FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

ASSIGNMENT OF MASTER'S THESIS

Title:	Example-Based Stylization on Mobile Devices
Student:	Bc. Aneta Moravcová
Supervisor:	Ing. Ondřej Texler
Study Programme:	Informatics
Study Branch:	Knowledge Engineering
Department:	Department of Applied Mathematics
Validity:	Until the end of winter semester 2019/20

Instructions

Explore state-of-the-art in the field of style transfer based on guided texture synthesis, focus namely on methods which handle human faces [1] and can perform synthesis in real-time [2]. Implement selected method on a mobile device. Find the most suitable approach to detect facial landmarks possibly based on Convolutional Neural Network that would enable creation of semantically meaningful guidance for the synthesis. Implement a mobile application that transfers user specified artistic style to a photo or a video of a human face captured by built-in camera. Try to achieve interactive response on currently available mobile devices and discuss the trade-off between the stylization quality and the stylization speed.

References

[1] J. Fišer, O. Jamriška, D. Simons, E. Shechtman, J. Lu, P. Asente, M. Lukáč, and D. Sýkora: Example-Based Synthesis of Stylized Facial Animations. ACM Transactions on Graphics 36, 4 (2017). (SIGGRAPH, Los Angeles, USA, July 2017)
[2] D. Sýkora, O. Jamriška, O. Texler, J. Fišer, M. Lukáč, J. Lu, and E. Shechtman: StyleBlit: Fast Example-Based Stylization with Local Guidance. In Computer Graphics Forum. (Eurographics, Genova, Italy, May 2019)

Ing. Karel Klouda, Ph.D.
Head of Department

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
Dean

Prague July 9, 2018

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Master's thesis

Example-Based Stylization on Mobile Devices

Bc. Aneta Moravcová

Department of Applied Mathematics
Supervisor: Ing. Ondřej Texler

May 2, 2019

Acknowledgements

I would like to thank my supervisor Ing. Ondřej Texler for his excellent guidance and helpful advice. My gratitude also belongs to prof. Ing. Daniel Sýkora, Ph.D. and Ing. Ondřej Jamriška for their valuable comments. Further, I thank my family for supporting me during my studies.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as school work under the provisions of Article 60(1) of the Act.

In Prague on May 2, 2019 .....

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Moravcová, Aneta. *Example-Based Stylization on Mobile Devices.* Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2019.

Abstrakt

Tato práce se zabývá realizací Android mobilní aplikace umožňující přenos výtvarného stylu z dané předlohy na lidskou tvář zachycenou vestavěnou kamerou. Jedinečnost naší aplikace spočívá v nezávislosti na jakýchkoliv online vypočetních zdrojích. Celý proces stylizace je počítán offline mobilním zařízením. Mobilní aplikace používá existující detektor k získání důležitých orientačních bodů ve snímaném obličeji a nově publikovaný efektivní algoritmus pro přenos výtvarného stylu podle předlohy - StyleBlit. V naší implementaci usilujeme o co nejsvižnější odezvu aplikace a zároveň o věrné napodobení uměleckého stylu.

Klíčová slova Přenos výtvarného stylu, stylizace obličeje, stylizace podle předlohy, vedená syntéza textury, mobilní zařízení, mobilní aplikace, Android.

Abstract

This thesis deals with an implementation of Android mobile application allowing example-based style transfer from a given style exemplar to a human face captured by a built-in camera. The uniqueness of our application consists in the independence on any online computing resources. The entire process of stylization is computed offline by a mobile device. The mobile application uses an existing face detector for obtaining essential landmarks of a captured face, and newly released efficient example-based style transfer algorithm - StyleBlit. In our implementation, we seek for as fast as possible response of the application together with faithful reproduction of an artistic style.

Keywords Style transfer, face stylization, example-based stylization, guided texture synthesis, mobile device, mobile application, Android.

Contents

List of Figures

List of Tables

Introduction

Example-based style transfer is a complex problem and a popular research topic. With increasing hardware performance, the interactive image stylization on mobile devices becomes possible. In this thesis, we focus on the interactive stylization of human faces. In the style transfer process, it is possible to produce a stylized video sequence of a human facial performance. Arbitrary long video sequence can be synthesized by just a single style exemplar image.

Recently, neural-based style transfer has expanded and become very popular. There are many overwhelming applications, but none of them is able to preserve local visual features of the style exemplar well. The result of neural-based stylization is often distorted, sometimes even the overall appearance notably differs from the original style (see especially the first and the last column in Figure 1.1). The absence of textural details is a significant issue because these low-level characteristics actually define the artistic style.

We combine the non-neural approach of Fišer et al. 2017 [1] for face style transfer and the efficient *StyleBlit* algorithm [2] to create a mobile application that allows interactive face stylization with a presence of style features and preservation of human's identity as well. The input style example does not have to have the artistic character exclusively, but also the photorealistic portrait, like a photo of a statue or another human's face, is possible to use.

A key idea that distinguishes regular texture synthesis from style transfer is controlling the stylization process by guidance channels. This guidance helps to achieve semantically meaningful style transfer. Thanks to these channels, it is possible to transfer a specific coherent area in the style exemplar to a corresponding area in the target image.

Thesis' goal

The primary goal of this thesis is to implement a mobile application, that transfers a style exemplar to a human face with emphasis on preservation of both subject's identity and visual features of the given style exemplar. We

make an effort on synthesis computational time to be as low as possible but with decent stylization results.

Chapter 1

Related Work

Related work consists of two sections. The first section describes related research about all relevant style transfer approaches. The second section deals with existing mobile applications allowing the stylization.

1.1 Related Research

In recent years, many different approaches in photographs and video stylization have been published. They can be divided into filtering-based and example-based techniques.

Filtering-based techniques (DiPaola 2007 [3], Gooch et al. 2004 [4], Tresset and Leymarie 2005 [5], Yang et al. 2010 [6]) only combine image processing methods like thresholding, segmentation, edge-detection, blurring. The range of output is limited by the visual properties of used image processing filters.

Example-based techniques mitigate the disadvantage of limited visual range by the possibility to add an arbitrary style exemplar. In an approach by Chen et al. 2002 [7], 2004 [8], 2002 [9]; Meng et al. 2010 [10]; Zhang et al. 2014 [11] the visually important parts of the face (eyes, nose, mouth, hair) are decomposed, and an artist stylizes them separately. Those stylized pieces are then spatially distributed and composed together to meet the specific proportions of the target face. The key drawbacks are that individual reused templates are often noticeable, and it is difficult to obtain a truer representation of specific visual characteristics of the target face without additional exemplars. Other example-based techniques use multiscale Markov Random Fields (Li et al. 2011 [12]; Wang et al. 2013b [13], 2014 [14]; Wang and Tang 2009 [15]; Zhou et al. 2012 [16]). They use a larger database of photo-style exemplar pairs, which can reproduce a higher variety of target faces. The disadvantage is that an artist has to prepare a lot of photo-style exemplars for different subjects and it might be time-consuming. Techniques that can decompose style to individual strokes (Berger et al. 2013 [17]; Wang et al. 2013a [18]; Zhao and

Zhu 2011 [19]) are able to partially mitigate this drawback. However, not all of the styles are easy to reduce to stroke-based decomposition.

In 2016, Gatys et al. [20] showed that responses of deep neural networks (e.g., VGG by Simonyan and Zisserman 2014 [21]) can be used to guide texture synthesis in the same parametric fashion as in Portilla and Simoncelli 2000 [22]. Gatys' approach produces impressive results, needs only one exemplar image, and can handle various styles. However, due to its optimization nature, the method is very slow. A faster solution based on feed-forward networks later provided Johnson et al. 2016 [23]. Ruder et al. 2016 [24] extended the original approach to generate stylized videos, and Selim et al. 2016 [25] presented improvements that give better results for portrait stylization.

A fundamental limitation of neural-based style transfer approaches is the inability to reproduce textural details of a style faithfully (see figure TODO). This key weakness solves a solution based on guided non-parametric texture synthesis introduced by Fišer et al. 2016 [26]. In this approach, guidance channels are obtained from 3D models; hence this method is not directly applicable to portraits.

Furthermore, there are approaches that do not transfer artistic style, but a specific photographic look (Kemelmacher-Shlizerman 2016 [27]; Shen et al. 2016 [28]; Shih et al. 2014 [29]; Yang et al. 2015 [30]). These techniques provide ideas (e.g., contrast enhancement, intensity levels equalization) that are useful in our domain.

However, none of the above mentioned approaches provide a solution for faithful artistic style transfer for faces, when computational budget is limited. The most relevant approach for our application that solves this problem, is *Example-Based Synthesis of Stylized Facial Animations* introduced by Fišer et al. 2017 [1]. Another highly important technique is *StyleBlit: Fast Example-Based Stylization with Local Guidance* presented by Sýkora et al. 2019 [2]. The first mentioned method provides stylization of portrait videos and is able to perfectly preserve the visual details of given artistic style. However, it is optimization based and strong GPU is required for this method to run fast. The second method secures the efficiency of example-based style transfer and its key advantage is ability to preserve low-level features of the transferred texture. Both techniques are described in more detail later in this thesis.

1.2 Related Mobile Applications

There exists a massive amount of Android and iOS application for style transfer and image stylization. Essentially, all of these applications use a neural network allowing style transfer from various predefined style exemplars (witch a neural network is trained on) to arbitrary photographs. Most of the reviewed applications need the internet connection to compute the stylization on a server and do not have a real-time response (the computational time of

the most popular applications is usually more than 5 seconds). The neural-based style transfer gives nice results at first glance, but is highly limited in the ability to faithfully reproduce artistic textural details, such as brush strokes. The reason is that neural-based style transfer techniques match statistics (e.g., features of VGG) of a style and a target instead of copying actual patches or chunks (coherent areas of pixels) of the style image to form the desired content.

In the picture 1.1, we can see a few results of the most popular mobile applications in the field of style transfer. At first glimpse, there are obvious all those limitations of neural-based stylization like the inability to faithfully preserve the artistic style and incapability to respect the content semantics of the target image.

Probably the most popular mobile application providing a neural-based style transfer is *Prisma Photo Editor*[1] by Prisma Labs, Inc. It was released in 2016 for Android and iOS, and the computation is performed online on Prisma Labs' servers (not real-time).

GoArt[2] by Chengdu Everimaging Science and Technology Co., Ltd is another very popular neural-based application. It was released in 2016 for Android and iOS and need the internet connection for its computation.

Next popular mobile application is *Varnist*[3] by Varnist, which was released in 2017 and it is based on neural network. It is available for Android and iOS and computes results online in a few seconds.

Another mobile application that uses deep learning algorithm for style transfer is called *Deep Art Effects*[4] released by Deep Art Effects GmbH in 2016 for Android and iOS. As well as above mentioned applications, it needs the internet connection and does not have a real-time response.

There are also applications that compute results offline, like *Copista*[5], *Pierra*, *Stylish*, and lots of others.

[1] www.prisma-ai.com
[2] goart.fotor.com
[3] www.varnist.com
[4] www.deeparteffects.com
[5] www.tinyline.com

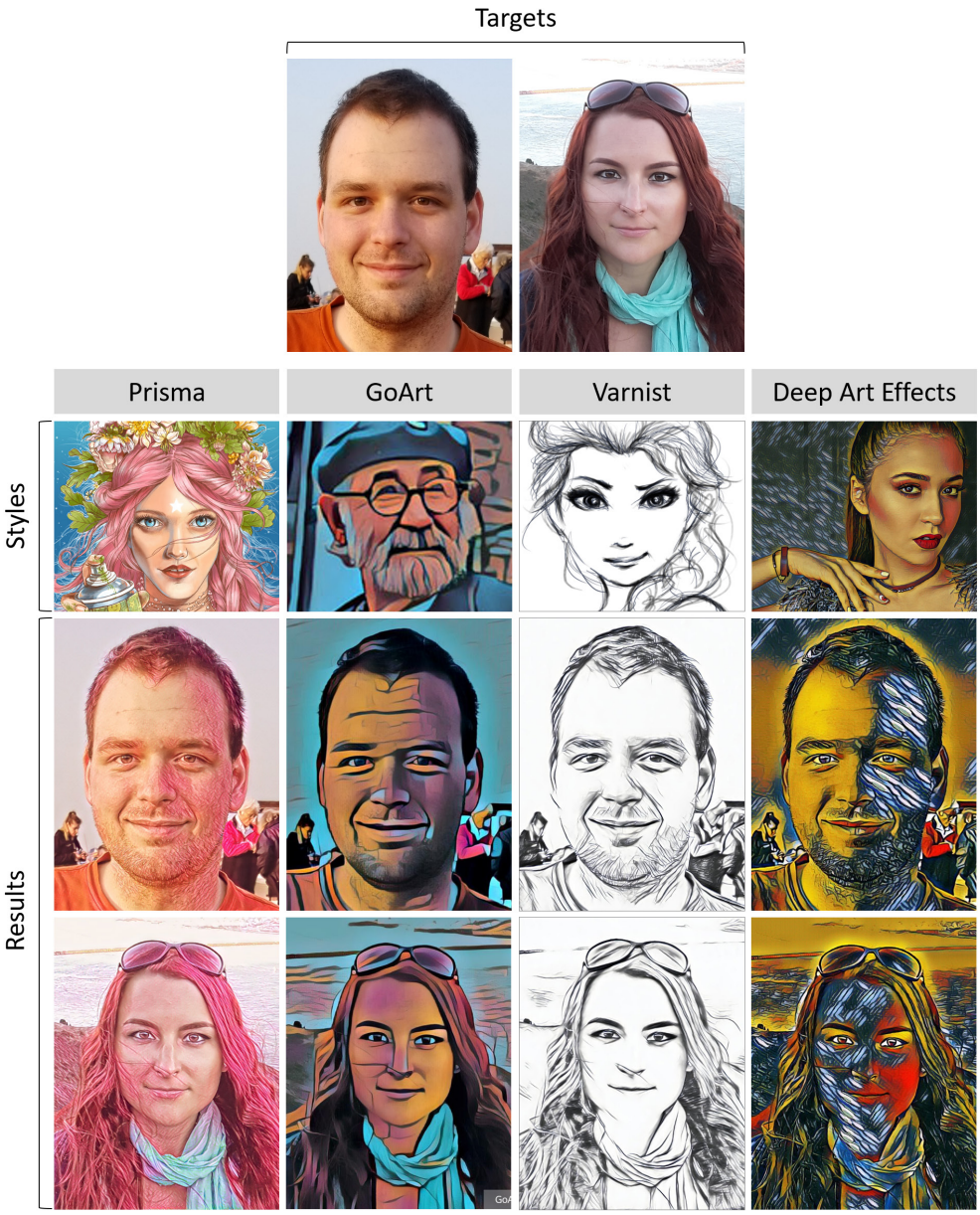

Figure 1.1: *Results of style transfer from related mobile applications.* Style and result images: ©Prisma, ©GoArt, ©Varnist, ©Deep Art Effects.

Chapter 2

The Most Relevant Related Approaches

In this chapter, the two most relevant approaches we get primary inspired with are described in more details. The first technique, *FaceStyle*, specializes in the stylization of facial animations that preserves detailed characteristics of the artistic style. The second method provides an efficient algorithm *StyleBlit* for the example-based style transfer.

2.1 FaceStyle: Example-Based Synthesis of Stylized Facial Animations

An approach called *Example-Based Synthesis of Stylized Facial Animations* was introduced by Fišer et al. 2017 [1] and provides a non-parametric texture synthesis. This method preserves both subject's identity and low-level visual features of the given artistic style. In contrast with some related techniques, *FaceStyle* does not expect explicit image warping (precise alignment of the style exemplar with the target picture). The goal of this approach is the ability to transfer the style from one style exemplar to a target video - sequence of target frames.

2.1.1 Method Description

There are two inputs to FaceStyle technique - a *style exemplar* image S of a portrait and a *target* video T of a human face. The output is a stylized video sequence O which carries the style visual features and respects the characteristics of subject's face. As a solution, it is applied the guided texture synthesis that preserves style details (Fišer et al. 2016 [26]) and is based on non-parametric texture synthesis (Kwatra et al. 2005 [31]; Wexler et al. 2007 [32]). The goal of guided texture synthesis is to generate the result image,

where a set of guidance channels is used to enable the faithful and semantically meaningful style transfer.

2.1.2 Guidance Channels

Guidance channels (see Figure 2.1) have to fulfill a few conditions to secure the rich and semantically meaningful style transfer; and are always generated for style exemplar and for each target frame T_i of the video.

Segmentation guide G_{seg}. Construction of segmentation guide is motivated by the fact that various semantic regions of artistic style typically have a different character (e.g., smaller and larger brush strokes for different regions). Hence, the segmentation guide divides the head into individual segments: skin, hair, eyes, eyebrows, nose, lips and oral cavity. To create the G_{seg}, the soft head mask (a pixel categorization to head and background) is obtained with the help of detected facial landmarks and closed-form matting (Levin et al. 2008 [33]). Next, the skin soft mask is constructed by using a simple statistical model of the skin (Gong and Sakauchi 1995 [34]). The pixel-wise difference between the head and skin masks determines the hair segment. Finally, the remaining regions (eyes, eyebrows, nose, lips, oral cavity) are defined by the detected facial landmarks. To avoid sharp transitions, the diffusion curves method (Orzan et al. 2008 [35]) is used to blur the segment boundaries.

Position guide G_{pos}. The position guide supports the transfer of style chunks to their corresponding positions in target image T_i. For the style exemplar, the generating of the G_{pos} is straightforward - each pixel receives a color determined by its (x, y) coordinates. The G_{pos} for the target image T_i is obtained by using moving least squares deformation (Schaefer et al. 2006 [36]), where connected positions of target facial landmarks corresponding to style facial landmarks are used as control lines for the image warp. The white grid in the Figure 2.1 is displayed only for illustrative purposes to make the MLS deformation from style to target more apparent.

Appearance guide G_{app}. The G_{app} helps to preserve wrinkles, facial characteristics and overall appearance of the target face. The temporal guide is obtained by converting the style and target images to grayscale and then adjusting the global intensity levels and local contrast values of the target frame T_i to match those in the style image. By increasing the weight for the G_{app} the resulting face looks more like the original subject, but the transferred exemplar's coherent areas of pixels (chunks) are smaller. It leads to lower preservation of the texture detail, and the overall look of the stylization result may seem blurry.

Temporal guide G_{temp}. To reproduce the appearance of a real hand-drawn video, there need to be created the temporal guide, that secures a certain amount of temporal noise. Generating the G_{temp} is inspired by the method *Color Me Noisy* introduced by Fišer et al. 2014 [37] that preserves

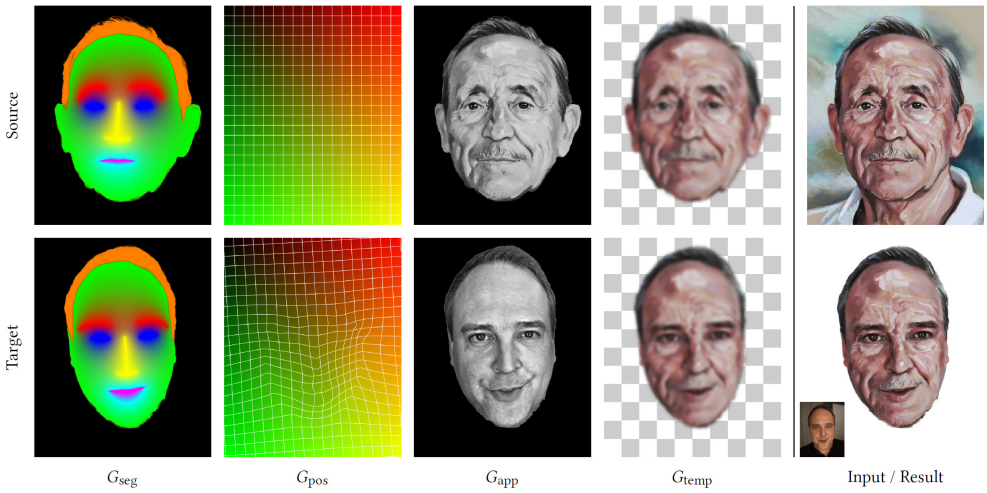

Figure 2.1: *Overview of guidance channels that enable faithful style transfer. The segmentation guide G_{seg} divides the source and target head into individual regions (skin, hair, eyes, eyebrows, nose, lips, oral cavity). The positional guide G_{pos} supports the transfer of style chunks to their corresponding positions in target image. The grid is displayed only for illustrative purposes. The appearance guide G_{app} highlight the subject's identity. The temporal guide G_{temp} blurs the source exemplar and previous target frame to preserve a certain amount of temporal noise of the stylized video. On the right, there are style exemplar and stylized result with the original target frame.* Image source: Fišer et al. 2017 [1].

the temporal coherence only at lower frequencies. The style image and the previously synthesized frame O_{t-1} are blurred and used as a temporal guide for the current frame.

2.1.3 Synthesis

Once all the guiding channels G_{seg}, G_{pos}, G_{app}, G_{temp} are generated, then there is used the guided texture synthesis algorithm called *StyLit: Illumination-Guided Example-Based Stylization of 3D Renderings* presented by Fišer et al. 2016 [26], a key advantage of which is the ability to suppress the so-called "wash-out" effect. This undesired effect described by Newson et al. 2014 [38] causes that the resultant texture has smooth or blurry parts. It occurs in the case when a small number of the same or similar chunks are extensively used to build the new texture. The *StyLit* method is based on light propagation in the rendered scene and can distinguish among different semantic regions of artistic style to give results with appearance more closer to realistic artwork.

2.1.4 Performance

Generating of guidance channels is an extremely time-consuming process. On a 3 GHz quad-core CPU for a one-megapixel frame, the generating of all necessary guiding channels takes around 30 seconds. The stylization process itself is also computationally very demanding, with the error metric approximation, a one-megapixel frame can be synthesized in 3 minutes on the CPU and in 5 seconds on the GPU (GeForce GTX 970).

2.2 StyleBlit: Fast Example-Based Stylization with Local Guidance

StyleBlit: Fast Example-Based Stylization with Local Guidance is a newly released efficient example-based style transfer algorithm presented by Sýkora et al. 2019 [2] that is able to provide high-quality stylized renderings in real-time on a single-core CPU. This approach is usefull for interactive mobile applications where available hardware resources are limited. This algorithm is especially convenient for stylization with the assistance of guidance channels, which are necessary for achieving a semantically meaningful transfer of content from the style to the target image. A key advantage is that StyleBlit preserves high-frequency characteristics of the texture since larger chunks (coherent areas of pixels) of the style image are transferred to form the target content.

2.2.1 Basic Algorithm

The basic brute-force algorithm can be implemented by available pseudocode (see Algorithm 1).

The location and shape of each chunk need to be estimated. Hence, for each pixel (seed) in the target image (random pick or scan-line order) is found its corresponding coordinates in style image (see Figure 2.2 a, b). Next, the error is calculated - the difference between the guidance values of target and source pixels in regions around the seed. The target pixels with an error lower than a user-defined threshold belong to the current chunk (see Figure 2.2 c). Finally, this coherent area of example pixels is cloned into the target image (see Figure 2.2 d). This process is repeated until all pixels in the target image are covered (see Figure 2.2 C_T).

The resulting image represents the original style faithfully even though StyleBlit does not explicitly enforce textural coherence. Seams between individual chunks are usually not apparent because hand-painted style exemplars are typically stochastic. Another reason is that guidance channels are continuous and smooth, so the resulting chunks are roughly aligned. Also, seams are hard to spot if every chunk has a different and irregular shape.

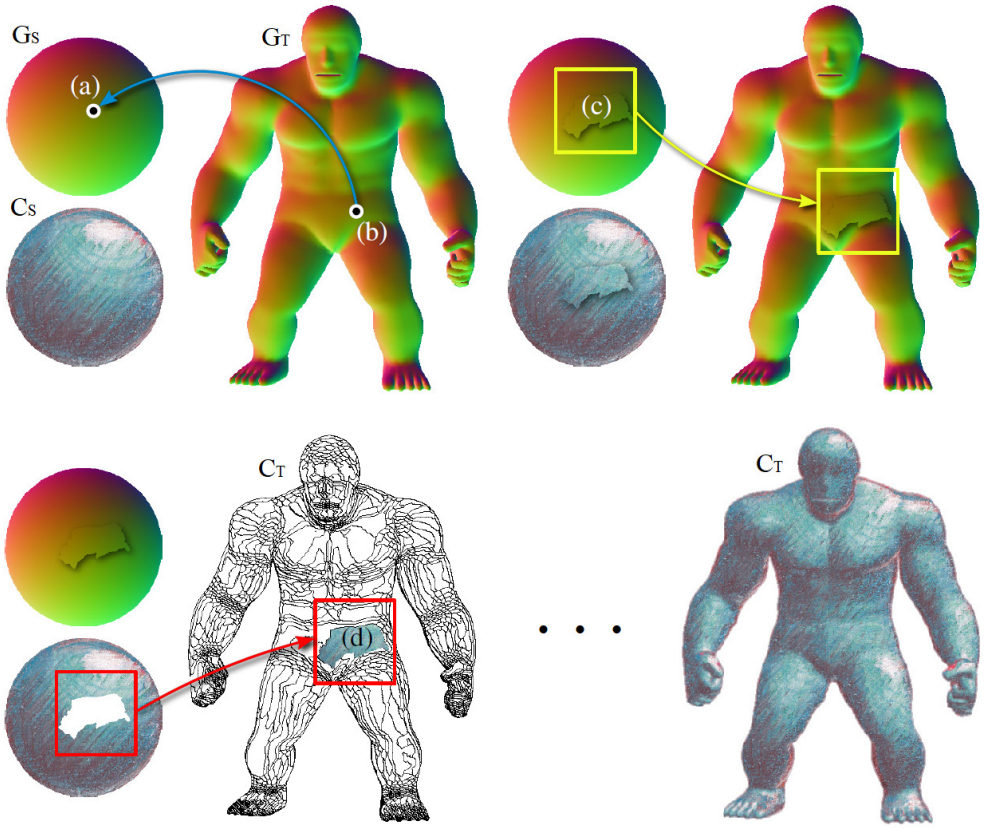

Figure 2.2: *The core idea behind StyleBlit. In this case, the artistic style from sphere C_S is transferred onto a more complex 3D object C_T. As the local guidance, normal maps G_S and G_T are used. For each target pixel (b) is retrieved the corresponding location of source pixel (a) by using its guidance value. The error between the guidance values of source and target pixels in regions around the seed is calculated. Pixels with the error lower than some threshold belong to the same chunk (c). Finally, the chunk is cloned into the target (d). By repeating this process, the final mosaic C_T is created.* Image source: Sýkora et al. 2019 [2].

Algorithm 1: StyleBlit [2]

Inputs : source style exemplar C_S, source guides G_S, target guides G_T, threshold t.

Output: target stylized image C_T.

```
StyleBlit():
    for each pixel p ∈ C_T do
        if C_T[p] is empty then
            u* = argmin_u ||G_T[p] − G_S[u]||
            for each pixel q ∈ C_S do
                if C_T[p + (q − u*)] is empty then
                    e = ||G_T[p + (q − u*)] − G_S[q]||
                    if e < t then
                        C_T[p + (q − u*)] = C_S[q]
```

There also exists a more efficient fully parallel version of StyleBlit algorithm for GPU. However, for our purposes, the basic sequential CPU algorithm is sufficient.

2.2.2 Performance

StyleBlit attains significant performance acceleration by a set of fast and simple operations at the level of individual pixels. On a single core modern CPU, the stylization process of one-megapixel image reaches 10 frames per second. On a common GPU, it is possible to achieve more than 100 frames per second at a 4K UHD resolution.

Chapter 3

Method

In this section, our approach to the realization of the mobile application is presented. Our method is based on approaches from Chapter 2 and consists of several consecutive steps. These steps and the definition of the problem are described in this chapter together with some related concepts.

3.1 Problem Definition

Inputs to the example-based style transfer are a *style exemplar* image S of a portrait and a *target* image T of a human face. There is an assumption, that target subject is facing the built-in camera, and no other objects are in front of the subject's face. The output is a *stylized image* O obtained by the example-based style transfer that preserves textural detail and overall appearance of the given style examplar and also the subject's identity. To solve this task, we perform the following consecutive steps:

1. Facial landmarks detection.

2. Generating a set of guiding channels.

3. Example-based texture synthesis using the StyleBlit algorithm.

In the subsequent sections, these three subtasks are described in more details.

3.2 Facial Landmarks Detection

One of the most important tasks we had to solve was detection of facial landmarks. Without these points would never be the semantically meaningful style transfer possible. We needed some API, library or framework, that would be available for the Android platform. We considered the following options.

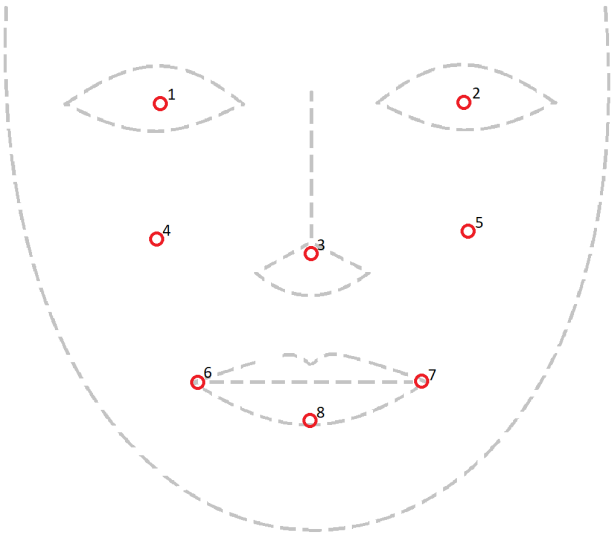

Figure 3.1: *Layout of facial landmarks obtained by Google Mobile Vision API detector. There are only 8 points detected which is strongly insufficient.*

3.2.1 Mobile Vision API by Google

This API is part of the *Google Play Services* and provides a detector of faces and facial landmarks. The API is very easy to use, but the number of facial landmarks and their spatial layout (see Figure 3.1) is for our purposes completely insufficient. To transfer the artistic style from an exemplar to the target face properly, we need to know the exact shape and position of all important parts of the face (e.g., eyes, mouth, nose).

3.2.2 Dlib Library

Another option we examined was a general purpose cross-platform C++ library *Dlib* that is possible to use on Android platform via JNI. *Dlib* is the open-source library released under a Boost Software License. Development began in 2002, and after focusing extensions on machine learning tools, *Dlib* was published in the Journal of Machine Learning Research (King 2009 [39]). *Dlib* contains several face detector implementations, and we considered the two most relevant ones of them. The first detector is based on ERT and the paper behind this technique is called *One Millisecond Face Alignment with an Ensemble of Regression Trees* and was presented by Kazemi and Sullivan [40] at Computer Vision and Pattern Recognition conference in 2014. The second newer detector is CNN based. Both implementations have pros and cons and it depends on the use case. ERT face detector is faster than CNN based, thus it is more suitable for real-time videos. However, CNN detector is able to

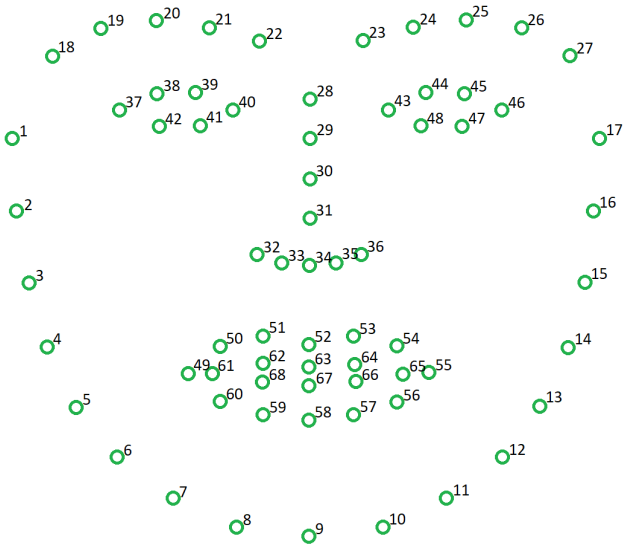

Figure 3.2: *Layout of facial landmarks returned by Dlib detector. The face detector from Dlib library is able to deliver 68 points, thus exact shape of eyes, mouth, nose, and other can be reconstructed.*

detect faces in much more angles. For our use case, the higher speed of ERT based detector is decisive. The inability to detect faces in plentiful angles is a minor problem since we presume that stylized subject is mostly facing the camera frontally. So, we decided to use the ERT based face detector in our application. Totally, this face detector returns remarkable 68 facial landmarks (their disposition shows Figure 3.2). In the following sections, we say more about ERT method for facial landmarks detection.

3.2.2.1 Ensemble of Regression Trees

This technique uses a cascade of regressors to estimate the position of the facial landmarks in a given image in milliseconds and with high-quality predictions. Each regressor in the cascade predicts a new estimate of landmarks positions from the input image and the previous estimate. Regressors make their predictions based on a fast and simple feature - pixel intensities. These predictions are then improved iteratively. The gradient tree boosting algorithm that optimizes the sum of square error loss is used for learning each regressor in the cascade.

3.2.3 Summary

After examination some other approaches in facial landmarks detection, we concluded that *Dlib* is currently the most suitable available tool for our requirements. All 68 points delivered by *Dlib*'s face detector are essential for rich and semantically meaningful style transfer.

It is necessary to detect facial landmarks in both images - style exemplar and target photo. The set of style exemplars we use in our application is static, so all these 68 facial points are detected in advance to save computational time. While target images are continuously changing, thus facial landmarks are detected at the beginning of the stylization process. Once we have positions of all facial landmarks in style and also in the target image, we are able to generate the *position guide* G_{pos} to support the transfer of style chunks to the corresponding regions in the target photo.

3.3 Guidance Channels

After facial landmarks detection, the next step is generating guiding channels that are tailored to both style and target portraits. Equally as prepared facial landmarks in the previous step, also guides for each style exemplar are generated in advance for a purpose to reduce computational time. Guidance channels enable faithful and semantically meaningful style transfer. In our application, we use *positional guide* G_{pos} and *appearance guide* G_{app} from approach by Fišer et al. [1]. However, since mobile devices have limited computing resources and we seek for the real-time performance, we use simplified versions of these guiding channels.

3.3.1 Positional Guide G_{pos}

To create the positional guide G_{pos} for style exemplar, we simply set each pixel to a color that is determined by (x, y) coordinates of the pixel. We use only green and red color channels and their values are normalized to the range of $0 - 255$ (see Figure 3.3). Once we have this exemplar's positional guide, we warp it using moving least squares (MLS) deformation to fit facial landmarks of the style exemplar onto the corresponding ones in the target image. This warped exemplar's G_{pos} represents the positional guide for the target image. In the Figure 3.4, we can see the MLS deformation from the exemplar's G_{pos} to the target's G_{pos} (white grid is shown only for visualization purposes).

3.3.1.1 Moving Least Squares Deformation.

Image deformation technique based on linear *Moving Least Squares* presented by Schaefer et al. 2006 [36] provides solution for the real-time image warping. To perform this deformation, there is a set of control points or lines that

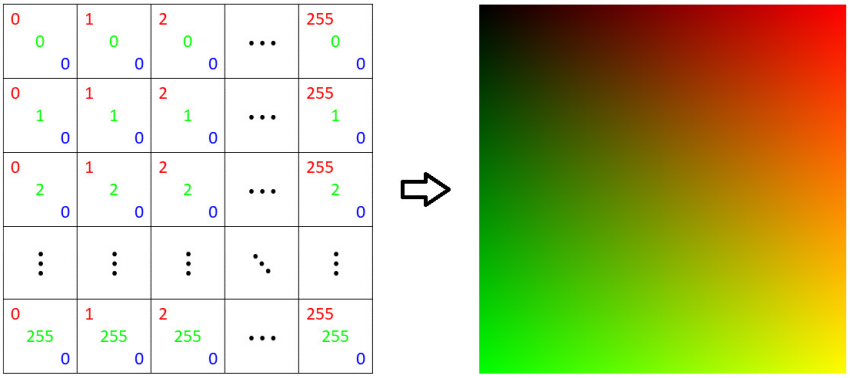

Figure 3.3: *The design method of the style exemplar's positional guide* G_{pos}.

handle the warping. Once the handles' positions are modified, the image deforms realistically and intuitively. According to Schaefer et al. [36], the deformation is a function f, which maps points in the original image to points in the deformed image. This function f is applied to each point v in the original undeformed image. For function f, set of original handles p and set of handles q moved to new positions (in our case, exemplar's and target's facial landmarks), the following properties are in effect:

- *Interpolation:* The set p maps directly to set q under warping (i.e., $f(p_i) = q_i$).
- *Smoothness:* f produces smooth deformations.
- *Identity:* f is identity function, if handles q have the same positions as handles p (i.e., $q_i = p_i \Rightarrow f(v) = v$).

Let p be a set of exemplar's facial landmarks and q the set of deformed positions (the target's facial landmarks) of the control points p. For a given point v in the image, we solve the best transformation $l_v(x)$, which minimizes the following sum of weighted squares:

$$\sum_i w_i |l_v(p_i) - q_i|^2,$$

where p_i and q_i are (x, y) coordinates of control and deformed points and the weights w_i have following form:

$$w_i = \frac{1}{|p_i - v|^{2\alpha}}.$$

This is called a *Moving Least Squares* minimization because the weights w_i depend on the evaluating point v. Hence, for each point v, we receive a

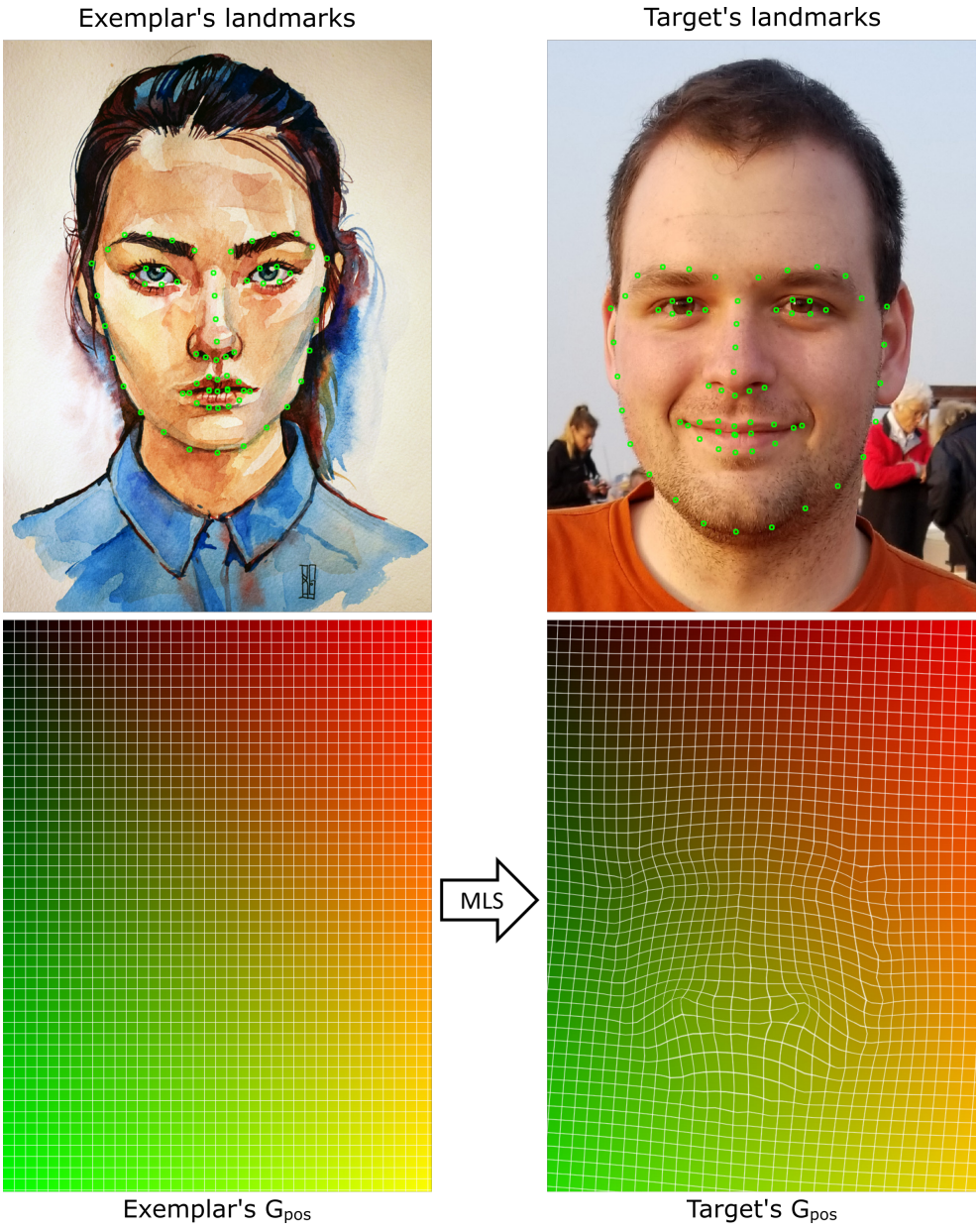

Figure 3.4: *The creation of the target's G_{pos} by warping the style exemplar's G_{pos} using MLS deformation, where detected facial landmarks are used as control points. The white grid is displayed only for visualization purposes.*

different transformation $l_v(x)$. We define the deformation function f as $f(v) = l_v(v)$.

As the transformation $l_v(x)$, we can choose from various types. Probably the most commonly used are these transformations:

- *Affine transformation:* A linear function that preserves points, straight lines, planes, ratios of distances between points on straight line. It is not guaranteed that preserve angles between lines or distances between points. For example, translation, scaling, reflection, rotation and so on.
- *Rigid transformation:* A function preserving the Euclidean distance between points (e.g., rotation, translation, reflection, or their combination).
- *Similarity transformation:* One or more rigid transformations followed by dilatation.

The MLS approach in image deformation has several advantages: warping is fast and smooth, there is no need to triangulate the input image (as in Igarashi et al. 2005 [41]) and by using similarity and rigid transformations, the amount of local scaling and shearing is minimal.

Point-based MLS deformation can be extended from set of control points to set of line segments. The MLS deformation with control lines gives better results and is used in Fišer et al. 2017 [1] approach. However, this extension costs more computational time, so in our approach, we use the original point-based version of MLS deformation. In the Figure 3.4, we can see the result of MLS deformation, that represents positional guides for style exemplar and target photo.

3.3.2 Appearance Guide G_{app}

The appearance guide G_{app} secures the preservation of the identity of the target subject. As the appearance of a human is given especially by wrinkles, freckles, and various blemishes, we do not need color information, but only a detailed structure of the face. So, to generate the guide, we at first convert both input images (style exemplar S and target photo T) to grayscale and then we blur them (see Figure 3.5). Next, we subtract these blurred grayscaled images from their grayscaled originals, thereby we obtain the required detailed structure. By this process, the appearance guide for style exemplar is complete. The G_{app} for the target image has to be further modified because we need to match the target's histogram to exemplar's histogram. The reason why histograms of these appearance guides are supposed to be as similar as possible is simple. We do not want to cause the distortion of the error metric used during the stylization process for searching corresponding chunks in the style exemplar. In the case of unmatched histograms, the error metric may

be too high in situations when chunks in the exemplar and the target image fully correspond to each other. In the Figure 3.5, we can see the result of generating appearance guides, where is the mean and standard deviation of both histograms very similar.

3.4 Example-based style transfer

Once we have generated guidance channels for both input images S and T (see overview in the Figure 3.6), the style transfer is performed. It means that we need to build the required result image from the most suitable coherent areas of pixels (chunks). To create this mosaic of chunks, we go over the target image in scan-line order, and if the current pixel is not stylized yet, we declare it as a *seed* (a pixel the chunk grows from). For obtaining the corresponding seed pixel in style exemplar that has the minimal possible error, we use a *lookup table* and then we let the seed grow.

3.4.1 Lookup Table

This multi-dimensional data structure maps the target's seed to the closest exemplar's seed given by the error metric. We can imagine it as a 3D-cube, that holds in its all positions 2D-coordinates which points to the location in the style exemplar. The time complexity of computing the lookup table is very high because all combinations of each color channel value of exemplar's guides are tested (red and green channel of positional guide and a channel of grayscaled appearance guide) and for each of those combinations, the error metric is computed many times (for each exemplar's pixel). Thus for each style exemplar used in our application, this lookup data structure is precomputed.

3.4.2 Seed Grow

Once we retrieve the exemplar's seed pixel directly determined by the lookup table, we grow the target's seed in all directions while the error is lower than a user-defined threshold. During the growth, only the pixels which are not stylized yet are taken into account. There can be various manners on how to let the seed grow. We tested two following approaches:

1. Besides a user-defined threshold, a square of a certain size restricts the growing area (see Figures 3.7 and 3.8 left).

2. The seed pixel is spread recursively into its left, right, top and bottom neighbor pixels and the growing is limited only by a threshold (see Figures 3.7 and 3.8 right).

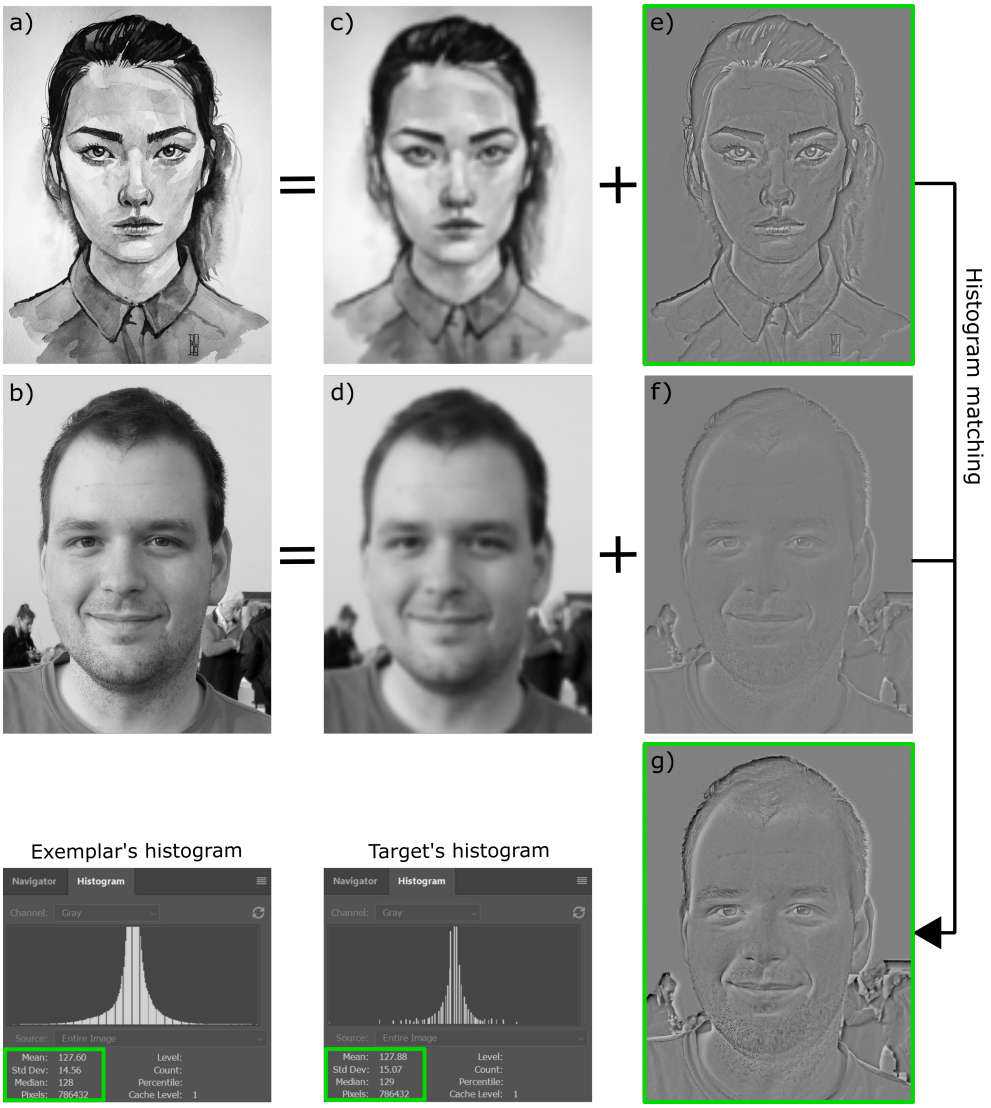

Figure 3.5: *The process of generating of appearance guides G_{app} for the style exemplar and the target image. The original input images are converted to grayscale (a-b), and next, they are blurred (c-d). A detailed structure of these images (e-f) is then obtained by subtraction of blurred images (c-d) from their originals (a-b). Image e) is finished exemplar's appearance guide. Image f) have to be modified to match its histogram to image's e) histogram and by this matching, the target's appearance guide g) is created. On histogram statistics of e) and g) images, we can see the mean and standard deviation to be almost the same.*

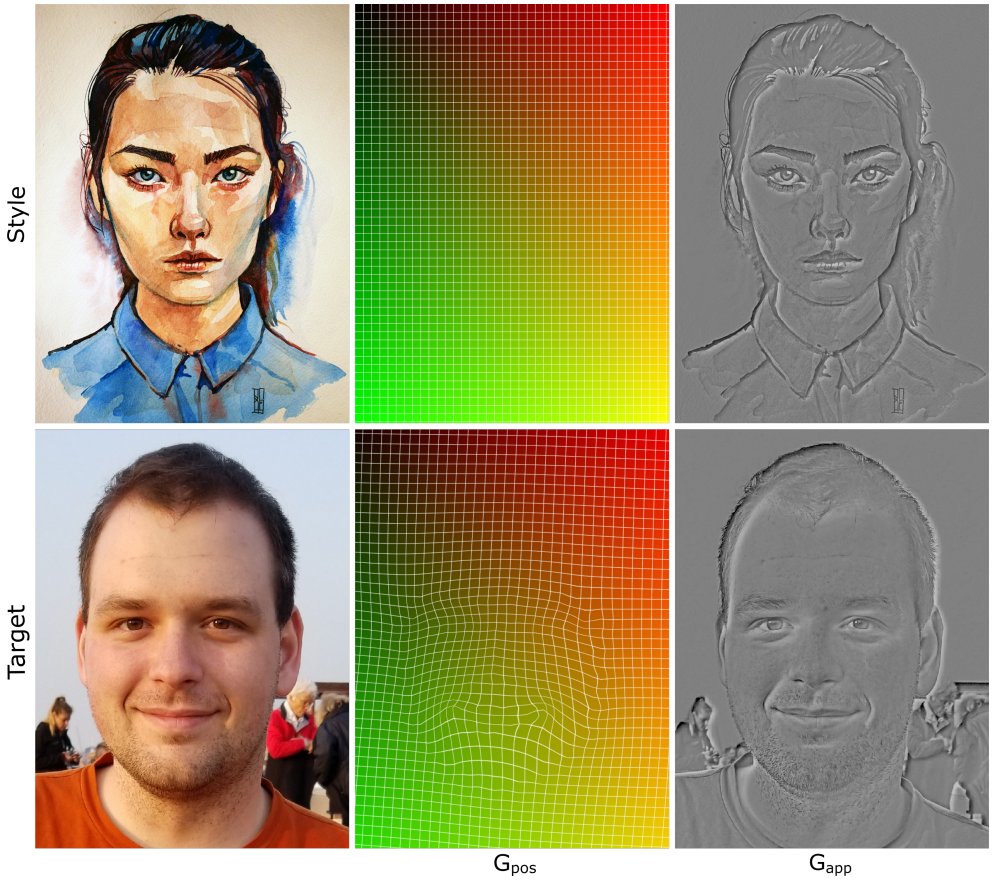

Figure 3.6: *Overview of the guiding channels used in our application. The positional guide G_{pos} secures the style transfer's local consistency. The appearance guide G_{app} encourages the subject's face characteristics.*

Both approaches deliver similar results, but the key advantage of the second method is that seams between individual chunks are barely visible thanks to their irregular shape.

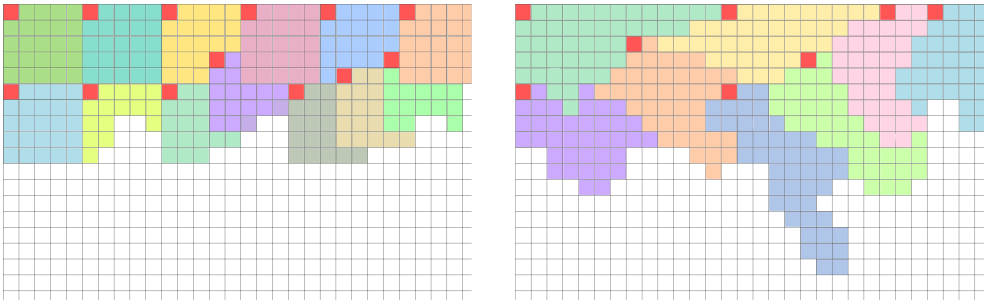

Figure 3.7: *Two methods of the seed pixel's growth. The seed is displayed as the red pixel and individual chunks are differentiated by various colors. In the first case (left), the growing area is restricted by a box of a given size. In the second way (right), the seed grows until the error is lower than a user-defined threshold and chunks are of an irregular shape.*

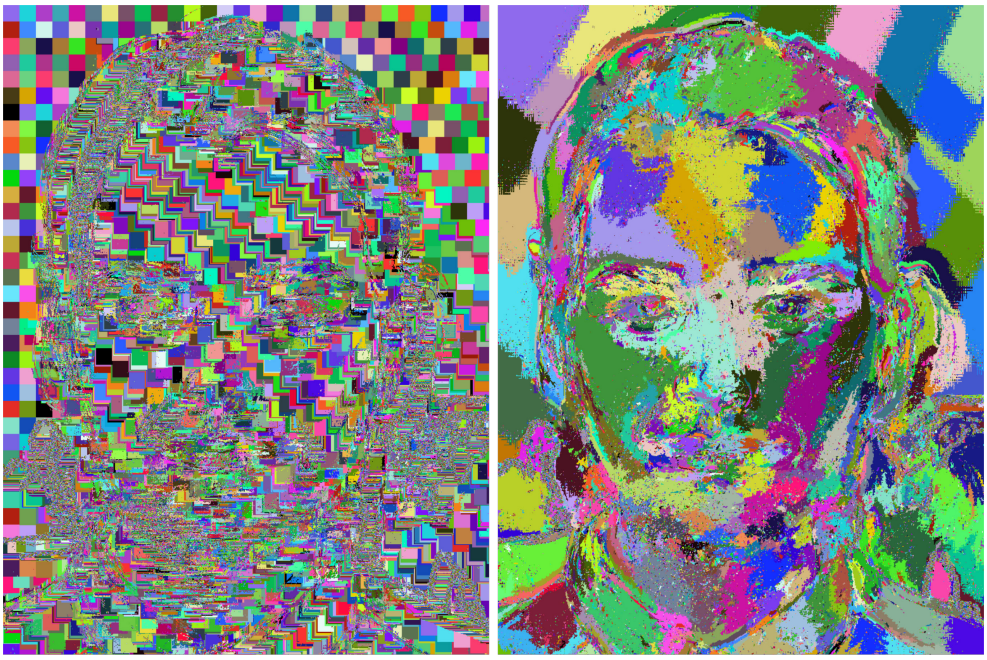

Figure 3.8: *Visualization of the result chunk mosaic when style transfer is performed (the same exemplar and target as in Figure 3.6). Individual chunks are differentiated by various colors. It is obvious that chunks in the left image are restrected by a box and chunks on the right are larger and have irregular shapes.*

Chapter 4

Implementation

In this chapter, we discuss some implementation details of our approach described in Chapter 3. Our application is implemented for devices with Android operating system with SDK Platforms Android 6.0 (Marshmallow) or higher. A significant part of the application is written in Java, but the generating of guidance channels and style transfer itself is implemented in C++. These two parts are connected by the Java Native Interface.

We decided to write the UI in Java because all dialogs are for Android native in this language, and the best possible performance is achieved. As we intend to execute the stylization core on desktop (Windows, Linux) and in the future, create an iOS version of our application, we secure the portability by using the C++ language for this stylization part of our application. Another advantage of C++ is its great performance, which is very important for achieving the interactive response of our mobile application. During the implementation, we tried to enhance the portability by avoiding the involvement of third party libraries that are not maintained well. We used only OpenCV and Dlib libraries that are multiplatform, open-source, and supported by a large community of users.

We designed our application for a single core CPU and still, the response on currently available mobile devices is excellent. Even though the StyleBlit publication (Sýkora et al. 2019 [2]) provides a parallel algorithm for GPU, we did not intend to implement it. The main reason for our decision is that this style transfer algorithm itself is not the bottleneck. The CPU version of StyleBlit is extremely fast, and acceleration on GPU would not give such a benefit for our use case. Most of the computational budget covers the set of steps preceding the StyleBlit algorithm, like facial landmarks detection, generating of guidance channels, and other preprocessing of input data, and final image postprocessing.

4.1 Java part

In Java, we implemented the entire UI of our application. We handle the built-in camera by using the Camera2 API, and via JNI, we call methods from Dlib library, which provides us a face detector, and methods from our C++ style transfer part, which delivers stylization results.

The application is set to portrait orientation, so the display rotation is not possible. For our style transfer algorithm, the target photo T has to be the same size as the exemplar image S. Since we planned to avoid the cropping of target images, the application provides a camera preview in ratio 4:3 which is the same ratio of all predefined style exemplars.

We implemented two versions of displaying results of the style transfer process - *single shot* version and stylized *live camera preview* version. Moreover, the user can choose between two modes - stylization ranges. *Face-only* where only facial region is stylized, and *full-fledged* where the entire target image is stylized.

4.1.1 Stylization Range

In our application, the user can toggle the stylization range - *face-only* and *full-fledged*. These two choices are available in both versions of results displaying, i.e., *single shot* and *live camera preview* versions. In the *full-fledged* stylization range, the style transfer is performed over the entire target image, and the result is returned without additional modifications. In the case of the *face-only* stylization range, the style transfer is applied only on the face region of the target image, and subsequently, the result is alpha-blended into the target's face.

4.1.2 Versions of Results Displaying

We implemented two versions of how the user can see the results of stylization. In the first version, the single target photo T is captured. Next, facial landmarks are displayed, and the user is prompt to confirm that landmarks were detected correctly. After positive confirmation, the style transfer result is displayed immediately. The second version allows observing stylization results directly in live camera preview.

4.1.2.1 Single Shot Version

We created the basic Activity[6] that handles the built-in camera and secures the capturing of the target photo T for style transfer. This Activity provides a button for switching the camera lens direction, thus capturing of photos is possible by front or back camera. Once the user captures a photo, another

[6]By Activity with capital 'A' is meant a crucial component of an Android application.

Activity is launched, and the captured photo is displayed. In this new Activity, the user can perform the following actions:

- Choose from a set of predefined style exemplars.
- Display facial landmarks detected in the captured picture.
- Display result of style transfer applied to the captured photo.
- Toggle the range of stylization (stylize only face or the entire image).

User can also go back to the previous Activity to take another target photo.

4.1.2.2 Live Camera Preview Version

This version allows stylizing the live camera preview directly. Each target frame T from camera preview is processed and results are displayed immediately. In each frame facial landmarks are detected, guidance channels are generated, and finally, the style transfer is performed. In this version, the results of landmarks detection are not displayed; the user sees only the final result of the stylization. The user can toggle between the face-only and full-fledged stylization and choose from a set of predefined style exemplars.

4.2 C++ part

We decided to implement the style transfer core of our application in C++ for performance and portability purposes. We use the OpenCV library which is focused on real-time computer vision, and we have guaranteed that all functions from this library are highly optimized and efficient.

4.2.1 Landmarks detection

We searched for an open-source tool that could provide as many facial points as possible, would be available for the Android platform and would be maintained well at the same time. After detailed research, the Dlib library seemed to be the best option. This library is written in C++ and is connected to the Java part with the help of JNI. We used pre-built *.so* binaries and Java wrapper to access the C++ implementation.

4.2.2 Generating of Guidance Channels

For the semantically meaningful style transfer, it is necessary to generate guidance channels that are tailored to both style and target images. All images (target, exemplar and their guiding channels) that enter into the style transfer algorithm must have exactly the same size.

Each guiding channel has its weight that determines how this guide will influence the error metric and thus the result of the style transfer process. The weight of positional guide we denote as λ_{pos}, and weight of appearance guide as λ_{app}. Values of these weights are based on the observation of Fišer et al. [1]. However, we do not use the full set of guiding channels as is described in the Fišer et al. approach, so we adjust the weights to suit our collection of guiding channels. Our values of weights are $\lambda_{pos} = 10$ and $\lambda_{app} = 2$, and it is important to use the same set of weights in computing the lookup table and in the entire style transfer process.

4.2.2.1 Positional Guide G_{pos}

The source positional guide G^S_{pos} is very easy to implement. It is an image of the same size as the style exemplar that contains all combinations of red and green intensities. Red values are incremented in the x coordinate direction and green ones in y coordinate direction.

The target positional guide G^T_{pos} is created from the exemplar positional guide G^S_{pos} with the help of facial landmarks detected in exemplar and target images. We perform the MLS deformation of the source positional guide based on the given facial landmarks.

4.2.2.2 Appearance Guide G_{app}

The source appearance guide G^S_{app} is also simple for implementation. The color of the style exemplar image is converted to the grayscale and the image is blurred. To obtain the G^S_{app}, we perform the subtraction of each pixel of the grayscaled image and corresponding pixel of the blurred image. Finally, we perform *histogram stretching* to enhance the contrast of the image.

To create the target appearance guide G^T_{app}, we repeat the same steps as in the case of the source appearance guide, except for the histogram stretching. However, the target appearance guide demands additional adjustments. We have to match the G^T_{app} histogram to be as similar as possible to the G^S_{app} histogram.

Histogram matching. For both the style exemplar (reference) image and the target image, we compute their histograms. For each histogram, we calculate the cumulative distribution function F. We denote F_S as CDF of style's histogram and F_T as CDF of target's histogram. Subsequently, for each gray intensity $g_S \in [0, 255]$ is found the gray intensity g_T such that $F_S(g_S) = F_T(g_T)$.

4.2.3 Lookup Table

This multi-dimensional data structure allows to determine exemplar's seed that corresponds to the target's seed with complexity $\mathcal{O}(1)$. Lookup table maps 3D coordinates representing all combinations of color channels of G^S_{pos}

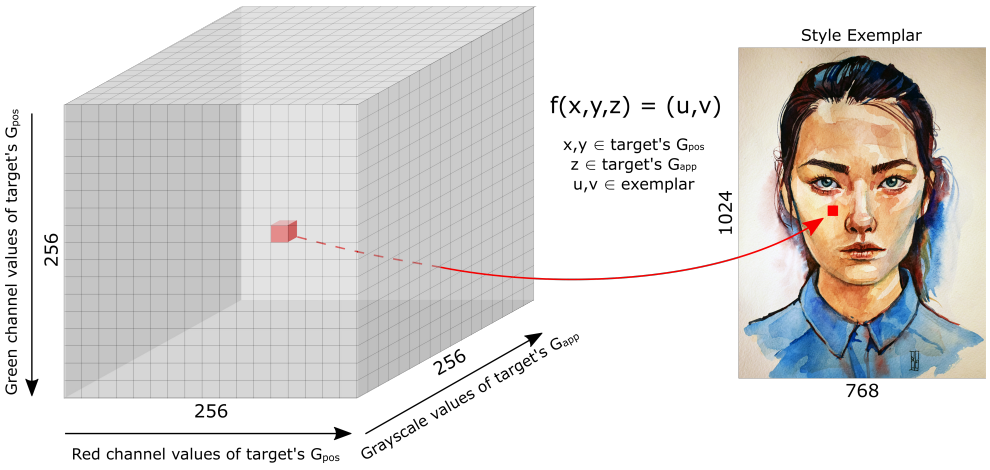

Figure 4.1: *The utilization of the multi-dimensional lookup table to obtain the exemplar's seed pixel. The cube stores coordinates of style exemplar's seed pixels and allows to find the seed with complexity* $\mathcal{O}(1)$ *during the style transfer.*

and G^S_{app} to 2D coordinates representing the seed pixel in style exemplar. Each color channel (red, green) and grayscale have 256 possible intensities, so the size of this cube is $256 \times 256 \times 256$.

To create this cube of the most suitable exemplar's seed pixel coordinates, we need only two inputs: exemplar's positional guide G^S_{pos} and exemplar's appearance guide G^S_{app}. We iterate through all positions in the cube which give us all the possible combinations of red, green and grayscale intensities. For every cube position, the entire style exemplar (its guides respectively) is searched for the closest pixel according to the error metric. The position of the closest pixel is then saved to the current cube location. Weights of individual guiding channels λ_{pos} and λ_{app} are included in the calculation of the error metric (see Algorithm 2).

The computing of the cube has a high complexity ($256 \times 256 \times 256 \times$ *exemplar_width* $\times$ *exemplar_height*), and it takes a lot of computational time. Since the cube is independent on the target image, it can be precomputed and serialized into a file for each style exemplar. When the user chooses required exemplar from a set, the cube is loaded from the file and used as input to style transfer.

To use the lookup table for obtaining the exemplar's seed, we determine the position in the cube by the target's guides G^T_{pos} and G^T_{app} which gives us the seed in the style exemplar with complexity $\mathcal{O}(1)$ (see Figure 4.1).

Algorithm 2: Lookup Table

Inputs : source positional guide G^S_{pos}, source appearance guide G^S_{app}.
Output: lookup table *lut*.

```
LookupTable():
    for each red intensity x ∈ [0, 255] do
        for each green intensity y ∈ [0, 255] do
            for each gray intensity z ∈ [0, 255] do
                for each row v ∈ [0, exemplar_height − 1] do
                    for each column u ∈ [0, exemplar_width − 1] do
                        e = λ_pos * (||G^{S,red}_pos[v][u] − x|| + ||G^{S,green}_pos[v][u] −
                          y||) + λ_app * ||G^S_app[v][u] − z||
                        if e < e_min then
                            lut[x][y][z] = (u, v)
```

4.2.4 StyleBlit

Since we have generated exemplar's and target's guidance channels and lookup table, we can perform the style transfer algorithm. All these data together with the style image and user-defined threshold are used as input parameters to the stylization method. The target photo and the style exemplar have the same size and all their guidance channels as well. During the style transfer implementation, we followed the basic brute-force StyleBlit algorithm (see Algorithm 1). In this section, we provide a description of the algorithm and pseudocode in more detail.

We iterate through the target image pixel by pixel in scan-line order, and each uncovered pixel is declared as a seed, for which we obtain via lookup table the corresponding exemplar's seed with complexity $\mathcal{O}(1)$ (see Algorithm 3). Once we have the seed pixel, we use one of the two approaches that secure the growth of the area around the seed and formation of the chunk. We provide pseudocodes of both types of seed growth - methods `BoxSeedGrowth()` in Algorithm 4 and `FreeShapeSeedGrowth()` in Algorithm 5. The result chunk is then placed into the target image.

4.2.4.1 Box Seed Growth

In this way of the seed pixel growth, the chunk is restricted by a *box* of user-defined size. For each offset point o in this *box*, we first check whether the pixel on position $seed_T + o$ in the target image is covered (stylized). If not, we

Algorithm 3: Our StyleBlit

Inputs : source style exemplar C_S, source positional guide G^S_{pos}, source appearance guide G^S_{app}, target positional guide G^T_{pos}, target appearance guide G^T_{app}.

Output: target stylized image C_T.

```
OurStyleBlit():
    for each pixel p ∈ C_T do
        if C_T[p] is empty then
            seed = lookupTable[G^{T,red}_pos[p]][G^{T,green}_pos[p]][G^T_app[p]]
            chunk = BoxSeedGrowth() or FreeShapeSeedGrowth()
            insert chunk to C_T
```

compute the error metric e for pixels on position $seed_S + o$ in the exemplar's guides G^S_{pos} and G^S_{app} and on position $seed_T + o$ in the target's guides G^T_{pos} and G^T_{app}. If the error e is lower than the user-defined threshold t, the current pixel belongs to the coherent *chunk* and thus the corresponding pixel from the style exemplar C_S on the position $seed_S + o$ is copied to the *chunk* on the position $seed_T + o$. The output of this method is stylized coherent *chunk* (see Algorithm 4).

Algorithm 4: Box Seed Growth

Inputs : source seed $seed_S$, target seed $seed_T$, source style exemplar C_S, source positional guide G^S_{pos}, source appearance guide G^S_{app}, target positional guide G^T_{pos}, target appearance guide G^T_{app}, threshold t.

Output: coherent area of pixels *chunk*.

```
BoxSeedGrowth():
    for each offset point o ∈ box do
        if coveredPixels[seed_T + o] is empty then
            e = λ_pos * (||G^{S,red}_pos[seed_S + o] − G^{T,red}_pos[seed_T + o]|| +
              ||G^{S,green}_pos[seed_S + o] − G^{T,green}_pos[seed_T + o]||) + λ_app *
              ||G^S_app[seed_S + o] − G^T_app[seed_T + o]||
            if e < t then
                chunk[seed_T + o] = C_S[seed_S + o]
```

4.2.4.2 Free Shape Seed Growth

Free shape manner of the seed pixel growth is not limited by a box of a given size. The seed pixel is growing to all directions until the error is lower than the user-defined threshold. In the beginning, we have a point with $(0, 0)$ coordinates (the seed pixel with no offset) in the *offsetQueue*. While the *offsetQueue* is not empty, we take a front offset point o from this queue, and if the pixel on position $seed_T + o$ in the target image is not yet covered, we compute the error metric e in the same way as in the `BoxSeedGrowth()` method. If the error e is lower than the user-defined threshold t, the pixel from the style exemplar C_S on the position $seed_S + o$ is transferred to the *chunk* on the position $seed_T + o$. Subsequently, we insert left, right, up and down neighbor of offset point o into the *offsetQueue*. The output of this method is also coherent *chunk* of stylized pixels (see Algorithm 5).

Algorithm 5: Free Shape Seed Growth

Inputs : source seed $seed_S$, target seed $seed_T$, source style exemplar C_S, source positional guide G^S_{pos}, source appearance guide G^S_{app}, target positional guide G^T_{pos}, target appearance guide G^T_{app}, threshold t.

Output: coherent area of pixels *chunk*.

```
FreeShapeSeedGrowth():
    while offsetQueue is not empty do
        o = front offset point from offsetQueue
        if coveredPixels[seed_T + o] is empty then
            e = λ_pos * (||G^{S,red}_pos[seed_S + o] − G^{T,red}_pos[seed_T + o]|| +
              ||G^{S,green}_pos[seed_S + o] − G^{T,green}_pos[seed_T + o]||) + λ_app *
              ||G^S_app[seed_S + o] − G^T_app[seed_T + o]||
            if e < t then
                chunk[seed_T + o] = C_S[seed_S + o]
                insert left neighbor of o into offsetQueue
                insert right neighbor of o into offsetQueue
                insert up neighbor of o into offsetQueue
                insert down neighbor of o into offsetQueue
```

4.3 Java Native Interface

Java Native Interface is a framework that enables two-way connection of Java code running in Java Virtual Machine and native programs and libraries that are written in different programming languages. JNI connects the Java and C++ part of our application and secures the transmission of data between those parts. In Java, we can call methods that perform the landmarks detection and the style transfer process, and the results of executed C++ code are delivered to the Java part.

4.3.1 JNI and Landmarks Detection

We work with *Dlib* implementation of the face detector via Java wrapper. Before we provide a Java `Bitmap` object to the face detector, we subsample it to speed up the detection. The reasonable subsampling affects the accuracy of detector negligibly and makes the detection significantly faster. The *Dlib* detector returns a position of the rectangle surrounding the detected face and an array of points representing the landmarks found in the face.

4.3.2 JNI and Style Transfer Method

Before we call the stylization method, we prepare all the parameters of this method for transmission. Some of them we serialize (`Bitmap` type in Java or `Mat` type from C++) and then, all non-primitive Java data types are converted to C++ data types. Into the C++ part, we transfer following data:

- *Target image* captured by a built-in camera (in Java part, this image is represented as a `Bitmap` type, so before transmission, it is serialized to an array of `byte`s and then casted to an array of `unsigned char`s).
- *Style exemplar* from a set of predefined exemplars (also `Bitmap` type, it is transformed in the same way as the target image).
- *Lookup table* (in Java part, this structure is represented as an array of `byte`s, so it is casted to an array of `unsigned char`s).
- *Target's facial landmarks coordinates* (in Java part, landmarks are stored in `string` data type, so this parameter is converted to an array of `char`s).
- *Exemplar's facial landmarks coordinates* (same as target's landmarks).
- *Width* and *height* of target/exemplar image (primitive data types that are not converted).

The stylization method takes all the parameters described above and returns the stylized target image as an array of `unsigned char`s. This data structure is converted to an array of `byte`s to correspond to the Java representation of serialized images.

Chapter 5

Results and Measurements

In this chapter, we show some outputs of our method and time measurements of the entire process and individual steps leading up to a result of our application. Also, some experiments with parameters of style transfer are discussed.

5.1 Results

All results (Figures 5.2 and 5.3) provided in this section were generated by applying our style transfer algorithm on subjects in the Figure 5.1 with using all guidance channels and following synthesis parameters:

- Weights of guidance channels: $\lambda_{pos} = 10$, $\lambda_{app} = 2$.
- Threshold: $t = 50$ (the `FreeShapeSeedGrowth()` method was used).

We display both stylization ranges - *face-only* and *full-fledged*. In the *full-fledged* case, the "ghosting" caused by insufficient guidance of style transfer in the regions of hair and background is apparent. There is no better way how to handle this issue without the image segmentation. We intend to add the segmentation guiding channel in the future to get better results in a *full-fledged* stylization range.

For more results, see Chapter A in Appendix.

5.1.1 Screenshots of the application

In Figures 5.4 and 5.5 screenshots of the live camera preview are displayed.

5.2 Experiments

We performed some experiments with parameters of the style transfer process. In this section, we demonstrate how important it is to use guidance channels and other operations in order to deliver pleasing results. Although we can

Figure 5.1: *Target photos - male and female subject.*

speed up the style transfer process by omitting some subtasks, the quality of stylization result decreases significantly. We consider all the individual subtasks we use to solve the style transfer problem as a necessary minimum to get faithfully looking results.

The absence of the positional guide G_{pos} may cause that coherent chunks of style exemplar are transferred at wrong locations, e.g., chunks from exemplar's lips can appear on the chin or cheeks of the resultant portrait (see the second column in Figure 5.6).

Without the appearance guide G_{pos}, the subject's identity is not preserved, e.g., different eyebrows, the absence of wrinkles, and the wrong shape of the target's nose (see the third column in Figure 5.6).

As another experiment, we tested the necessity of the histogram matching when we generate the target's appearance guide G^T_{app}. Without matching the appearance guides' histograms, the error e overcomes the threshold t too soon, which leads to smaller chunks and the result may seem blurry (see Figure 5.7).

5.3 Measurements

We measured the total computational time and also the time of all individual subtasks needed to be solved to deliver faithfully looking final result (see Table 5.1). The measurement was performed in Release build variant on *Samsung Galaxy Note8* with CPU *Samsung Exynos 8895, 2.3 GHz*, GPU *Mali G71 MP20* and *6 GB* of RAM.

In Table 5.1, we do not mention operations with very low time cost and also such subtasks that are precomputed or loaded from cache (e.g., precomputed lookup table, JNI allocations, cached exemplar's positional and appearance guides, cached exemplar's landmarks and style exemplar image). As we can

Figure 5.2: *Results from our application. Target: male subject from Figure 5.1. Style exemplars are in the first column, results of face-only stylization are in the second column, and full-fledged results are in the last column.*

Figure 5.3: *Results from our application. Target: female subject from Figure 5.1 Style exemplars are in the first column, results of face-only stylization are in the second column, and full-fledged results are in the last column.*

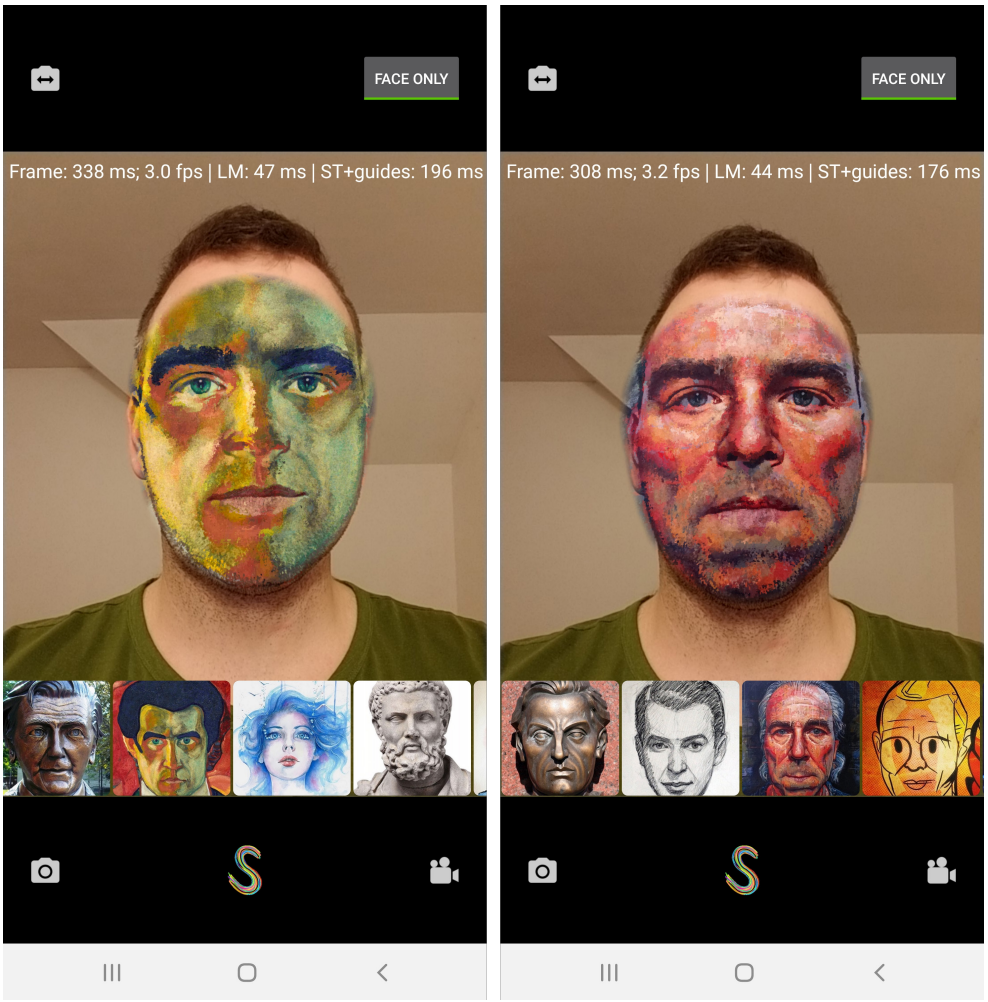

Figure 5.4: *Screenshots of our application (live camera preview, face-only mode).*

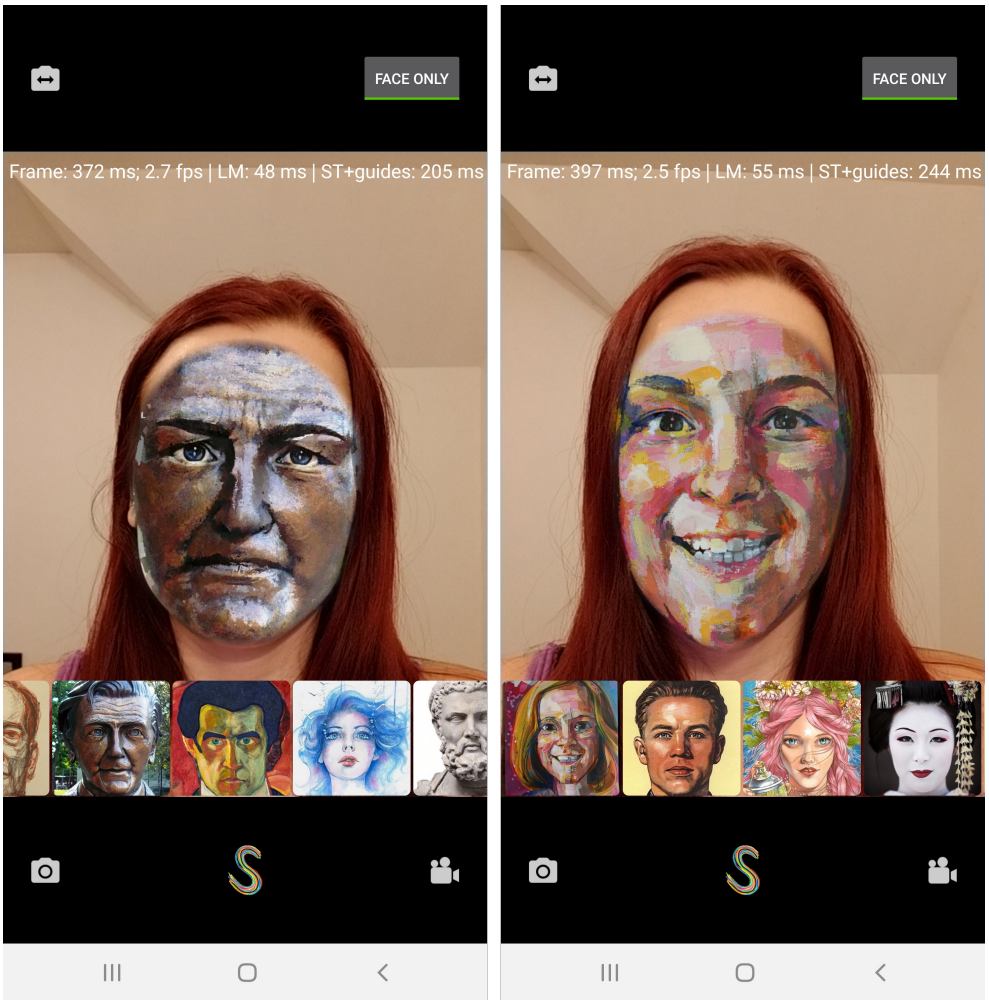

Figure 5.5: *Screenshots of our application (live camera preview, face-only mode).*

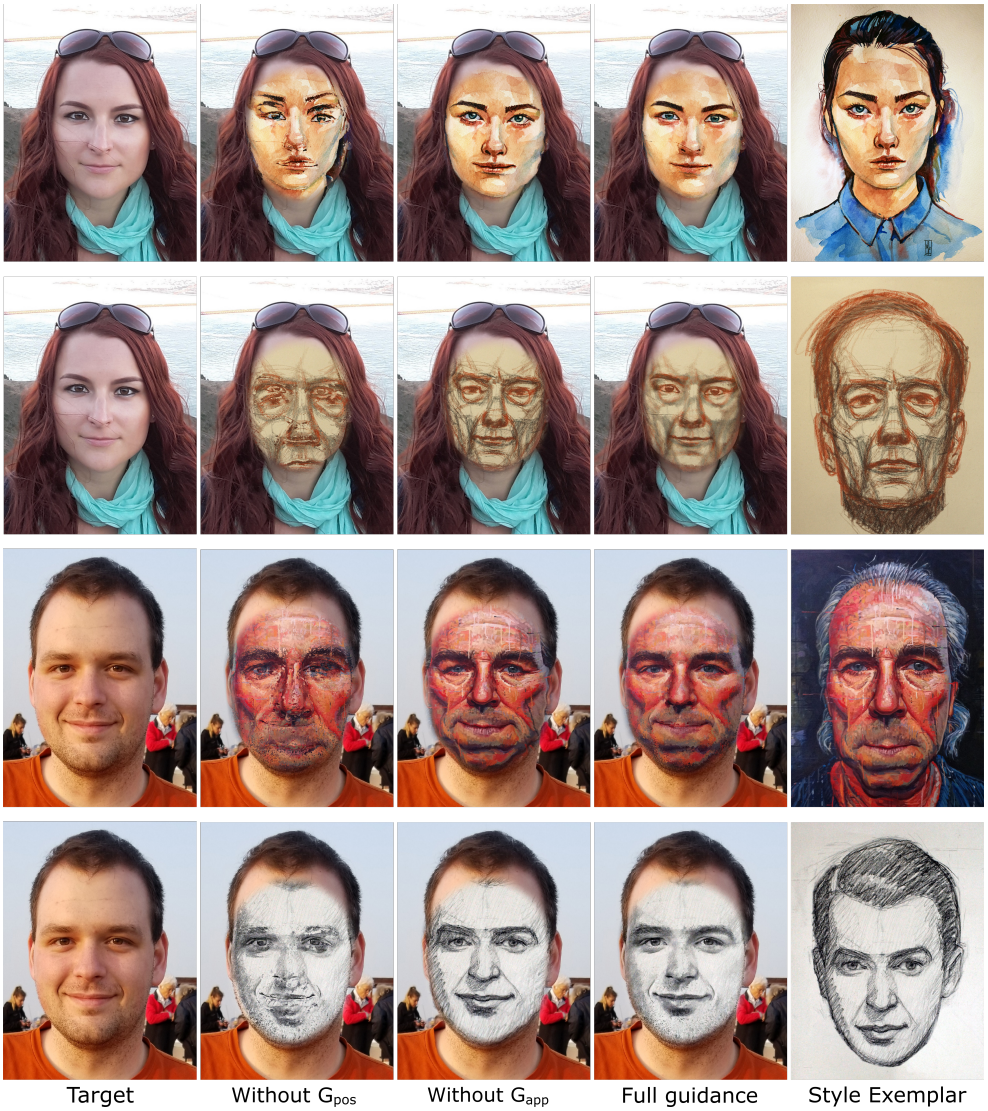

Figure 5.6: *Importance of individual guidance channels. We can see that the use of the positional guide G_{pos} is essential and its absence causes that style transfer gives nonsensical results. The appearance guide G_{app} is also necessary when we want to preserve the target's identity.*

Figure 5.7: *Importance of histogram matching during the generating the target's appearance guide* G_{app}^{T}. *Without the histogram matching, the subject's identity is not preserved well and the result seems blurry.*

Table 5.1: *Average time measurements.*

Subtask	**Time**
Facial landmarks detection	43 ms
Generating of G^T_{pos} (MLS deformation)	54 ms
Generating of G^T_{app} (grayscale converting, blurring, subtracting, histogram matching)	10 ms
StyleBlit (face-only)	88 ms
StyleBlit (full-fledged)	120 ms
Alpha-blend (after the face-only style transfer)	28 ms
Additional overhead (camera handling, frame preprocessing, result drawing)	100 ms
TOTAL	323 ms / 3 FPS

see in Table 5.1, the time cost of the landmarks detection, generating target's guiding channels and additional overhead is two times higher than the time cost of StyleBlit algorithm. Overall, we achieved the real-time (3 FPS) face stylization on currently available mobile devices.

Chapter 6

Limitations and Future Work

6.1 Limitations of our approach

Our method presumes that the stylized subject is facing the built-in camera mostly frontally. This presumption is caused by utilization of ERT based face detector for obtaining the facial landmarks of the subject. This approach does not support odd angles of the subject's face. Therefore, for these unusual positions of the head, the landmarks detection fails, and thus the style transfer result cannot be provided.

Another limitation of this style transfer approach is the inability to precisely reproduce textures that are not stochastic but somehow regular or that have longer linear structures. It causes that result image is composed from visibly misaligned individual chunks.

Also, there could appear excessive repetitions when the scale of the target subject's head is distinctly larger than the portrait in the style exemplar.

6.2 Future work

We intend to continue working on this project. We plan to add several new features, like the detection of hair, skin, and background to achieve better-looking results. The current solution delivers pleasing results, but the utilization of the image segmentation (identification of different segments of the photo) is very important for the overall impression of the stylized result. Especially, the correct hair stylization is essential for the better preservation of the subject's identity. The hair/skin/background segmentation secures that coherent chunks of the exemplar will be copied to subject's hair/skin/background region exclusively from exemplar's hair/skin/background area. Admittedly, the integration of other tools for detection will be difficult, particularly for performance reasons. We will have to come up with some additional optimizations to keep the computational time low while improving style transfer results.

Last but not least, we plan to provide the user with the option to add new user-defined style exemplars and record a stylized video.

Conclusion

In accordance with the assignment of this thesis, we created an Android application that automatically transfers an artistic or a photorealistic style from a single style exemplar portrait to a target head in motion or in still positions with preservation of the subject's identity. Target photographs or live camera preview for stylization are obtained by the built-in camera of a given mobile device. The style transfer is implemented with the highly efficient StyleBlit [2] algorithm, and guidance channels supporting the semantically meaningful synthesis are generated according to the FaceStyle [1] approach. According to assignment, the application should achieve at least interactive response on currently available mobile devices. We managed to reach the real-time stylization with no help of online computing resources, even though, for the sake of the portability and simplicity, we designed our application for a single core CPU.

Bibliography

[1] Fišer, J.; Jamriška, O.; et al. Example-Based Synthesis of Stylized Facial Animations. *ACM Transactions on Graphics*, volume 36, no. 4, 2017: p. 155.

[2] Sýkora, D.; Jamriška, O.; et al. StyleBlit: Fast Example-Based Stylization with Local Guidance. *Eurographics*, 2019.

[3] DiPaola, S. Painterly rendered portraits from photographs using a knowledge-based approach. In *Proceedings of SPIE Human Vision and Electronic Imaging*, volume 6492, 2007, pp. 33–43.

[4] Gooch, B.; Reinhard, E.; et al. Human Facial Illustrations: Creation and Psychophysical Evaluation. *ACM Transactions on Graphics*, volume 23, no. 1, 2004: pp. 27–44.

[5] Tresset, P.; Leymarie, F. F. Generative Portrait Sketching. In *Proceedings of International Conference on Virtual Systems and Multimedia*, 2005, pp. 739–748.

[6] Yang, M.; Lin, S.; et al. Semantics-driven portrait cartoon stylization. In *Proceedings of International Conference on Image Processing*, 2010, pp. 1805–1808.

[7] Chen, H.; Zheng, N.; et al. PicToon: A Personalized Image-Based Cartoon System. In *Proceedings of ACM International Conference on Multimedia*, 2002, pp. 171–178.

[8] Chen, H.; Liu, Z.; et al. Example-Based Composite Sketching of Human Portraits. In *Proceedings of International Symposium on Non-Photorealistic Animation and Rendering*, 2004, pp. 95–102.

[9] Chen, H.; Liang, L.; et al. Example-Based Automatic Portraiture. In *Proceedings of Asian Conference on Computer Vision*, 2002, pp. 171–178.

[10] Meng, M.; Zhao, M.; et al. Artistic paper-cut of human portraits. In *Proceedings of ACM Multimedia*, 2010, pp. 931–934.

[11] Zhang, Y.; Dong, W.; et al. Data-driven Face Cartoon Stylization. In *SIGGRAPH Asia Technical Briefs*, 2014, p. 14.

[12] Li, H.; Liu, G.; et al. Guided Face Cartoon Synthesis. *IEEE Transactions on Multimedia*, volume 13, no. 6, 2011: pp. 1230–1239.

[13] Wang, N.; Tao, D.; et al. Transductive Face Sketch-Photo Synthesis. *IEEE Transactions on Neural Networks and Learning Systems*, volume 24, no. 9, 2013: pp. 1364–1376.

[14] Wang, N.; Tao, D.; et al. A Comprehensive Survey to Face Hallucination. *International Journal of Computer Vision*, volume 106, no. 1, 2014: pp. 9–30.

[15] Wang, X.; Tang, X. Face Photo-Sketch Synthesis and Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, volume 31, no. 11, 2009: pp. 1955–1967.

[16] Zhou, H.; Kuang, Z.; et al. Markov Weight Fields for face sketch synthesis. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2012, pp. 1091–1097.

[17] Berger, I.; Shamir, A.; et al. Style and abstraction in portrait sketching. *ACM Transactions on Graphics*, volume 32, no. 4, 2013: p. 55.

[18] Wang, T.; Collomosse, J. P.; et al. Learnable Stroke Models for Example-based Portrait Painting. In *Proceedings of British Machine Vision Conference*, 2013.

[19] Zhao, M.; Zhu, S.-C. Portrait Painting Using Active Templates. In *Proceedings of International Symposium on Non-Photorealistic Animation and Rendering*, 2011, pp. 117–124.

[20] Gatys, L. A.; Ecker, A. S.; et al. Image Style Transfer Using Convolutional Neural Networks. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 2414–2423.

[21] Simonyan, K.; Zisserman, A. Very Deep Convolutional Networks for Large-Scale Image Recognition. *CoRR*, volume abs/1409.1556, 2014.

[22] Portilla, J.; Simoncelli, E. P. A Parametric Texture Model Based on Joint Statistics of Complex Wavelet Coefficients. *International Journal of Computer Vision*, volume 40, no. 1, 2000: pp. 49–70.

[23] Johnson, J.; Alahi, A.; et al. Perceptual Losses for Real-Time Style Transfer and Super-Resolution. In *Proceedings of European Conference on Computer Vision*, 2016, pp. 694–711.

[24] Ruder, M.; Dosovitskiy, A.; et al. Artistic Style Transfer for Videos. In *Proceedings of German Conference Pattern Recognition*, 2016, pp. 26–36.

[25] Selim, A.; Elgharib, M.; et al. Painting Style Transfer for Head Portraits Using Convolutional Neural Networks. *ACM Transactions on Graphics*, volume 35, no. 4, 2016: p. 129.

[26] Fišer, J.; Jamriška, O.; et al. StyLit: Illumination-Guided Example-Based Stylization of 3D Renderings. *ACM Transactions on Graphics*, volume 35, no. 4, 2016: p. 92.

[27] Kemelmacher-Shlizerman, I. Transfiguring Portraits. *ACM Transactions on Graphics*, volume 35, no. 4, 2016: p. 94.

[28] Shen, X.; Hertzmann, A.; et al. Automatic Portrait Segmentation for Image Stylization. *Computer Graphics Forum*, volume 35, no. 2, 2016: pp. 93–102.

[29] Shih, Y.-C.; Paris, S.; et al. Style Transfer for Headshot Portraits. *ACM Transactions on Graphics*, volume 33, no. 4, 2014: p. 148.

[30] Yang, Y.; Zhao, H.; et al. Semantic portrait color transfer with internet images. *Multimedia Tools and Applications*, 2015: pp. 1–19.

[31] Kwatra, V.; Essa, I. A.; et al. Texture optimization for example-based synthesis. *ACM Transactions on Graphics*, volume 24, no. 3, 2005: pp. 795–802.

[32] Wexler, Y.; Shechtman, E.; et al. Space-Time Completion of Video. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, volume 29, no. 3, 2007: pp. 463–476.

[33] Levin, A.; Lischinski, D.; et al. A Closed-Form Solution to Natural Image Matting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, volume 30, no. 2, 2008: pp. 228–242.

[34] Gong, Y.; Sakauchi, M. Detection of Regions Matching Specified Chromatic Features. *Computer Vision and Image Understanding*, volume 61, no. 2, 1995: pp. 263–269.

[35] Orzan, A.; Bousseau, A.; et al. Diffusion Curves: A Vector Representation for Smooth-Shaded Images. *ACM Transactions on Graphics*, volume 27, no. 3, 2008: p. 92.

[36] Schaefer, S.; McPhail, T.; et al. Image Deformation Using Moving Least Squares. *ACM Transactions on Graphics*, volume 25, no. 3, 2006: pp. 533–540.

[37] Fišer, J.; Lukáč, M.; et al. Color Me Noisy: Example-based Rendering of Hand-colored Animations with Temporal Noise Control. *Computer Graphics Forum*, volume 33, no. 4, 2014: pp. 1–10.

[38] Newson, A.; Almansa, A.; et al. Video Inpainting of Complex Scenes. *SIAM Journal of Imaging Science*, volume 7, no. 4, 2014: pp. 1993–2019.

[39] King, D. E. Dlib-ml: A Machine Learning Toolkit. *Journal of Machine Learning Research*, volume 10, 2009: pp. 1755–1758.

[40] Kazemi, V.; Sullivan, J. One Millisecond Face Alignment with an Ensemble of Regression Trees. *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2014.

[41] Igarashi, T.; Moscovich, T.; et al. As-rigid-as-possible shape manipulation. *ACM Transactions on Graphics*, 2005: pp. 1134–1141.

Appendix A

Additional Results

In this part of appendix, we provide more results of our implementation of example-based style transfer that was applied on the subjects from Figure 5.1.

Figure A.1: *Additional results. Target: male subject from Figure 5.1. Style exemplars are in the first column, results of face-only stylization are in the second column and full-fledged results are in the last column.*

Figure A.2: *Additional results. Target: male subject from Figure 5.1. Style exemplars are in the first column, results of face-only stylization are in the second column and full-fledged results are in the last column.*

Figure A.3: *Additional results. Target: male subject from Figure 5.1. Style exemplars are in the first column, results of face-only stylization are in the second column and full-fledged results are in the last column.*

Figure A.4: *Additional results. Target: male subject from Figure 5.1. Style exemplars are in the first column, results of face-only stylization are in the second column and full-fledged results are in the last column.*

Figure A.5: *Additional results. Target: female subject from Figure 5.1. Style exemplars are in the first column, results of face-only stylization are in the second column and full-fledged results are in the last column.*

Figure A.6: *Additional results. Target: female subject from Figure 5.1. Style exemplars are in the first column, results of face-only stylization are in the second column and full-fledged results are in the last column.*

Figure A.7: *Additional results. Target: female subject from Figure 5.1. Style exemplars are in the first column, results of face-only stylization are in the second column and full-fledged results are in the last column.*

Figure A.8: *Additional results. Target: female subject from Figure 5.1. Style exemplars are in the first column, results of face-only stylization are in the second column and full-fledged results are in the last column.*

Appendix B

Acronyms

API Application Programming Interface

CDF Cumulative Distribution Function

CNN Convolutional Neural Network

CPU Central Processing Unit

ERT Ensemble of Regression Trees

FPS Frames Per Second

GPU Graphics Processing Unit

JNI Java Native Interface

MLS Moving least squares

RAM Random Access Memory

SDK Software Development Kit

UHD Ultra High Definition

UI User Interface

VGG Visual Geometry Group (a pre-trained CNN for object recognition)

Appendix C

Contents of enclosed USB Flash Drive

- `readme.txt` ..... the file with USB Flash Drive contents description
- `src` ..... the directory of source codes
  - `impl` ..... implementation sources
  - `thesis` ..... the directory of LaTeX source codes of the thesis
- `text` ..... the thesis text directory
  - `DP_Moravcova_Aneta_2019.pdf` ..... the thesis text in PDF format
  - `assignment.pdf` ..... the thesis assignment in PDF format