Beispiel 1, eine Farbe die den grünen Farbton völlig vernachlässigt, dafür aber Rot und Blau eingestellt hat und daher violett erscheint, gezeigt. Zusätzlich ist der Alpha-Kanal auf Maximum eingestellt, wodurch die Farbe mit 0%-iger Transparenz erscheint. In Beispiel 2 sieht man die Farbe Orange, zusammengesetzt aus der Komponente Rot auf Maximalwert und der Komponente Grün auf halbem Wert. Letztlich, in Beispiel 3, setzt sich Türkis aus der Komponente Grün und Blau auf Maximalwert.

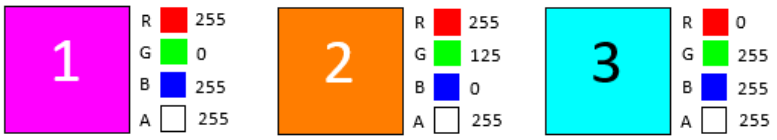


Abbildung 2.4: Auswirkung von R,G und B

In Abbildung 2.5 sieht man, dass wenn Rot, Grün und Blau alle den Minimalwert besitzen Schwarz, wenn sie jeweils zur Hälfte angezeigt werden Grau, und wenn sie alle den Maximalwert besitzen Weiß entsteht. Ohne Berücksichtigung des Alpha-Kanals.

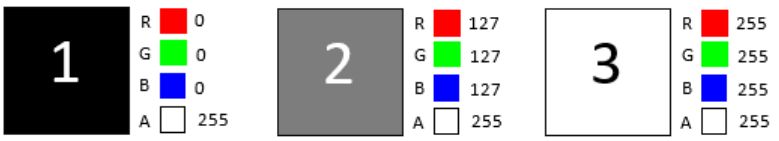


Abbildung 2.5: Schwarz-Weiß-Darstellung in ARGB

In Abbildung 2.6 wird gezeigt, wie sich die schwarze Farbe, wie in Abbildung 2.5 gezeigt, ändert wenn man den Alpha-Kanal erhöht, bzw. verringert. Ein Alpha-Kanal auf halbem Wert lässt schwarz grau erscheinen und ein Minimalwert im Alpha-Kanal lässt schwarz, weiß erscheinen.

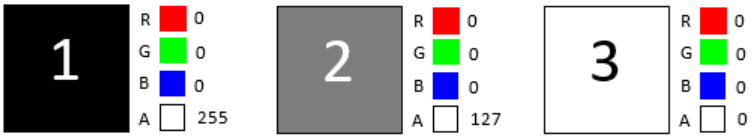


Abbildung 2.6: Auswirkung des Alpha-Kanals auf Schwarz-Weiß

2.5.3 Image Blending

Den Übergang zweier Bilder erreicht man durch das Überlappen dieser Bilder, welches durch einen Gewichtungsparameter reguliert wird. Wählt man beispielsweise den Gewichtungsparameter – folgend als ‚t' bezeichnet – im Intervall [0;1] so besagt dies, dass im Fall t=0 das Quellbild zu 100 % angezeigt wird und das Zielbild zu 0 %. Im Fall t=1 wird das Quellbild zu 0 % und das Zielbild zu 100 % angezeigt. Dieser Vorgang wird Image Blending (dt. Bildmischen) genannt.

Erzeugt man nun Zwischenbilder, wie in Abbildung 2.7 dargestellt, so beginnt das erste Bild mit t=0 (sprich Quellbild), die weiteren Bilder werden mit einem ständig inkrementierten Gewichtungsparameter erstellt, zum Beispiel wird bei zehn Zwischenbildern t, jedes mal um ein Zehntel erhöht – bis schließlich das letzte Bild mit t=1 (sprich Zielbild) abschließt.

Verfeinern kann man dieses Verfahren durch die Erhöhung der Zwischenbilder, wodurch der Übergang flüssiger erscheint. Man kann auch die Geschwindigkeit des Übergangs regulieren, in dem man t nicht linear, sondern zum Beispiel exponentiell wachsen lässt, wodurch der Übergang zu Beginn langsamer von statten geht als zum Schluss. Erreicht wird das Image Blending durch eine lineare Interpolation, der sich an der selben Stelle befindenden Bildpunkte aus beiden Bildern, mithilfe des Gewichtungsparameter t.

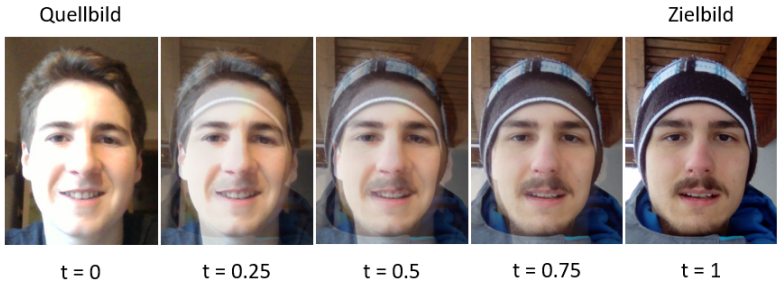


Abbildung 2.7: Image Blending

2.5.4 Lineare Interpolation

Die Lineare Interpolation ist ein Verfahren zur Erzeugung von Bildpunkten auf Basis anderer Bildpunkte. In der Bildbearbeitung wird es zum Beispiel verwendet, um zwischen einzelnen Bildpunkten Neue zu erstellen, um so Kanten zu glätten bzw. die Schärfe oder den Kontrast zu erhöhen/verringern.
In Abbildung 2.8 sieht man ein Beispiel einer linearen Interpolation. In diesem Beispiel wird der erste Bildpunkt mit dem ARGB-Code 255-250-0-0 (siehe Kapitel 2.5.2), unter Berücksichtigung

von t=0.5 mit dem zweiten Bildpunkt, mit ARGB-Code 255-0-0-200 linear interpoliert, indem die Werte der einzelnen Komponenten der beiden Bildpunkte nimmt und einen Zwischenwert errechnet.

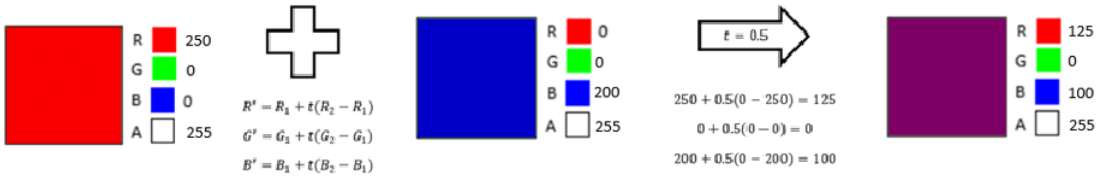


Abbildung 2.8: Lineare Interpolation

Die Abbildung 2.1 veranschaulicht die Methode, zur Errechnung dieses Zwischenwerts. Es wird der Wert der Komponente des ersten Bildpunkts F_1 genommen und mit der Differenz von (F_1) und dem Wert der selben Komponente des zweiten Bildpunkts F_2 addiert. Zuvor wird aber die Differenz mit dem Gewichtungsparameter t multipliziert, um den Einfluss des zweiten Bildpunkts zu regulieren.

$$F' = F_1 + t \cdot (F_2 - F_1) \tag{2.1}$$

Die Abbildung 2.9 [11] zeigt eine schwarze Linie auf weißem Hintergrund. Im linken Bild wurde keine lineare Interpolation angewendet. Im Rechten dagegen, wurden die Kanten der schwarzen Linie geglättet, in dem die Bildpunkte, die sich direkt am Übergang zwischen Linie und Hintergrund befinden, linear interpoliert wurden. Es wurde also ein Mittelwert erzeugt und so konnte ein fließender Übergang dargestellt werden.

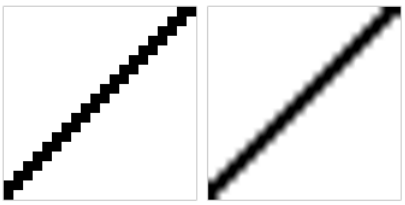


Abbildung 2.9: Kantenglättung durch lineare Interpolation

2.5.5 Image Warping

Neben der Überlappung der Bilder wird noch ein zweiter Vorgang benötigt. Die Zwischenbilder müssen nicht nur eine Überlappung nach dem Gewichtungsparameter des Quell- und des Zielbildes darstellen (Image Blending), sondern zusätzlich nach genau definierten Regeln (siehe Kapitel 2.6.1) verzerrt werden, um die einzelnen Gesichtsmerkmale des Quellbildes sequentiell an die des Zielbildes anzupassen. Dieser Vorgang nennt sich Image Warping (dt. Bildverzerrung) und kann auf zwei Arten realisiert werden: Durch Forward Mapping und Reverse Mapping. Um möglicherweise, durch Verzerrung hervorgerufene, gegebene Unschärfen im Bild zu bereinigen, wird eine lineare Interpolation auf dem verzerrten Bild durchgeführt.

2.5.6 Forward Mapping

Beim Forward Mapping wird das Quellbild Pixel für Pixel ausgelesen, wobei für jeden Pixel die richtige Position im Zielbild errechnet und in ein zuvor leer initialisiertes Zwischenbild eingefügt wird. Das Problem hierbei ist, dass nach dieser Methode manche Pixel im Zwischenbild „leer“ gelassen werden, da nicht garantiert wird, dass jeder Quellpixel einen zugehörigen Zielpixel zugeordnet bekommt. [12]
Die Abbildung 2.10 verdeutlicht diese Methode noch einmal. Beim ersten Schritt wird der Zielpixel aus dem Quellpixel errechnet und auf dem Zielbild eingefügt. Im zweiten Schritt, wird ein anderer Quellpixel genommen und der zugehörige Zielpixel daraus, an einer anderen Position als der Erste, berechnet. Im dritten Schritt jedoch bekommt der dritte Zielpixel die selbe Position wie der bereits existierende zweite Zielpixel. Das bedeutet, bei einer gleichen Anzahl an Pixel in Quell- und Zielbild, muss im Zielbild ein Pixel zwangsläufig leer bleiben, da aus zwei verschiedenen Quellpixeln nur ein Zielpixel errechnet wurde.

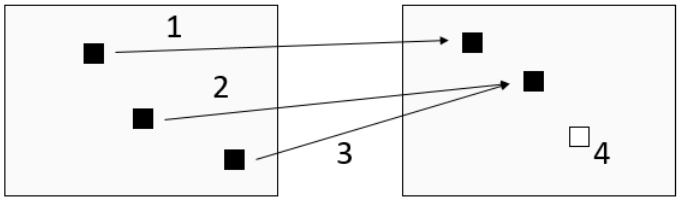


Abbildung 2.10: Forward Mapping

2.5.7 Reverse Mapping

Der Unterschied zwischen Forward Mapping und Reverse Mapping lässt schnell den Grund erahnen, welche Methode sinnvoller zu implementieren war. Die „leeren“ Pixel könnten im Nachhinein durch Interpolation der umliegenden Pixel ergänzt werden, was aber zu aufwändig wäre und es eine einfachere und vor allem schnellere Alternative gibt. Diese Alternative ist das Reverse Mapping, bei welchem man anstatt dem Quellbild das Zielbild Pixel für Pixel durchläuft und jedem Pixel einen Pixel aus dem Quellbild entnimmt und sich daraus den neuen Pixel für das Zwischenbild errechnet. Dadurch kann das Problem, dass nicht jeder Pixel einen korrespondierenden Pixel besitzt, gelöst werden.[12]
In der Abbildung 2.11 wird dieser Vorgang veranschaulicht. Es wird jede Position im Zielbild durchlaufen, was verhindert, dass ein Zielpixel leer gelassen oder ignoriert wird. Es kann vorkommen, dass mehrere Zielpixel den selben Quellpixel als Basis nehmen (siehe Schritt 2 und 3). Das ist kein Problem, da bei der Berechnung des neuen Pixels die Position des Zielpixels wichtig ist (siehe Kapitel 2.6.1) und diese bei verschiedenen Zielpixeln natürlich unterschiedlich ist.

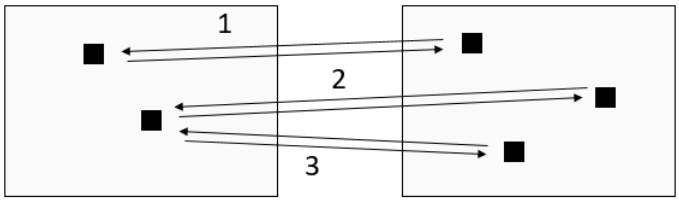


Abbildung 2.11: Reverse Mapping

2.6 Beier-Neely Algorithmus

Der Beier-Neely-Algorithmus ist ein auf bestimmten Merkmalen basierender Algorithmus, um zwei Bilder - hier Quell- und Zielbild genannt - ineinander verschmelzen zu lassen und so einen visuellen, flüssigen Übergang des Quellbildes in das Zielbild zu gewährleisten. Dies wird durch zuvor definierte Gesichtsmerkmale, in Form von Linien an den markanten Stellen auf beiden Bildern und der Errechnung von – in weiterer Folge als Zwischenbilder bezeichnet – Teilbildern garantiert. Der Algorithmus nutzt dazu zwei wesentliche Methoden, das so genannte Image Blending (siehe Kapitel 2.7) und Image Warping (siehe Kapitel 2.5.5).

2.6.1 Image Warping beim Beier-Neely-Algorithmus

Das Image Warping des Beier-Neely-Algorithmus basiert auf Berechnungsregeln die von den Erfindern des Algorithmus definiert wurden und zwei Fälle unterscheiden. Die Berechnung mit einem Linienpaar (siehe Kapitel 2.6.2) und die Berechnung bei mehreren Linienpaaren (siehe Kapitel 2.6.3).

2.6.2 Linienpaare

Ein Linienpaar bezeichnet ein Paar zugeordneter, gerichteter Linien – also zwei Vektoren: Ein Vektor aus dem Quellbild (definiert durch die Punkte $\boldsymbol{P}$ und $\boldsymbol{Q}$ und folgend als Quellvektor bezeichnet) und der zugehörigen Linie im Zielbild (definiert durch die Punkte $\boldsymbol{P'}$ und $\boldsymbol{Q'}$ und folgend als Zielvektor bezeichnet). Daraufhin ermittelt man nun die ($\boldsymbol{u}$,$\boldsymbol{v}$)-Koordinaten für jeden Quellpixel $\boldsymbol{X}$. Der Parameter $\boldsymbol{u}$ (siehe Abbildung 2.2) gibt den prozentuellen Abstand entlang des Quellvektors im Intervall [0;1] an. Der Parameter $\boldsymbol{v}$ (siehe Abbildung 2.3) gibt die Distanz zum Quellvektor an.[12]

$$\boldsymbol{u} = \frac{(\boldsymbol{X} - \boldsymbol{P}) \cdot (\boldsymbol{Q} - \boldsymbol{P})}{||\boldsymbol{Q'} - \boldsymbol{P'}||^{\boldsymbol{2}}} \tag{2.2}$$

$$\boldsymbol{v} = \frac{(\boldsymbol{X} - \boldsymbol{P}) \cdot \boldsymbol{norm}(\boldsymbol{Q} - \boldsymbol{P})}{||\boldsymbol{Q'} - \boldsymbol{P'}||} \tag{2.3}$$

Die Funktion norm() berechnet den Normalvektor des übergebenen Vektors, also dem Vektor der dieselbe Länge wie der übergebene Vektor und einen 90 Grad Winkel auf diesen, aufweist. Anschließend wird entlang des Zielvektors, der korrespondierende Zielpixel X‘ mit gleichen (u,v)-Koordinaten aus dem Quellbild ermittelt. Dieser wird darauffolgend als Zielpixel des Zielbildes verwendet. In Pseudocode sieht dieser Vorgang wie folgt aus:

```
1 for alle Pixel X aus Zielbild do
2 |   Berechne u und v von X aus Quellbild;
3 |   Suche Zielpixel X' mithilfe von u und v aus Quellbild;
4 |   Schreibe X' aus Quellbild in Zielbild;
5 end
```

Der Pseudocode-Algorithmus (siehe Abbildung 1) beschreibt den Vorgang des Image Warpings zweier Bilder mit einem paar Linien. Der etwas kompliziertere Vorgang mit mehreren Liniepaaren

wird im Kapitel 2.6.3 beschrieben. Bei einem Linienpaar werden, wie in Zeile 1 ersichtlich, alle Pixel $\boldsymbol{X}$ aus dem Zielbild, per Reverse Mapping (siehe Kapitel 2.5.7), durchlaufen. In Zeile 2 werden zu jedem Zielpixel die ($\boldsymbol{u}$,$\boldsymbol{v}$)-Koordinaten des Quellpixels, der sich an der selben Position befindet, in Relation zum Quellvektor berechnet. In Zeile 3 wird aus diesen (u,v)-Koordinaten die Position des neuen Zielpixel $\boldsymbol{X'}$, in Relation zum Zielvektor, berechnet. in Zeile 4 wird schließlich der Pixel an der Stelle von $\boldsymbol{X'}$ aus dem Quellbild genommen und ins Zielbild kopiert.

2.6.3 Mehrere Linienpaare

Im Gegensatz zum Image Warping bei einem einzelnen Linienpaar, wie in Kapitel 2.6.2, muss, bei mehreren Paaren, für jedes Linienpaar eine neue Position des Zielpixels berechnet werden. Von jedem der erstellten Zielpixel wird sein Abstandsvektor zum Quellpixel berechnet. Diese Abstandsvektoren werden unter Berücksichtigung einer Gewichtung (siehe Kapitel 2.6.3) addiert und zum Quellpixel dazugerechnet, um die Position des Zielpixels zu ermitteln.

Gewichtung

Die Gewichtung gibt an, wie viel Einfluss die Abstände, zum Quellpixel, und die Längen der Vektoren der einzelnen Linienpaare auf die Berechnung des Zielpixels, nehmen.

$$\boldsymbol{Gewichtung} = \left(\frac{\boldsymbol{L^p}}{\boldsymbol{(a+D)}}\right)^{\boldsymbol{b}} \tag{2.4}$$

Die Formel, in Abbildung 2.4, zeigt, wie sich diese Gewichtung berechnet. Der Parameter $\boldsymbol{L}$ gibt die Länge des Zielvektors und der Parameter $\boldsymbol{D}$ die Distanz vom Quellpixel zum Zielvektor an. Der Parameter p wird im Bereich [0;1] gewählt und gibt an, wie viel Einfluss die Länge der Linie auf die Gewichtung nimmt. Dabei sagt der Wert p=0 aus, dass die Länge gar keine Auswirkung und der Wert p=1, dass die Länge volle Auswirkung hat. Bei höheren Werten des Parameters a wird die Kantenglättung (siehe Kapitel 2.5.4) stärker. Der Parameter b gibt an, wie stark sich unterschiedliche Gewichtungen auswirken. Bei einem Wert von b=0 nimmt der Pixel von jeder Linie gleich viel Einfluss.

```
1  for alle Pixel X aus Zielbild do
2  |  Abstandsvektorsumme = (0,0);
3  |  Gewichtungssumme = 0;
4  |  for alle Linien PiQi do
5  |  |  Berechne u und v mithilfe von PiQi;
6  |  |  Berechne Xi aufgrund von u, v und Pi'Qi';
7  |  |  Berechne Abstandsvektor = Xi'-Xi;
8  |  |  Berechne Gewichtung aufgrund von X und PiQi;
9  |  |  Addiere Gewichtung * Abstandsvektor zu Abstandvektorsumme;
10 |  |  Addiere Gewichtung zu Gewichtungssumme;
11 |  end
12 |  X' = X + Abstandsvektorsumme / Gewichtungssumme;
13 |  Schreibe X' aus Quellbild ins Zielbild;
14 end
```

Der Pseudocode-Algorithmus, in Abbildung 2, zeigt den Vorgang der Berechnung des Zielpixels aus einem Quellpixel und mehreren Linienpaaren. Der Unterschied zu einem Linienpaar ist, dass in Zeile 2 und 3, für jeden Quellpixel, ein neuer Summenvektor und eine neue Gewichtungssumme definiert werden. In Zeile 4 wird für jedes Linienpaar ein Schleifendurchlauf gemacht, um den Abstandvektor zum Quellpixel, in Zeile 7, und die Gewichtung (siehe Kapitel 2.6.3), auf Basis des Linienpaars in Zeile 8, zu berechnen. Dieser Abstand und die Gewichtung werden von allen Linienpaaren addiert und nach der Schleife in Zeile 12 zur Berechnung der Position $\boldsymbol{X'}$ des Zielpixels zur Hilfe gezogen.

3 Benutzung

3.1 Programmablauf

Nach der Installation der App, kann der Benutzer die App wie gewohnt per Klick starten, woraufhin sie sich öffnet und die Startseite angezeigt wird.

3.1.1 Startseite

Auf der Startseite, welche auf Abbildung 3.1 zu sehen ist, wird die Kamera-Vorschau angezeigt, sowie ein Button zum Speichern eines Fotos. Außerdem gibt es eine Menüleiste, welche auf allen Seiten der App angezeigt wird, Menüobjekte werden jedoch nur auf der Startseite angezeigt. Die verfügbaren Menüobjekte sind: Ein Button zum Wechseln der Front- und Back-Kamera, ein Button welcher zur Morphing Seite weiterleitet und ein Button, welcher zu den Einstellungen führt.

Abbildung 3.1: Startseite

3.1.2 Behalten des Bildes

Nach der Aufnahme eines Fotos wird ein neues Fragment (siehe Abb. 3.2) geöffnet, auf dem sich der Benutzer das Foto noch einmal ansehen kann. Er wird gefragt, ob das Foto verworfen oder gespeichert werden soll. Beim Verwerfen wird man wieder auf die Startseite weitergeleitet und beim Behalten auf die Seite zum Einzeichnen der Linien. Diese Seite ist in Abbildung 3.4 zu sehen.

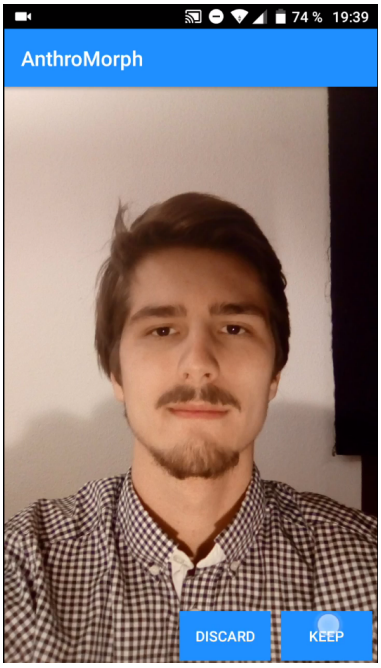


Abbildung 3.2: Benutzerentscheidung: Bild behalten?

3.1.3 Einzeichnen der Linien

Sobald der Benutzer auf diese Seite kommt, erscheint noch eine Snackbar, wie in Abbildung 3.3 ersichtlich, welche das Teilen des behaltenen Fotos in Sozialen Netzwerken und diversen Chats mit wenigen Klicks möglich macht.

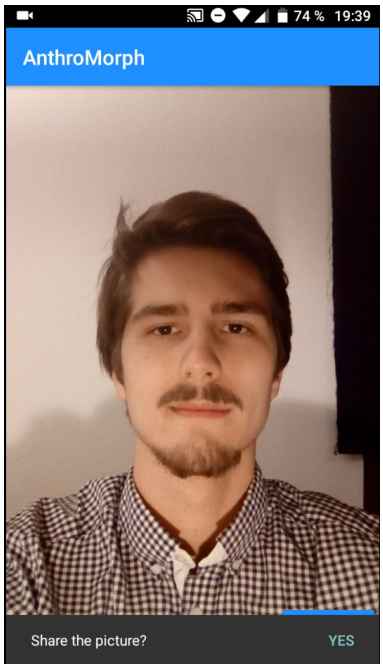


Abbildung 3.3: Snackbar zum Teilen

Nun muss der Benutzer die Linien, welche für den Beier-Neely Algorithmus (siehe Kapitel 2.6) benötigt werden, auf dem angezeigten Foto einzeichnen. Ein Beispiel für die eingezeichneten Linien ist auf Abbildung 3.4 zu sehen. Beim Bestätigen der Linien werden diese gespeichert und es wird wieder die Startseite angezeigt.

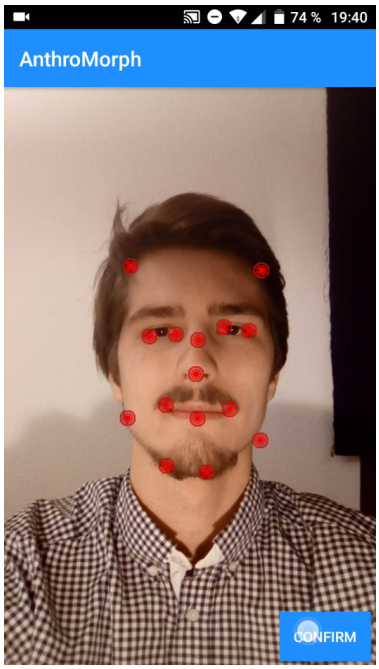


Abbildung 3.4: Einzeichnen der Linien durch den Benutzer

3.1.4 Einstellungen

In den Einstellungen kann eine Erinnerung gesetzt werden, welche täglich zu einer bestimmten Uhrzeit, den Benutzer mit einer Benachrichtigung daran erinnert, ein Bild aufzunehmen. Die Auswahl der Zeit ist auf Abbildung 3.5 ersichtlich. Bei einem Klick auf die Benachrichtigung (Abb. 3.6) wird die App, falls sie nicht schon geöffnet ist, gestartet.

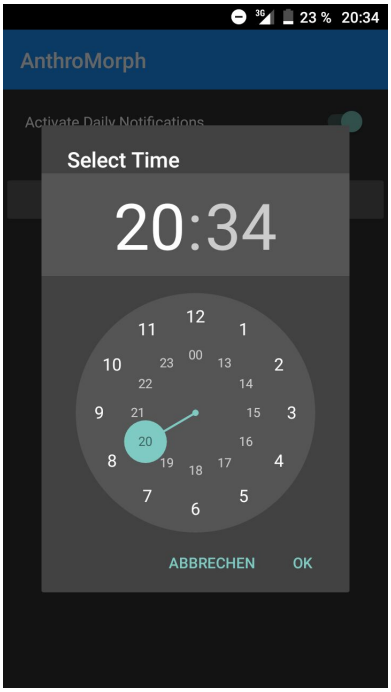


Abbildung 3.5: Auswählen der Zeit

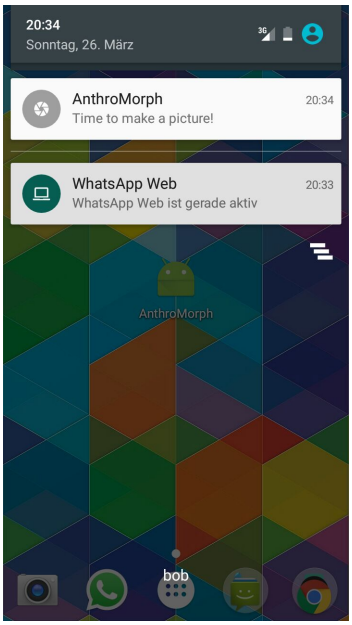


Abbildung 3.6: Benachrichtigung

3.1.5 Morphing Seite

Diese Seite wird beim Drücken des Morph-Buttons in der Menüleiste aufgerufen. Darin wird dem Benutzer die Möglichkeit geboten, Bilder zu morphen oder eine Videodatei aus den gemorphten Bildern zu erzeugen.

4 Programmstruktur

4.1 Systemmodell

In diesem Kapitel werden die Software-Architektur und das Oberflächen Design der Android App beschrieben.

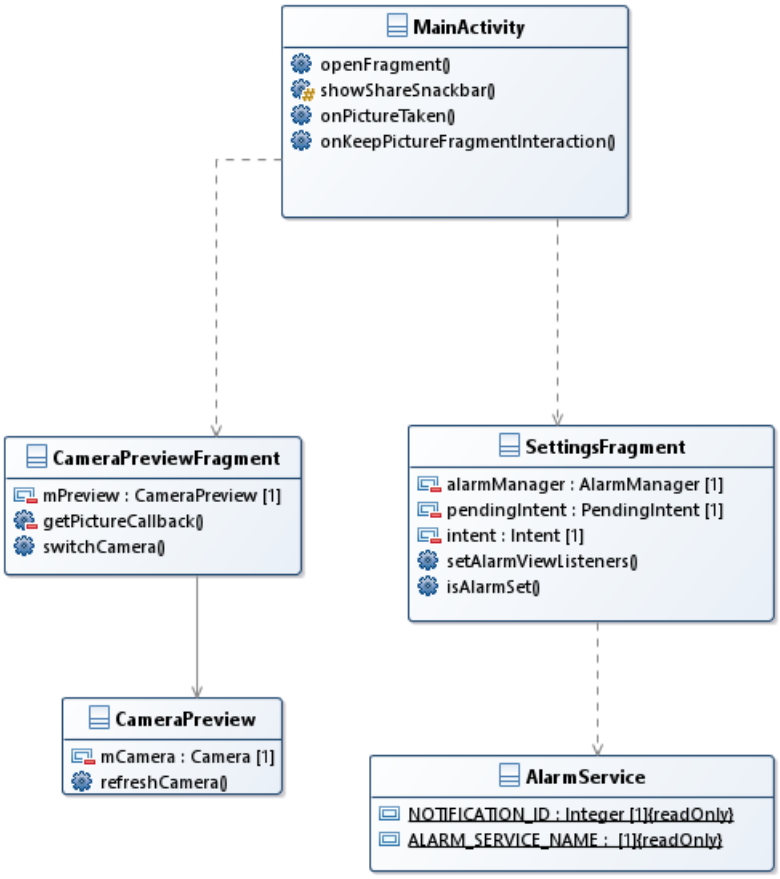


Abbildung 4.1: Klassendiagramm der App - Teil 1[13]

4.1.1 Main-Activity

Beim Start der App wird zu allererst die Main-Activity, welche in Abbildung 4.1 abgebildet ist, gestartet. Die dazugehörende XML-Datei, welche sich im Ordner *res/layout* befindet, legt das Layout der Ansicht fest. In der Main-Activity wird die Menüleiste instanziiert und gleich zu Beginn das *CameraPreviewFragment* geöffnet. Die Main-Activity hat die Methode *openFragment()*, welche Fragments mithilfe des SupportFragmentManagers öffnen kann. Der SupportFragmentManager der App wird dann von den Fragments aus aufgerufen, um sich selbst wieder zu schließen. [4]

4.1.2 Camera-Preview-Fragment

Das *CameraPreviewFragment* bietet die Kamerafunktion, es wird gleich beim Starten der App von der MainActivity geöffnet. Die Vorschau der Kamera wird hier in einem LinearLayout angezeigt. Ein FloatingActionButton dient als Auslöser der Kamera. Die Menüleiste wird initialisiert und es wird festgelegt, was beim Klicken einer Menütaste passiert. Die verschiedenen Auswahlmöglichkeiten in der Menüleiste sind: Das Wechseln zwischen Front- und Back-Kamera, das Öffnen des *Video-Morphing-Fragments* und das Öffnen der Einstellungen.

Ein Objekt der Klasse *CameraPreview* (Abb. 4.1) ist für das ordnungsgemäße anzeigen des Kamera Previews zuständig. Das umfasst unter anderem das Finden der optimalen Vorschaugröße, Einstellen der Kameraauflösung und Starten, Stoppen sowie Wiederaufnehmen des Previews. Die *CameraPreviewFragment* Klasse selbst wiederum kümmert sich um das Wechseln zwischen den verschiedenen, vorhandenen Kameras sowie um das Aufnehmen eines Fotos und dem Initialisieren des Previews und der Kamera selbst.

4.1.3 Settings-Fragment

Das *SettingsFragment* wird über die Menüleiste aufgerufen und dient dazu, eine tägliche Benachrichtigung einzustellen. Dazu wird in der Klasse ein Intent, mit dem *AlarmService* (Abb. 4.1) als IntentService, angelegt und ein AlarmManager wird initialisiert.[14] Der AlarmManager lässt den eingestellten Alarm losgehen und der Intent sorgt dafür, dass die App geöffnet wird, wenn die Benachrichtigung angeklickt wird.[7] Ein Switch öffnet bei Betätigung einen TimePicker bei dem die Uhrzeit, wann die Benachrichtigung täglich auftauchen soll, ausgewählt wird. Sobald die Uhrzeit bestätigt wird, bekommt der AlarmManager die Uhrzeit und den Intent übergeben.[15] Mithilfe dieser Parameter löst der AlarmManager die Benachrichtigung aus, auch wenn die App zwischendurch geschlossen wird. Der Name des Alarms ist dabei sehr wichtig und muss zum Abbrechen, Kontrollieren und Setzen des Alarms immer der Gleiche sein. Wenn der Name nicht gespeichert wird, kann der Alarm nicht mehr beeinflusst werden und das Gerät muss neu gestartet werden, um ihn zu beenden. In der *AlarmService* Klasse sind zusätzlich noch alle Informationen über die Benachrichtigung, wie zum Beispiel den Text Inhalt, das Layout und den Klingelton oder ob das Gerät dabei Vibrieren soll, festgelegt.

4.1.4 Show-Picture-Fragment

Das *ShowPictureFragment* wird geöffnet, wenn im *ImagePreviewFragment* ein Bild aufgenommen wird. Sie zeigt das eben aufgenommene Bild in einer ImageView an und bietet die Möglichkeit das Bild zu verwerfen oder es zu behalten. Wenn das Bild verworfen wird, ruft das Fragment den *SupportFragmentManager* auf und schließt sich selbst. Wird das Bild behalten, bleibt das *ShowPictureFragment* offen und dem RelativeLayout, in dem das Bild angezeigt wird, wird ein onTouchListener hinzugefügt.

Dieser Listener ermöglicht das Einzeichnen der Punkte, welche für den Morphing-Prozess benötigt werden. Außerdem gibt es noch zusätzliche Listener, welche den Punkten hinzugefügt werden, welche dafür sorgen, dass diese Punkte verschoben werden können. Die eingezeichneten Punkte können dann bestätigt werden und das Fragment wird wieder geschlossen und der Interaction-Listener des Fragments übergibt der Main-Activity das Bild, welches behalten werden soll. Die Main-Activity ruft nun die Util-Klasse *FileManager* auf, welche dem Bild einen einzigartigen Namen (bestehend aus dem aktuellen Datum, der aktuellen Uhrzeit und einer fortlaufenden Nummer, falls in einer Minute mehrere Bilder gemacht werden) gibt und dieses dann als JPEG-

Datei speichert. Zusätzlich dazu, werden die Gesichtsmerkmale gemeinsam mit einem Pfad zum zugehörigen Bild in einer Datei als Serializable gespeichert. Danach wird noch eine Snackbar angezeigt, welche es ermöglicht das gespeicherte Bild auf Sozialen Netzwerken oder in Chats zu teilen und der Android-Mediascanner wird benachrichtigt, dass das Bild gespeichert wurde. Dies aktualisiert den Speicher des Gerätes, um das neue Bild für den Benutzer zugänglich zu machen.[8]

4.1.5 Video-Morphing-Fragment

Im *VideoMorphingFragment* werden Hintergrundprozesse (Abb. 4.1.3) aufgerufen, welche dafür zuständig sind die Bilder der App zu morphen beziehungsweise die gemorphten Bilder in einer Videodatei zu vereinen.[16] Der *MorpherTask* bekommt die zwei Bilder, zwischen denen der Morphingprozess stattfinden soll und speichert diese als Bilder. Der *GenerateVideoTask* hängt dann alle Bilder aneinander und speichert diese, mithilfe von JCodec[17], als eine MP4-Datei ab.

4.1.6 Facemorphing

Im *PictureMorpher* werden jeweils zwei Bilder eingelesen und miteinander über Image Warping (siehe Kapitel 2.5.5) und Image Blending (siehe Kapitel 2.5.3) in einer gewissen, durch *FRAMES* definierten, Anzahl an Zwischenbildern gemorpht und gespeichert. Der *PictureMorpher* wird vom *MorpherTask* aufgerufen (siehe Kapitel 4.1.5) und startet dann für die Erstellung eines jeden Zwischenbildes einen *GenerateNIntermediateThread*, welche alle in einem Threadpool verwaltet werden. Die Parameter *PARAMETER_A*, *PARAMETER_B* und *PARAMETER_P* werden zur Berechnung der Zwischenbilder benötigt und geben Auskunft darüber, wie das Bild verzerrt wird. Die *sourceFeatures* und *targetFeatures* sind die Listen von Quell- und Zielvektoren, die über den onTouchListener des *Show-Picture-Fragment*s eingezeichnet wurden. Der *GenerateN-IntermediateThread* ruft die morph2Images-Methode auf, in der alle wichtigen Methoden (siehe Kapitel5.3) zur Erstellung der Zwischenbilder aufgerufen werden und für die zwei übergebenen Bilder die Zwischenbilder abgespeichert werden.

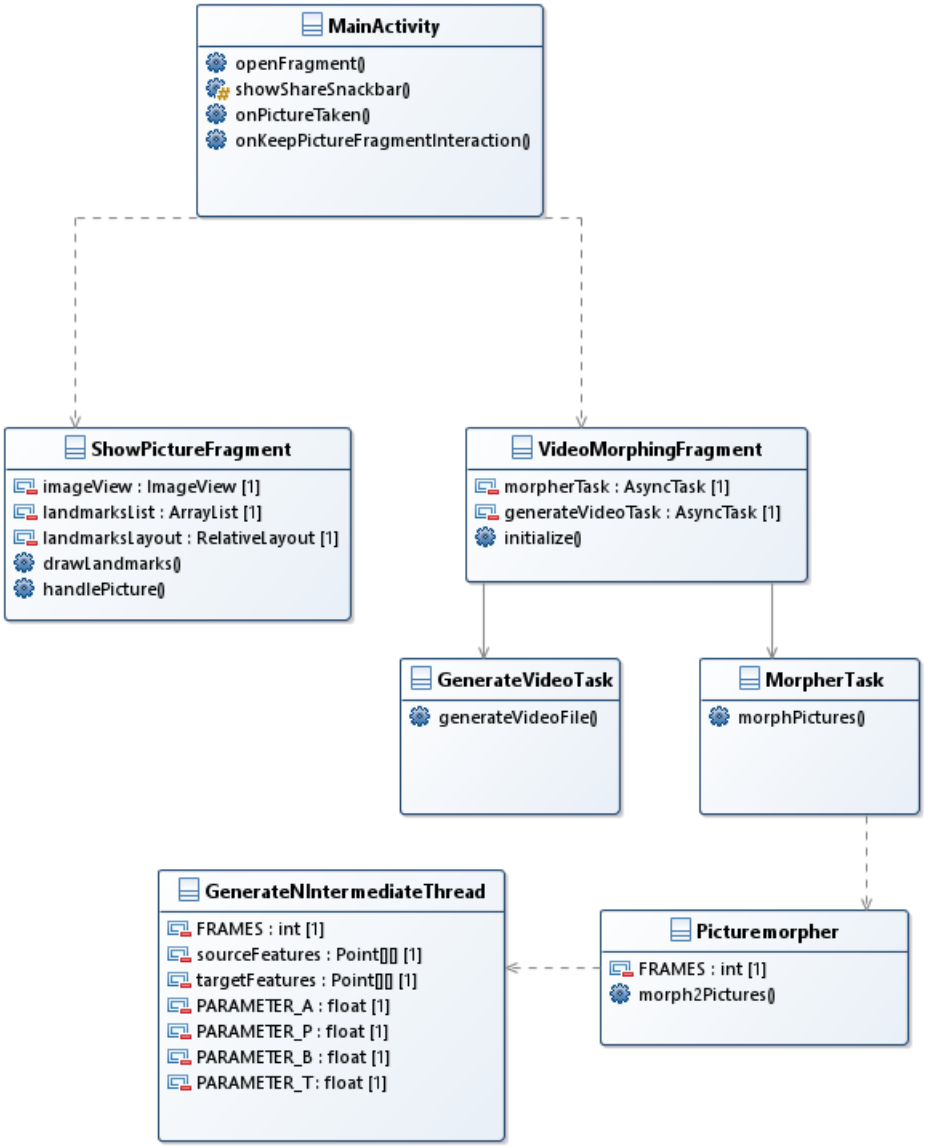


Abbildung 4.2: Klassendiagramm der App - Teil 2

5 Implementierung

5.1 Benachrichtigung

Die Benachrichtigung der App ist ein so genannter wiederholender Alarm und basiert auf der AlarmManager Klasse. [18][15] Diese bietet die Möglichkeit zeitbasierte Operationen außerhalb der Applikation auszuführen. Der Alarm kann bei schlechtem Design die verfügbaren Ressourcen zu sehr belasten. Beispielsweise, wenn der Alarm die Daten der App jeden Tag um elf Uhr mittags mit einem Server synchronisieren soll, kann die Kommunikation von mehreren Instanzen auf verschiedenen Geräten dazu führen, dass der Server überlastet wird. Dies resultiert in erhöhter Latenz und kann schlimmsten falls zum Ausfall des Servers führen.

Daher sollten folgende Dinge berücksichtigt werden:

- Zum Zeitpunkt des Alarms werden mögliche anfallende Daten, welche an den Server gesendet werden, bereitgestellt und erst nach einer zufälligen Zeit wirklich verschickt.
- Minimieren der Wiederholungen.
- Das Gerät nur aufwecken, wenn es wirklich notwendig ist.
- Die auslösende Zeit so unpräzise wie möglich setzen, dies ist durch die Benutzung von *setInexactRepeating()* anstatt von *setRepeating()* gegeben. Wobei zweiteres den Alarm sofort feuert. Ersteres, sammelt diverse Alarme und feuert diese zur selben Zeit. Das reduziert die Gesamtzahl der Weckungen des Gerätes und verlängert somit die Batterielaufzeit.
- Feuern des Alarms nach verstrichener Zeit und nicht zu einem bestimmten Zeitpunkt.

Ein wiederholender Alarm hat:

- einen Alarmtyp
- eine Zeit, welche ihn auslöst
- ein Alarm Intervall
- einen PendingIntent, welcher startet, wenn der Alarm ausgelöst wird[19]

Umsetzung

Im *SettingsFragment* wird ein Intent mit dem Context und einer Referenz auf die Klasse AlarmService initialisiert. Mithilfe von *getSystemService()* wird ein AlarmManager vom Context der Main-Activity geholt, dieser wird nur benötigt, um den wiederholenden Alarm zu setzen. Danach wird überprüft, ob der Dienst der Anwendung bereits läuft, durch die `NOTIFICATION_ID` des AlarmService ist es möglich den Alarm wieder zu finden (siehe Abb. 5.1). Ein PendingIntent, welcher mit der `NOTIFICATION_ID` und dem oben gesetzten Intent initialisiert wird, wird beim Setzen des Alarms (siehe Abb. 5.2) vom AlarmManager an das System übergeben. Der Intent sorgt dafür, dass beim Feuern des Alarms der AlarmService seine Notifikation anzeigt.

Der PendingIntent speichert nur den Intent, um ihn zu einem geeigneten Zeitpunkt weiter zu geben.

```
intent = new Intent(getContext(), AlarmService.class);
intent.setAction(AlarmService.ALARM_SERVICE_NAME);
alarmManager = (AlarmManager) getContext().getSystemService(MainActivity.ALARM_SERVICE);
pendingIntent = PendingIntent.getService(getContext(), AlarmService.NOTIFICATION_ID, intent,0);
```

Abbildung 5.1: Initialisierung der Objekte

```
Calendar alarmTime = Calendar.getInstance();
alarmTime.set(Calendar.HOUR_OF_DAY, selectedHour);
alarmTime.set(Calendar.MINUTE,selectedMinute-1);

alarmManager.setInexactRepeating(AlarmManager.RTC_WAKEUP,alarmTime.getTimeInMillis()
        , AlarmManager.INTERVAL_DAY, pendingIntent);
```

Abbildung 5.2: Setzen eines Alarms

5.2 Einzeichnen der Linien

5.2.1 Umsetzung

Im *ShowPictureFragment* wird das gemachte Bild nicht nur angezeigt um es zu behalten oder zu verwerfen. Sondern es ist hier auch eine wichtige Funktion für das Morphen implementiert, nämlich das Einzeichnen der Linien, welche im Beier-Neely Algorithmus (siehe Kapitel 2.6) benötigt werden. Ein RelativeLayout dient als Container für die Punkte, welche angezeigt werden und für das aufgenommene Bild, welches in einer ImageView im RelativeLayout angezeigt wird. Im Fragment wird dann ein onTouchListener auf das Layout gesetzt und dieser kümmert sich darum, was geschieht, wenn der Benutzer auf den Bildschirm drückt, um einen Punkt einzuzeichnen. Die Implementierung dieses Listeners ist in Abbildung 5.3 zu sehen.

Als Erstes wird die Berührung in einem View Objekt namens Landmark gespeichert, dann wird dieser View ein Hintergrund zugewiesen, welcher den Punkt für den Benutzer sichtbar macht. Dem Objekt Landmark wird nun wiederum ein onTouchListener hinzugefügt, welcher es erlaubt den Punkt, nach dem er gesetzt wurde, beliebig am Bildschirm zu verschieben, siehe Abbildung 5.4. Um nun die passenden Koordinaten der Landmark auf dem gespeicherten Bild zu ermitteln, müssen zuerst die Koordinaten im Verhältnis zur View-Größe ermittelt werden, da das gespeicherte Bild nicht die gleiche Auflösung wie die View besitzt. Nun kann man die relativen Koordinaten zu absoluten Koordinaten umrechnen, indem man die Relativen mit der Höhe und Breite des gespeicherten Bildes multipliziert.

Im linken unteren Teil des Bildschirms, wird in einem Button angezeigt, welches Gesichtsmerkmal gerade eingezeichnet werden soll. Nachdem der Benutzer ein Merkmal eingezeichnet hat, muss er auf den Button drücken und das nächste Merkmal wird angezeigt. Dies macht der Benutzer so lange bis alle Merkmale eingezeichnet sind. Diese werden gemeinsam mit dem Pfad des Bildes, zu welchem sie gehören, serialisiert und abgespeichert.

```
landmarksLayout.setOnTouchListener((v, event) → {
        final int x = (int) event.getX();
        final int y = (int) event.getY();

        View landmark = new View(getContext());

        landmark.setBackgroundResource(R.drawable.point);
        RelativeLayout.LayoutParams lp = new RelativeLayout.LayoutParams(30, 30);
        landmark.setLayoutParams(lp);
        landmark.setX(x-15);
        landmark.setY(y-15);
        landmark.setOnTouchListener(new View.OnTouchListener() {...});

        ImageView imageView=(ImageView) getView().findViewById(R.id.takenPicture);

        int viewWidth=imageView.getWidth();
        int viewHeigth=imageView.getHeight();

        float xRelative = (float) x/viewWidth;
        float yRelative = (float) y/viewHeigth;

        currentLandmark.add(new MyPoint((int) (xRelative*imageView.getDrawable().getMinimumWidth()),
                (int) (yRelative*imageView.getDrawable().getMinimumHeight())));

        landmarksLayout.addView(landmark);

        return false;
});
```

Abbildung 5.3: onTouchListener zum Einzeichnen der Punkte

```
landmark.setOnTouchListener(new View.OnTouchListener() {
                    float dX, dY;
                    @Override
                    public boolean onTouch(View view, MotionEvent event) {
                        switch (event.getAction()) {
                            case MotionEvent.ACTION_DOWN:
                                dX = view.getX() - event.getRawX();
                                dY = view.getY() - event.getRawY();
                                break;
                            case MotionEvent.ACTION_MOVE:
                                view.animate()
                                        .x(event.getRawX() + dX)
                                        .y(event.getRawY() + dY)
                                        .setDuration(0)
                                        .start();
                                break;
                            default:
                                return false;
                        }
                        return true;
                    }
                });
```

Abbildung 5.4: onTouchListener zum Verschieben der Punkte

5.2.2 Andere Möglichkeiten

Für das Finden von Gesichtsmerkmalen auf einem Bild, gibt es einen Google Service namens *Mobile Vision API"*. Dieser bietet eine einfache Möglichkeit für das Finden von bestimmten Gesichtsmerkmalen, jedoch gibt es nur eine beschränkte Anzahl an Punkten, wie in Abbildung 5.5 zu sehen, welche durch diesen Service ermittelt werden. Zum Beispiel liefert der Service nur einen Punkt an der Stelle, an der das linke Auge ist. Der Beier-Neely Algorithmus benötigt aber pro Gesichtsmerkmal mehrere Punkte, also eine Linie, um den Morphingprozess darzustellen. Da das automatische Finden der Gesichtsmerkmale in diesem Service nicht erweitert werden kann, konnte er für die Anwendung nicht benutzt werden.[20]

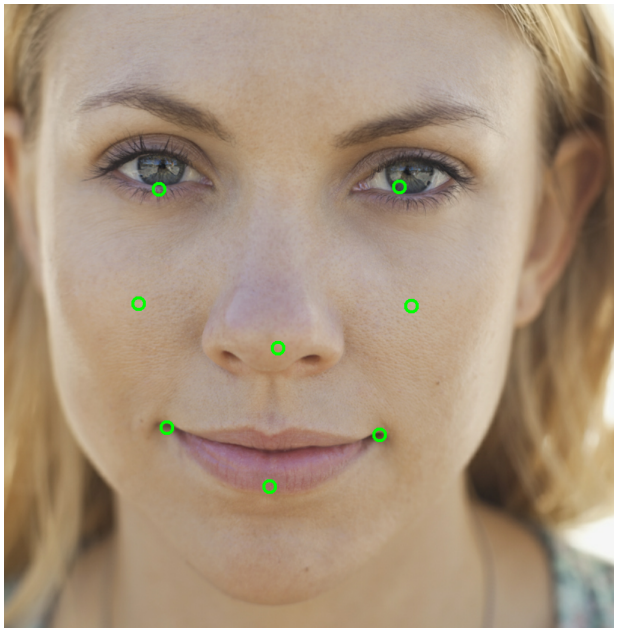


Abbildung 5.5: Beispiel der Gesichtsmerkmale, welche durch den Google Service gefunden werden.

Eine weitere Möglichkeit für das automatische Finden der Gesichtsmerkmale, wäre die Bibliothek *Dlib* oder das *CLM-framework*, welche aber beide nicht für die Programmiersprache Java verfügbar sind, sondern für C++ bzw. Dlib auch für Python.[21]

5.3 Beier-Neely Algorithmus

Umsetzung

Der Morphing-Algorithmus wird in den Klassen PictureMorpher und GenerateNIntermediateThread realisiert. Diese implementieren alle Methoden die notwendig sind, um zwei Bilder mithilfe von Image Warping (siehe Kapitel 2.5.5) zu verschmelzen und mithilfe von Image Blending (siehe Kapitel 2.5.3) zu überblenden.

5.3.1 Lineare Interpolation

Die lineare Interpolation (siehe Kapitel 2.5.4) wird in der Methode lerp umgesetzt. In Abbildung 5.6 sieht man, dass die konkrete Methode in zwei Ausführungen vorhanden ist. Einerseits als Interpolation zwischen zwei Ganzzahlwerten und andererseits zwischen zwei Fließkommawerten. Diese Unterteilung erklärt sich dadurch, dass die lineare Interpolation sowohl auf die einzelnen Komponenten des ARGB-Codes (siehe Kapitel 2.5.2), der als Ganzzahlwert dargestellt wird, als auch auf die Position der Linien und Pixel (siehe Kapitel 2.5.5) beim Image Warping, die als Fließkommawert dargestellt werden, um genaue Berechnungen durchführen zu können. Die Methode gibt einfach den interpolierten Wert aus zwei, jeweils vom Quell- und Zielbild, übergebenen Werten zurück.

```
private float lerp(float source, float target, float t){
    return source+(t*(target-source));
}

private int lerp(int source, int target, float t){
    return (int) (source + (t*(target-source)));
}
```

Abbildung 5.6: Lineare Interpolation mit Ganzzahl- und Fließkommawerten

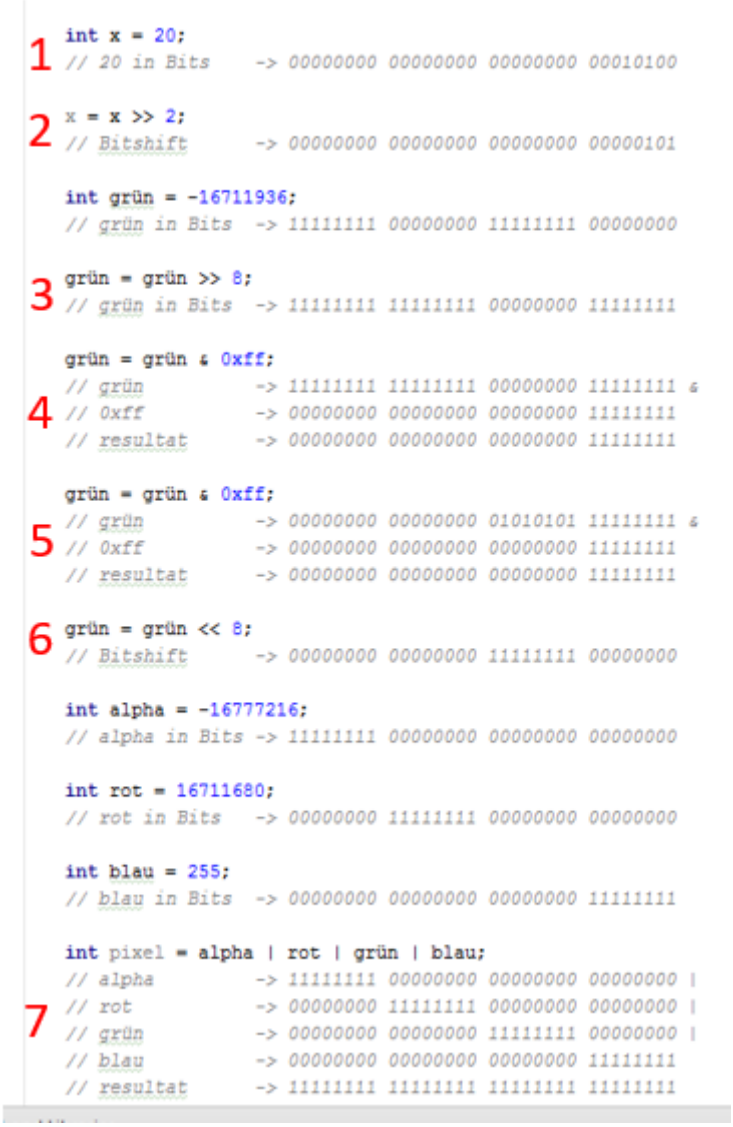
5.3.2 Aufbau einer Farbe

Die Farbe eines Bildpunkts wird als ein Integer dargestellt. In Java besteht ein Integer aus vier Bytes, also aus vier mal acht Bits. Dies ergibt eine perfekte Möglichkeit einen ARGB-Code (siehe Kapitel 2.5.2) in einem Integer, bei dem jedes der vier Bytes jeweils eine der vier Komponenten repräsentiert, unterzubringen. Im Kapitel 5.3.2 wird verdeutlicht, wie man aus einem einzelnen Integer, die Werte der vier einzelnen Bytes ausliest und diese jeweils in einem eigenen Integer unterbringen kann.

Bitmanipulation eines Integers

Die Abbildung 5.7, beschreibt die Anwendung des Bitshift-Operators (¿¿) und macht deutlicher wie man mit diesem die vier verschiedenen Bytes eines Integer auslesen und manipulieren kann. Der Bitshift in Schritt 1 bewirkt, dass sich alle Bits, wie in Schritt 2 gezeigt, um die Anzahl an Stellen, die nach dem Bitshift-Operator angegeben werden, nach rechts verschieben und die Bits die auf der linken Seite leer bleiben mit dem Bit, das ganz links gestanden ist, aufgefüllt werden. Wenn man die Bits nun in Oktetten (8-er Schritten), wie in Schritt 3, verschiebt, erhält man im ganz rechten Oktett, auch unterstes Oktett genannt, den Wert einer der vier Bytes. Um wirklich nur den Wert des untersten Oktetts zu erhalten, muss man den ganzen Integer, wie in Schritt

4, noch mit dem Wert 0xff (00000000 00000000 00000000 11111111) über den "&¨Operator verknüpfen. Dadurch werden die obersten drei Oktetten verborgen und nur der Wert des untersten erscheint. Um die einzelnen Komponenten wieder zu einem Integer zusammenzufassen, muss man zuerst, wie in Schritt 5, einfach nur jede Komponente wieder mit 0xff über den "&Operator verknüpfen. Für den Fall, dass diese verändert wurden und man wieder sicherstellt nur den Wert des jeweiligen Oktetts zu erhalten. Weiters verschiebt man die Oktetts, mittels eines umgekehrten Bitshifts (¡¡), wieder an die richtige Stelle, wie in Schritt 6 gezeigt. Und letztlich, in Schritt 7, werden diese über den Operator miteinander verknüpft, was dazu führt, dass sich alle Bytes wieder an der richtigen Stelle und in einem gemeinsamen Integer befinden.




```
int x = 20;
// 20 in Bits    -> 00000000 00000000 00000000 00010100

x = x >> 2;
// Bitshift      -> 00000000 00000000 00000000 00000101

int grün = -16711936;
// grün in Bits  -> 11111111 00000000 11111111 00000000

grün = grün >> 8;
// grün in Bits  -> 11111111 11111111 00000000 11111111

grün = grün & 0xff;
// grün          -> 11111111 11111111 00000000 11111111 &
// 0xff          -> 00000000 00000000 00000000 11111111
// resultat      -> 00000000 00000000 00000000 11111111

grün = grün & 0xff;
// grün          -> 00000000 00000000 01010101 11111111 &
// 0xff          -> 00000000 00000000 00000000 11111111
// resultat      -> 00000000 00000000 00000000 11111111

grün = grün << 8;
// Bitshift      -> 00000000 00000000 11111111 00000000

int alpha = -16777216;
// alpha in Bits -> 11111111 00000000 00000000 00000000

int rot = 16711680;
// rot in Bits   -> 00000000 11111111 00000000 00000000

int blau = 255;
// blau in Bits  -> 00000000 00000000 00000000 11111111

int pixel = alpha | rot | grün | blau;
// alpha         -> 11111111 00000000 00000000 00000000 |
// rot           -> 00000000 11111111 00000000 00000000 |
// grün          -> 00000000 00000000 11111111 00000000 |
// blau          -> 00000000 00000000 00000000 11111111
// resultat      -> 11111111 11111111 11111111 11111111
```

Abbildung 5.7: Bitmanipulation eines Integers

Bitmanipulation einer Farbe

Die Methode colorLerp, siehe Abbildung 5.8, implementiert die Funktion zur konkreten Veränderung einer Farbe. In Schritt 1 werden mittels Bitmanipulation, siehe Kapitel 5.3.2, die vier einzelnen

Komponenten aus dem Quell- und dem Zielpixel ausgelesen und über die Methode lerp, siehe Kapitel 5.3.1, miteinander kombiniert. Danach werden sie in Schritt 2 wieder zusammengefügt und zurückgegeben.

```
private int colorLerp(int source, int target, float t){
    int a = lerp( (source >> 24) & 0xff , (target >> 24) & 0xff , t );
1   int r = lerp( (source >> 16) & 0xff , (target >> 16) & 0xff , t );
    int g = lerp( (source >>  8) & 0xff , (target >>  8) & 0xff , t );
    int b = lerp( (source      ) & 0xff , (target      ) & 0xff , t );

2   return (a & 0xff) << 24 | (r & 0xff) << 16 | (g & 0xff) << 8 | (b & 0xff);
}
```

Abbildung 5.8: Bitmanipulation einer konkreten Farbe

In der Abbildung 5.8 bezeichnen *source* und *target* die Werte des Pixels aus dem Quell- und dem Ziebild und a, r, g und b sind die einzelnen Komponenten des ARGB-Codes des zusammengeführten Pixels.

5.3.3 Image Warping

Die Methode createMorphedImageIntermediate berechnet aus dem Quell- und dem Zielbild zwei Teilbilder, die jeweils eine verzerrte Variante des Quell- und des Zielbilds darstellen. Der wichtigste Teil dieser Methode ist die Schleife die in Abbildung 5.9 zu sehen ist. Diese berechnet die neue Position für den Zielpixel (siehe Kapitel 2.5.5) und somit den Pixel für das bereits verzerrte Bild. Dafür nimmt sie in Schritt 1, die durch lineare Interpolation errechneten Werte des Quell- und des Zielvektors, berechnet damit die Position und errechnet sich später in Schritt 2 die Gewichtung und die Abstandsvektorsumme (siehe Kapitel 2.6.3). Dieser Schleifendurchlauf wird für jede Linie durchlaufen, wobei dieser für jeden Pixel extra durchlaufen wird. Daraus resultieren bei einem Full HD Bild mit 1920 x 1080 Pixel und einer durchschnittlichen Zahl von 20 Linien pro Bild, 41 472 000 Schleifendurchläufe.

```
for(int j=0 ; j<aimed[P].length ; j++){
    float pi_x = lerp(current[P][j].getX(), aimed[P][j].getX(), PARAMETER_T);
1   float pi_y = lerp(current[P][j].getY(), aimed[P][j].getY(), PARAMETER_T);
    float qi_x = lerp(current[Q][j].getX(), aimed[Q][j].getX(), PARAMETER_T);
    float qi_y = lerp(current[Q][j].getY(), aimed[Q][j].getY(), PARAMETER_T);
    ●
    ●
    ●

    float weight = (float) Math.pow(Math.pow(pi_qi.len(),PARAMETER_P)/(distance + PARAMETER_A), PARAMETER_B);
2   distance_i_weighted = Point.mul(weight,distance_i);

    distance_sum.add(distance_i_weighted);
    weight_sum += weight;
```

Abbildung 5.9: Schleifendurchlauf zur Berechnung der Position des Zielpixels

In der Abbildung 5.9 bezeichnen *current* und *aimed* die Listen der Quell- und Zielvektoren, *pi* und *qi* sind die Endpunkte des durchlaufenen Linienpaares, *weight* und *weight_sum* sind die Gewichtung des durchlaufenen Linienpaares und die Gewichtungssumme. *distance_sum* ist die Abstandsvektorensumme.

5.3.4 Image Blending

Die Methode blend2Images, siehe Abbildung 5.10, erstellt ein einzelnes, neues Bild aus zwei übergebenen Bildern, indem sie diese beiden überblendet. Bei den beiden übergebenen Bildern handelt es sich allerdings nicht um Quell- und Zielbild, sondern bereits um die zwei durchs Image Warping (siehe Kapitel 5.3.3) erstellten Teilbilder. Zuerst erstellt die Methode blend2Images ein neues Integer-Array, das die Farbkodierung des Bildes enthält. Danach läuft sie die beiden Teilbilder durch und überblendet, die sich an der selben Stelle befindlichen Bildpunkte, mittels colorLerp (siehe Kapitel 5.3.2) und dem Parameter t. Daraus resultiert ein Zwischenbild, das das erste Teilbild überblendet mit dem zweiten Teilbild darstellt und somit ein fertiges Zwischenbild ist.

```
private int[] blend2Images(int[] firstIntermediate, int[] secondIntermediate){
    int[] result = new int[firstIntermediate.length];
    for(int i=0 ; i<result.length ; i++){
        result[i] = colorLerp(firstIntermediate[i], secondIntermediate[i], PARAMETER_T);
    }
    return result;
}
```

Abbildung 5.10: Überblendung zweier Bilder

In Abbildung 5.10 bezeichnen *firstIntermediate* und *secondIntermediate* die zwei Teilbilder, die überblendet werden und *result* das daraus resultierende Zwischenbild.

5.3.5 Kombination von Image Warping und Image Blending

Die Vereinigung von Image Warping (siehe Kapitel 5.3.3) und Image Blending (siehe Kapitel 5.3.4) wird in der Methode merge2Images realisiert (siehe Abbildung 5.11). Hier werden in Schritt 1 die beiden verzerrten Teilbilder erstellt und diese in Schritt 2, mittels blend2Images überblendet. Danach wird das fertige Zwischenbild zurückgegeben.

```
private int[] merge2Images(int[] sourcePicture, int[] targetPicture){
  1 int[] morphedSourceIntermediate = createMorphedImageIntermediate(sourcePicture, sourceFeatures, targetFeatures);
    int[] morphedTargetIntermediate = createMorphedImageIntermediate(targetPicture, targetFeatures, sourceFeatures);
  2 return blend2Images(morphedSourceIntermediate, morphedTargetIntermediate);
}
```

Abbildung 5.11: Zusammenführen von Image Warping und Image Blending

In der Abbildung 5.11 bezeichnen *sourcePicture* und *targetPicture* das Quell- und das Zielbild und *morphedSourceIntermediate* und *morphed TargetIntermediate* die verzerrten Teilbilder.

6 Beurteilung

In diesem Kapitel wird die Arbeit kurz zusammengefasst und es werden auf die Ergebnisse, die am Ende der Arbeit erreicht wurden, die Erfahrungen, die dem Team vermittelt wurden und die Verbesserungsvorschläge, die zukünftige ähnliche Entwicklungen erleichtern sollen, näher eingegangen.

6.0.1 Ergebnisse

Im Erstellungszeitraum der Diplomarbeit wurde eine mobile Anwendung für die Android-Plattform implementiert, die es benutzenden Personen erlaubt täglich erinnert zu werden, ein Foto aufzunehmen, dieses auf sozialen Plattformen teilen zu können und mithilfe eines Morphing-Algorithmus alle aufgenommenen Bilder in ein Video zusammenzufügen, das einen flüssigen Übergang zwischen den einzelnen Fotos gewährleistet.
Die Anwendung wurde in der Programmiersprache Java, mithilfe der Entwicklungsumgebung Android Studio, entwickelt. Der Algorithmus wurde zuerst am PC, unabhängig von der mobilen Anwendung, vorgeschrieben und im Nachhinein implementiert.

6.0.2 Erfahrungen

Im Rahmen dieser Arbeit wurden einige - für das Team neue - Ansätze für die Entwicklung eines Projekts in Erfahrung gebracht. Das Auseinandersetzen mit einem mathematisch komplizierten Verfahren, das zur gleichen Zeit, vom Programmieraufwand her gesehen, relativ einfach gestaltet ist, war eine völlig neue, anspruchsvolle und gleichzeitig spannende Erfahrung.
In der schulischen Ausbildung wird man mit immer höheren Technologien gefordert, wobei sich die selbst erstellte Logik dahinter, zumeist in Grenzen hält. In dieser Arbeit war der technologische Aufwand minimal, da der Morphing-Algorithmus eine Aneinanderreihung von trivialen, mathematischen Operationen wie Addition, Multiplikation und Vektorrechnung bildet. Die ganze selbsterstellte Logik, die das Gesamtbild des Algorithmus darstellt, ist wesentlich komplexer. Daher erfordert eine solche Arbeit ein intensives Studium und tiefes Verständnis für die verwendeten Verfahren.
Weiters ist auch die Erstellung einer vollständigen, mobilen Anwendung, in Rücksicht darauf, dass diese von Benutzern die sich mit der Materie nicht auskennen, benutzt werden soll, eine wertvolle Erfahrung. Erst sobald eine selbst erstellte Anwendung wirklich einem Benutzer zur Verfügung stehen soll, schätzt man die Benutzerfreundlichkeit, die viele alltäglich verwendeten Anwendungen an den Tag legen, umso mehr. Die von der Anwendung getrennte Entwicklung des Algorithmus stellte sich als sehr hilfreich heraus, da bei beiden unerwartet Fehler aufgetreten sind und auf diese Weise sehr viel schneller behoben werden konnten.

6.0.3 Verbesserungsvorschläge

Wie bei der Erstellung einer jeden Arbeit, gibt es Wege, die das ganze einfacher gestaltet hätten. Im Falle dieser Diplomarbeit wäre es wohl sinnvoller gewesen, die Morphingfunktion auf einen externen Webservice auszulagern. Da der Prozessor eines Mobiltelefons die Berechnungen um einiges langsamer, als ein herkömmliche Prozessor eines Desktop-Computers, durchführt.

Abbildungsverzeichnis

Literaturverzeichnis

[1] P. Viswanathan, "The pros and cons of native apps and mobile web apps," last Visited: 27.02.2017. [Online]. Available: https://www.lifewire.com/pros-and-cons-of-native-apps-and-mobile-web-apps-2373173/

[2] "Android ndk — android developers," last Visited: 03.04.2017. [Online]. Available: https://developer.android.com/ndk/index.html

[3] unbekannt, "Unterschiede und vergleich native apps vs. web apps," last Visited: 01.03.2017. [Online]. Available: http://www.app-entwickler-verzeichnis.de/faq-app-entwicklung/11-definitionen/107-unterschiede-und-vergleich-native-apps-vs-web-apps

[4] "Fragments — android developers," last Visited: 12.03.2017. [Online]. Available: https://developer.android.com/guide/components/fragments.html

[5] "App manifest — android developers," last Visited: 23.03.2017. [Online]. Available: https://developer.android.com/guide/topics/manifest/manifest-intro.html

[6] "Gradle wikipedia," last Visited: 23.03.2017. [Online]. Available: https://de.wikipedia.org/wiki/Gradle

[7] "Intent — android developers," last Visited: 12.03.2017. [Online]. Available: https://developer.android.com/reference/android/content/Intent.html

[8] "Snackbar — android developers," last Visited: 20.03.2017. [Online]. Available: https://developer.android.com/reference/android/support/design/widget/Snackbar.html

[9] "Morphing," last Visited: 2017.03.20. [Online]. Available: https://commons.wikimedia.org/wiki/Category:Morphing?uselang=de

[10] "Argb-farbmodell," last Visited: 2017.03.20. [Online]. Available: http://www.css3.info/preview/rgba/

[11] "Lineare interpolation," last Visited: 2017.03.20. [Online]. Available: http://www.mathepedia.de/Interpolation.aspx

[12] "Feature-based image metamorphosis," last Visited: 15.03.2017. [Online]. Available: https://www.cs.princeton.edu/courses/archive/fall00/cs426/papers/beier92.pdf

[13] "Uml designer - zur erstellung des klassendiagramms," last Visited: 04.04.2017. [Online]. Available: http://www.umldesigner.org/

[14] "Services — android developers," last Visited: 12.03.2017. [Online]. Available: https://developer.android.com/guide/components/services.html

[15] "Alarmmanager — android developers," last Visited: 12.03.2017. [Online]. Available: https://developer.android.com/reference/android/app/AlarmManager.html

[16] "Asynctasks — android developers," last Visited: 12.03.2017. [Online]. Available: https://developer.android.com/reference/android/os/AsyncTask.html

[17] "Jcodec," last Visited: 04.04.2017. [Online]. Available: http://jcodec.org/

[18] "Scheduling repeating alarms — android developers," last Visited: 20.03.2017. [Online]. Available: https://developer.android.com/training/scheduling/alarms.html

[19] "Pendingintent — android developers," last Visited: 20.03.2017. [Online]. Available: https://developer.android.com/reference/android/app/PendingIntent.html

[20] "Get started with the mobile vision api," last Visited: 26.03.2017. [Online]. Available: https://developers.google.com/vision/android/getting-started

[21] "Facial landmark detection," last Visited: 26.03.2017. [Online]. Available: http://www.learnopencv.com/facial-landmark-detection/