Czech technical university in Prague
Faculty of Electrical Engineering

Department of Computer Graphics and Interaction

Program: Computer Graphics

Example-based Stylization of Headshot portraits on the GPU

Stylizace potrétu na GPU s využitím výtvarné předlohy

MASTER'S THESIS

Author:	Bc. Marek Nechanský
Supervisor:	prof. Ing. Daniel Sykora, Ph.D.
Academic year:	2020/2021

ČVUT
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Nechanský** Jméno: **Marek** Osobní číslo: **460660**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačové grafiky a interakce**

Studijní program: **Otevřená informatika**

Specializace: **Počítačová grafika**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Stylizace portrétu na GPU s využitím výtvarné předlohy

Název diplomové práce anglicky:

Example-based Stylization of Headshot Portraits on the GPU

Pokyny pro vypracování:

Seznamte se s technikami pro přenos výtvarného stylu z předlohy na portrét [1, 2, 3]. Impelmentuje algoritmus FaceStyle [2], jenž k syntéze využívá přístup založený na použití záplat [4, 5]. Implementaci proveďte v jazyce CUDA tak, aby bylo možné výpočet paralelizovat na GPU. Zvolte vhodnou varianty algoritmu, která bude pro paralelní zpracování výhodnější. Výslednou implementaci následně ověřte v aplikačním scénáři generování trénovací množiny dat pro algoritmus [3]. Sadu vstupních dat pro tento experiment dodá vedoucí práce.

Seznam doporučené literatury:

[1] Selim et al.: Painting Style Transfer for Head Portraits Using Convolutional Neural Networks, ACM Transactions on Graphics 35(4):129. 2016.
[2] Fišer et al.: Example-Based Synthesis of Stylized Facial Animations, ACM Transactions on Graphics 36(4):155, 2017.
[3] Futschik et al.: Real-Time Patch-Based Stylization of Portraits Using Generative Adversarial Network, Proceedings of the 8th ACM/EG Expressive Symposium, pp. 33-42, 2019.
[4] Fišer et al.: StyLit: illumination-guided example-based stylization of 3D renderings, ACM Transactions on Graphics 35(4):92, 2016.
[5] Kaspar et al.: Self Tuning Texture Optimization, Computer Graphics Forum 34(2):349-359, 2015.

Jméno a pracoviště vedoucí(ho) diplomové práce:

prof. Ing. Daniel Sýkora, Ph.D., Katedra počítačové grafiky a interakce

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **07.01.2021** Termín odevzdání diplomové práce: ____________

Platnost zadání diplomové práce: **30.09.2022**

[illegible] [illegible] [illegible]

prof. Ing. Daniel Sýkora, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

. Datum převzetí zadání	Podpis studenta

Declaration

Acknowledgements

I would like to thank prof. Ing. Daniel Sýkorka, Ph.D. for his expert guidance.

Marek Nechanský

Název práce:

Stylizace potrétu na GPU s využitím výtvarné předlohy

Autor: Bc. Marek Nechanský

Studijní program: Otevřená informatika

Obor: Počítačová grafika

Druh práce: Diplomová práce

Vedoucí práce: prof. Ing. Daniel Sykora, Ph.D.
České vysoké učení technické v Praze, Fakulta elektrotechnická

Abstrakt: Tato práce se zabývá studiem technik stylizace obrazu, které umožňují replikovat umělecký styl a použít jej na vlastní obraz. Velmi bohatá větev tohoto zaměření je přenos stylu portrétů z jednoho obrazu na druhý. Cílem této metody je ponechat fyzický vzhled vyzobrazené osoby a při tom napodobit styl potrétu osoby druhé. Hlavní cíl této diplomové práce je realizace jedné z těchto metod a produkce stylizovaných portrétů.

Klíčová slova: dle předlohy, přenos uměleckého stylu, syntéza textury

Title:

Example-based Stylization of Headshot portraits on the GPU

Author: Bc. Marek Nechanský

Abstract: This thesis deals with techniques of style transfer, which allow us to transfer a specific artistic style onto our chosen image. A very important branch of these methods is portrait stylization, specifically one where artistic style of one portrait is applied to another. The target of this method is to keep the person's appearance while using the style of the other picture. The main goal of this thesis is the realization of one of these methods and the production of stylized images.

Key words: example-based, style transfer, texture synthesis

Contents

List of Figures

Introduction

Image manipulation is an important part of computer graphics, still very popular in recent years. One example of modern image manipulation is style transfer. The style transfer algorithm applies the chosen artistic style to an image. With this technology, we can replicate the artistic styles of painters throughout history or reduce the amount of work an artist has to do when creating a stylized video sequence. This thesis aims to explore all commonly used style transfer methods and implement the creation of guide channels for example-based portrait style transfer algorithm used by Fišer et al. [1]. Style transfer algorithms use two sets of images, where the first set is style examples, and those in the second set are original images. Style transfer methods use those two sets of images to create a third set, which take the style of images in the first set and apply it to the second set. Example of a style transfer algorithm output can be seen in Figure 1. Original images are usually very common images. It can be an image taken by the algorithm user, or we might want to do a style transfer on an image from a movie. On the other side, style examples are very uncommon, and quite often they are paintings created by a famous painter. In this thesis, we focus on portrait style transfer. An example of this transfer can be found in Figure 2. Even though the variety of images is much smaller, human eyes are very sensitive to artifacts in a face image. Therefore, portrait style transfer algorithms have to focus on small details and have to be different from those used for style transfer of nature or industrial sites.

Style transfer can be used to apply a certain style to a video sequence and thus create a non-realistic video, which does not have to be fully computer created by standard techniques and can be transferred from a video captured with a camera. Very frequent usage of portrait style transfers is in the photo industry, where users can use predefined painting styles on their photos or even use style transfer on a live feed from a video camera to see the result in real-time.

The inputs of our algorithm are the two images, which we use in transfer, their head masks, skin masks and face landmarks, which can be seen in Figure 3.

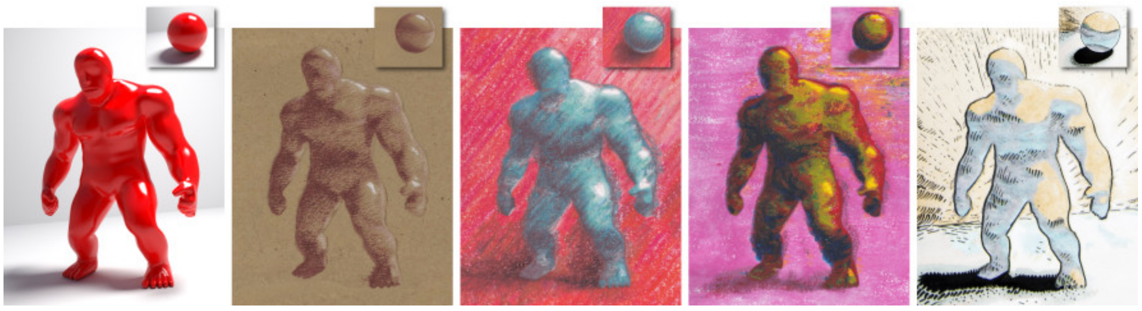


Figure 1: Example of style transfer. In left most image, we can see the model with its lightning and in following images, we can see multiple styles applied to this model. Source [2]

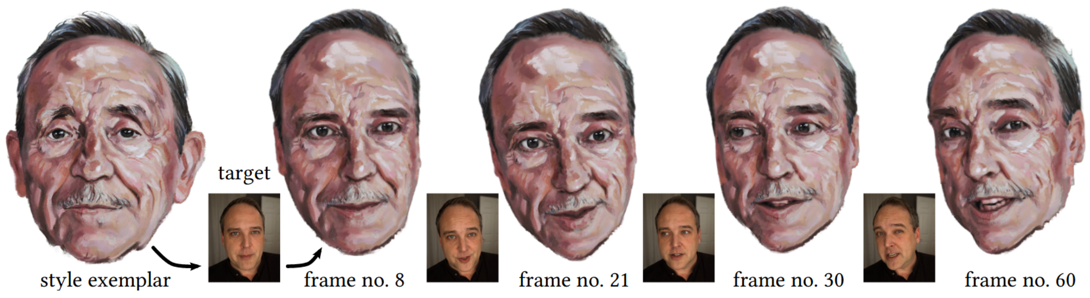


Figure 2: Example of the portrait style transfer. In the leftmost image, we can see the style example and then stylized images creating a video sequence. Source [1]

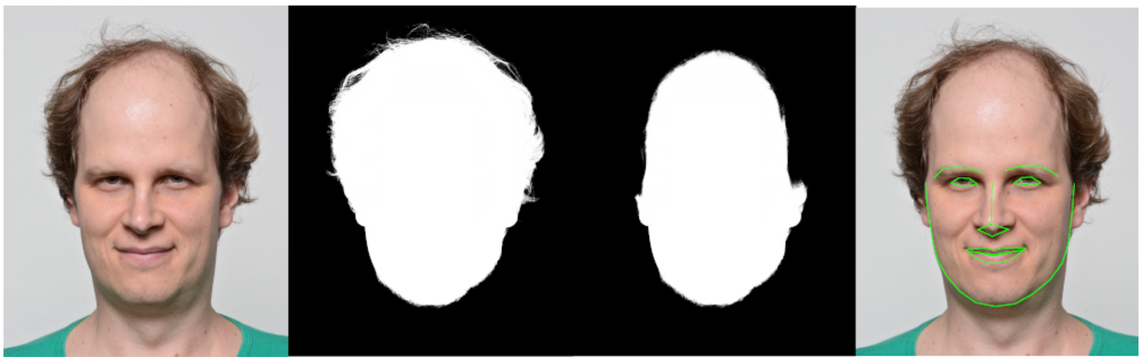


Figure 3: From left to right: Original image, its head mask, its skin mask and visualisation of face landmarks.

Chapter 1

Related work

1.1 Stroke-based rendering

This chapter presents work related to the topic of the thesis, like stroke-based rendering and image processing. Afterwards, examples of of stylization algorithms are introduced. There are many different studies on topic of non-realistic rendering, where one does not try to capture the reality as it is 9like the photo-realistic rendering0, but tries to give the image a unique style. Stylization of an image means choosing a specific style and, by using advanced methods, transfer the style onto the chosen image. One of the techniques used for style transfer is called stroke-based rendering. Algorithms of this technique try to replicate brush strokes, which are used when painting an image. Some of stroke-based techniques focus on giving brush strokes colors from the original image and then applying brush strokes to create a new image with them [3]. Another technique uses a geometry and colors of a scene in the image, which then uses predefined brush image and paints the image [4]. The article [5] introduces variables, which can be used to set a good brush stroke to replicate the content of the image. All those techniques have a huge variability and can perform multiple styles, but are only restricted to paint brushes. These methods work very well with video sequences. Their purpose is not to transfer a style, but to setup a style with settings and apply it to the image or a video sequence. How these methods perform can be seen in Figure 1.1.

1.2 Image processing techniques

Article [6] explains different aspects of watercoloring. They introduce a pipeline, which is able to render these effects onto a 3D model or a photo. The pipeline uses techniques such as color modification, which simplifies complex shading, then computes gradient intensity

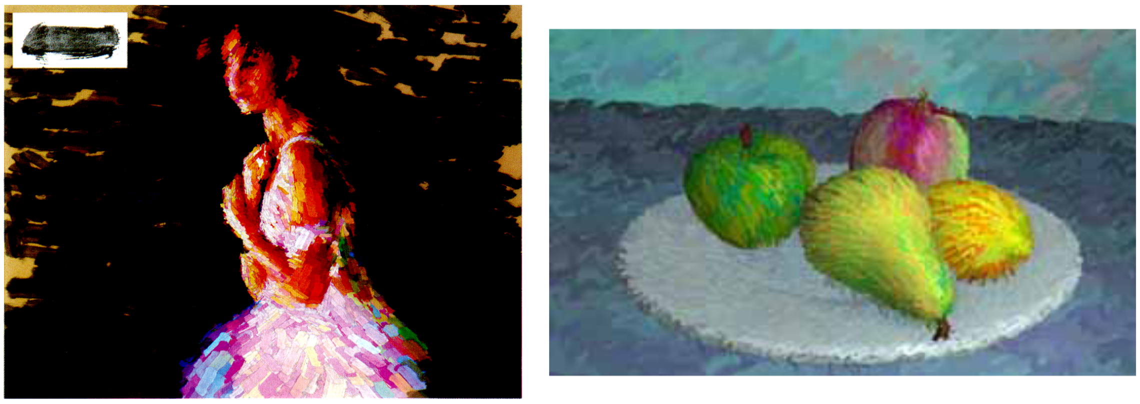


Figure 1.1: Examples of usages of stroke-based rendering techniques. Sources [3] (left), [4] (right).

to darken edges in the image. In final steps it applies series of textures, which have a purpose of locally distorting the image due to the paper granularity, globally diffusing colors that should create a visual effect of flow of the water, applying the paper texture to the result. Image processing framework introduced by Montedeoca et al. [7] uses image processing to transfer a style onto a 3D computer graphics. This technique can apply styles such as watercoloring, oil painting or charcoal painting. This framework has wide range of control to help a user create a wanted result. The effects are divided into four categories. Pigment-based effects which change how the colors are used within a specific style. Substrate-based effects, which affect the result based on its substrate, whether it is a paper, canvas or anything else. Edge-based effects are those that can be seen on edges of rendered object, which is used in cartoon style rendering or also in water color rendering. Abstraction-based effects might affect shape, detail or blending of colors during rendering. Examples of these two methods can be seen in Figure 1.2.

1.3 Example-based stylization algorithms

In 2006 Selim et al. [8] published an article concerned about transferring a painting style for head portraits. This article introduced the first single-example style transfer, which is not dependent on the chosen style. For their algorithm, they used convolutional neural networks. In the training stage of the algorithm, the network acquires knowledge of representations of the style example. Then, by using an algorithm [9], the style of example is transferred using the learned representation in the network. An example of results produced by this algorithm can be seen in Figure 7.

The article from Fišer et al. [1] uses the previous algorithm as a state-of-the-art algorithm for comparison and introduces a new algorithm, which would transfer style between portrait images. At first, the algorithm extracts information from both the example and

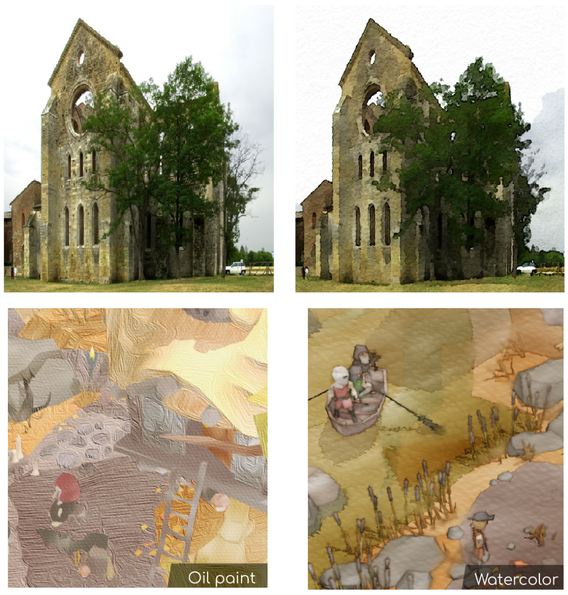


Figure 1.2: Examples of usages of image processing techniques. In top row we can see how a watercoloring algorithm can be used on a photo, Source [6]. In the bottom row we can see products of the framework on different 3D scenes with different styles, Source [7].

the original image, creating images called guide channels. This information is then stored in a form of new images called guide channels. These channels are used in guided texture synthesis, which finds the best patch (a small rectangular part of the texture with set dimensions) from the example to place in to the original image. As shown in Figure 1.3, results created by this algorithm can be different from those produced by the Selim et al. algorithm and can produce a result more resembling the style example.

The newest method by Futschik et al. [10] uses a generative adversarial network to create similar or better results than those generated by the method of Fišer et al. An advantage of this approach is that the whole process of stylization is very fast, and the algorithm used by this article can be used in real-time. The disadvantage of this method is that it needs training sets of examples and original images to set up its parameters for the specific style. A good approach is to use the Fišer et al. method to create a training set and then use the particular style parameters to produce results in a small amount of time. Even though this algorithm uses the Fišer et al. one as a training set, it can have different results, as shown in Figure 1.4.

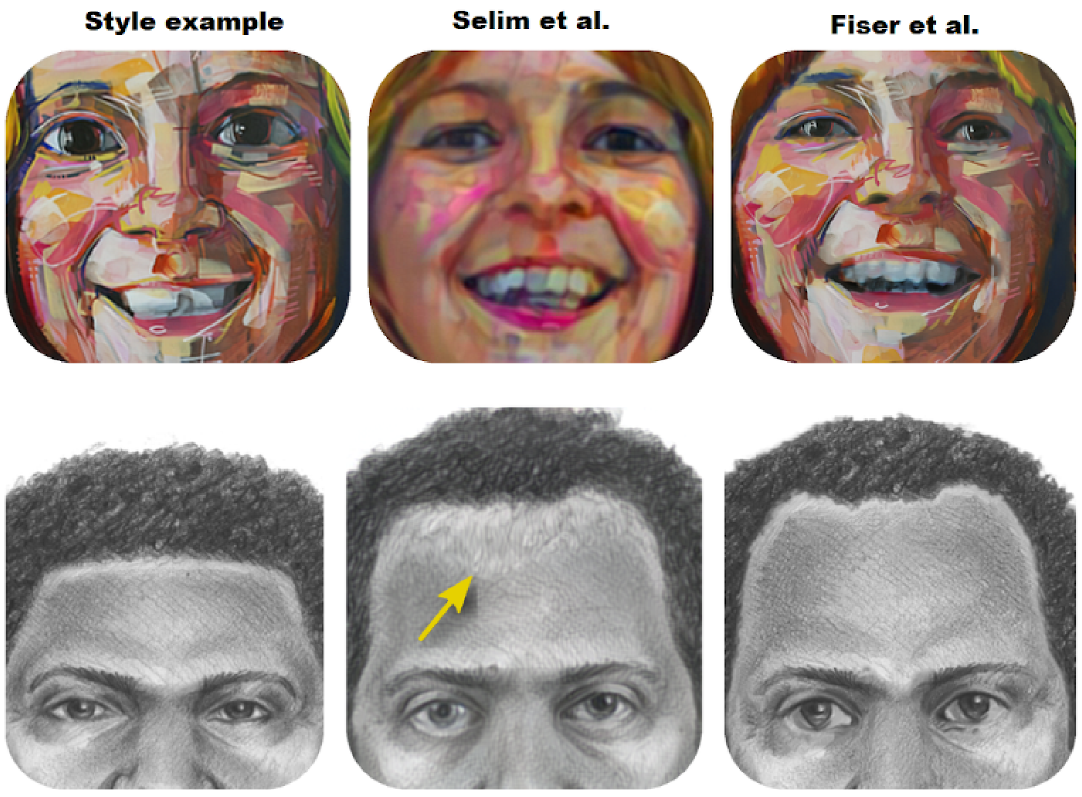


Figure 1.3: Comparison between two style transfer algorithms. In the first images, we can see that structures such as the circle on the left cheek are preserved in the Fišer et al. algorithm. In the second set of images, we can see warping artifacts in the algorithm of Selim et al., which is not visible in the Fišer et al. algorithm usage. Source [1].

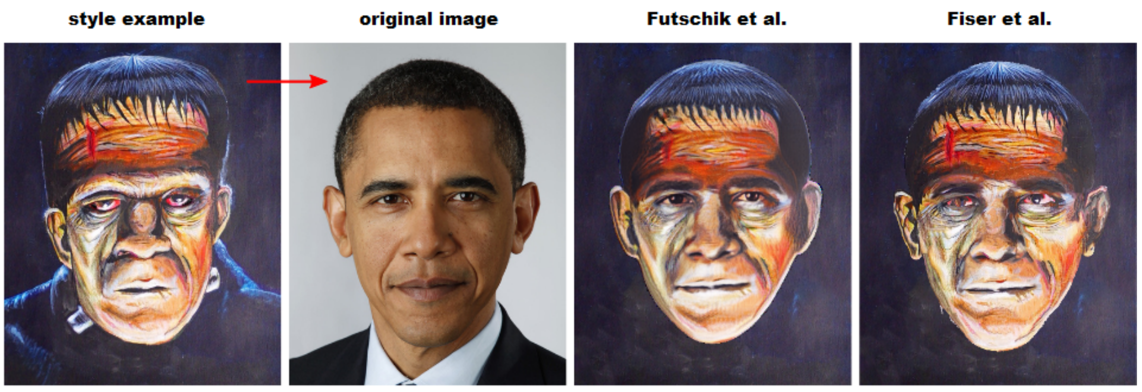


Figure 1.4: Comparison between two style transfer algorithms. In the Fišer et al. algorithm, we can see visual artifacts in the left eye, but the right ear is synthesised better in the Fišer et al. algorithm. Source [10].

Chapter 2

Background

Our target is to use an algorithm, which could be used to create training sets for the Futschik et al. method, introduced in the previous chapter. The algorithm of Fišer et al. eradicated visual artifacts, which the algorithm of Selim et al. can cause, and it is suitable for GPU implementation.

This algorithm finds the best function, assigning patches of an example image to some position in the output image. To define how good the function is, we define an error, also called energy function, which we try to minimize:

$$E(A, B, A', B', p, q, w, n) = ||A(p) - B(q)||^2 + \sum_{i=1}^{n} w_i ||A_i(p) - B_i(q)||^2. \qquad (2.1)$$

Here I(x), is an image patch, centered at pixel x with constant predefined width in the image I. Images used in this formula are A and B, which are the style example and the original image, respectively, and A_i is the i-th guide channel of image A, and B_i is the i-th guide channel of image B. Variable w_i is a weight of the i-th guide channel, and variable n is the number of guide channels that we use. The difference of patches is the sum of differences of individual pixels of patches with the same offset to the center of the patch. This function is used in the final part of the algorithm, where we already have prepared guide channels.From them, we synthesize the output image. There are multiple ways how this can be done, and the method used by Fišer et al. will be further explained in the section 3.4.

2.1 Guide channels

The basis of the algorithm is guide channels, which have to be created before the texture synthesis. They are images used in texture synthesis to help us preserve information from the original image. Therefore, we encode the data into them, which is essential for recognition of the person in the original image. There are three base guide channels, which are essential for our usage. Those are the segmentation guide, appearance guide, and positional guide. For stylization of video sequences, one could also use the temporal guide, which helps preserve the look of images in the output video sequence. Our focus is on single images, and therefore we have no use for this guide channel.

The segmentation guide stores position of important parts of the face. These are the eyes, the eyebrows, the nose, the lips, and the teeth. This guide helps to separate parts of the face from each other and also from regular skin parts. We also separate the skin from the hair. This guide does not store much information about the image. Still, it is very important because the human eye is very sensitive about visual artifacts around the eyes and mouth. Therefore, we enforce the usage of good patches in these specific areas. To encode this information into an image, we use different colors for different parts. To encode seven different parts into color with three components, we use very different colors for parts next to each other. For example, eyes are blue, and eyebrows are red. Since we cannot be sure that the algorithm marks those parts exactly, we want the edges of the parts as blurred together as possible. An example of a segmentation guide channel can be seen in Figure 2.1.

The second guide channel is the positional guide. As the name suggests, this channel stores position data of images. Style example and original image heads usually do not have the same shape. Also, heads are placed in different spots in their pictures. That is why we need this channel, as it helps the synthesis algorithm transform the shape and structure of the head. If the stylization algorithm aimed to change the shape of the head of the style example to the shape of the original image, this would be the only guide channel we would need. This channel also stores how differently are the parts of the faces shaped. The data stored in the image is a function, which tells us to which pixel of the style example does a chosen pixel corespond. Therefore pixels in the middle of the forehead of the original image should store indices to pixels in the center of the style example's forehead. An example of a positional guide visualization can be seen in Figure 8 in attachments. The final positional guide channel does not have visualization lines in it, as they could create artifacts, but they serve for better visualization of the data.

The last guide channel is the Appearance guide. The purpose of this guide is to preserve the look of the person in the original image. The appearance guide of the style example is just the image transferred into grayscale. We also turn the original image to grayscale, but we still need to change the appearance guide to have global intensity levels and local contrasts of the appearance guide of the style example. In appearance guide images, we can see scars, wrinkles, shadows, and other features of the input face. This channel is significant because it tells the algorithm to create the output image with the same facial features as the original image. We can alter the importance of this channel with the weight in the error function 2.1. Sometimes we want the face to look more like the style example, and for that purpose, we would use a small weight. On the other hand, if we would like the output image to look very similar to the original image and we do not need the similarity to the style example, for this purpose, we would use high weight in the error function. An example of an appearance guide channel can be seen in Figure 2.1. What impact on a final result has an absence of a guide channel can be seen in Figure 2.2.

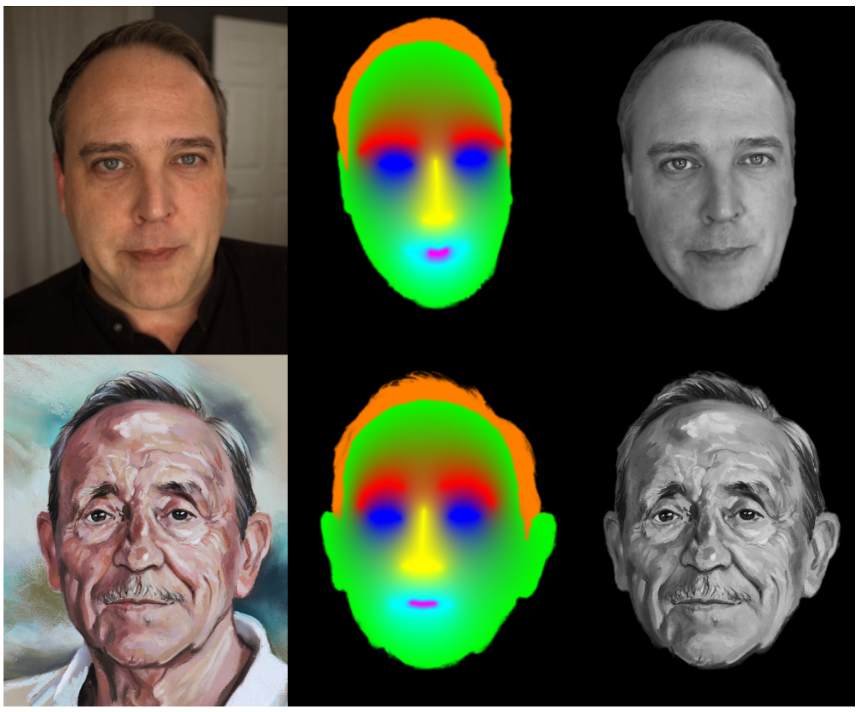


Figure 2.1: Visualisations of segmentation and appearance guide channels of original image (top) and style example (bottom). Segmentation channel is the second one and appearance guide is the last one in each row.

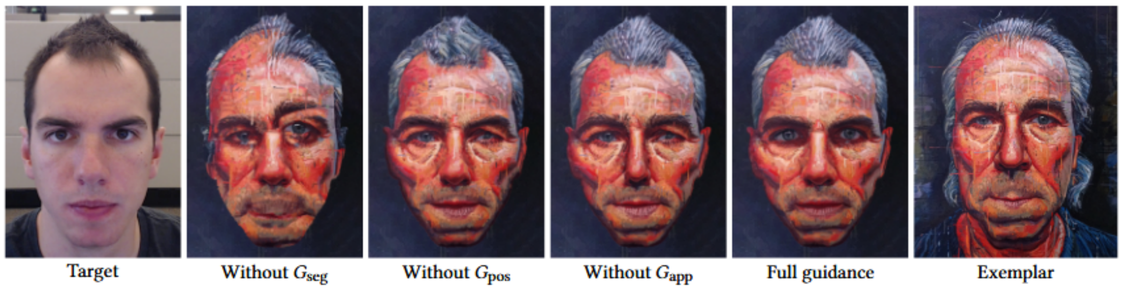


Figure 2.2: Example of absences of guide channels during synthesis and how it affects the result. We can see that without the segmentation guide, the synthesis algorithm can not compile parts of the face together well. The difference of missing positional channel is the smallest, but we can see that the we lose the positional correctness of smaller features. With absence of appearance channel the resulting portrait loses all facial features of the person in the original image and only looks like a deformed face of the style example. Source [1].

Chapter 3

Implementation

In the previous chapter, we explained the basis of the Fišer et al. algorithm and what guide channels are. This chapter will describe what algorithms we use to obtain these guide channels and go over the synthesis algorithm in more detail.

The input for the algorithm in our case is six images and two sets of positions. The first two images are the style example and the original image. The following two images are masks, which denote head regions and the last two images are masks, which show in which parts of images we can expect skin. Skin in this context is the head mask but without the hair regions. The last input data are two sets of 68 positions, which are called landmarks. They store positions of fixed points on a face. Landmarks are the chin, the right eyebrow, the left eyebrow, the nose, the right eye, the left eye, the lips and the teeth points. Connected landmarks that form individual face features can be seen in Figure 3.1.

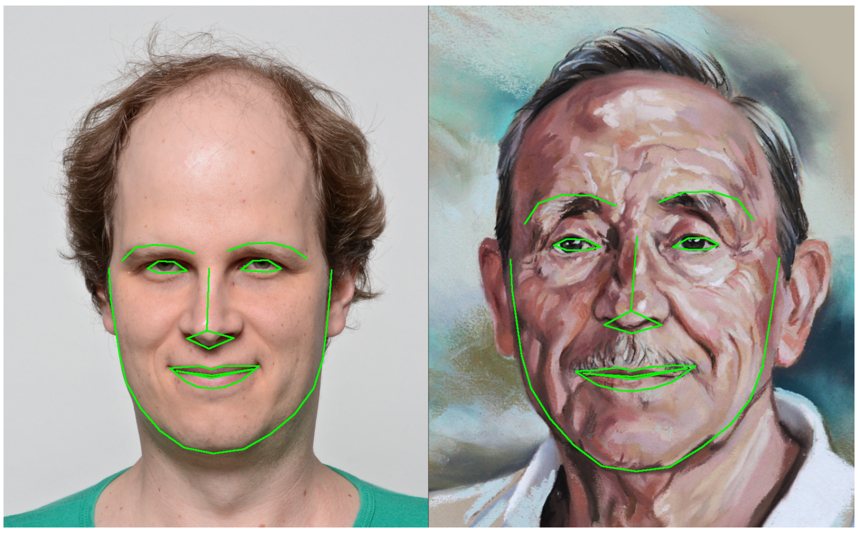


Figure 3.1: Visualisation of face features created by connecting landmarks.

In some cases, the points create a polygon around a part of the face, as seen in

Figure 3.1, e.g., the eye. In others, they depict a poly-line, where we should expect the part of the face, e.g., eyebrow. Algorithms detecting face landmarks usually work on basis of deep learning [11]. Head mask can be obtained by getting a portrait segmentation by method of Shen et al. [12]. Using the information about portrait segmentation, using known landmarks, we can locate head of a person in the image to obtain a head mask. Skin mask can be obtained by setting a threshold of pixel's chance of being a skin pixel by comparing its color in YCbCr color space with cheek pixels.

As masks from inputs are usually only ones and zeros and nothing in between (there are no areas with a percentage between 0 and 100), we blur masks before using them to generate guide channels. Without this change, some parts of hair could be missed, and also masks in space between skin and hair usually are not very smooth, which could be seen in insufficient segmentation guide results. These results then create unsatisfying output from the synthesis, and this simple change completely solves this problem. Another approach to this problem is with matting. This approach might lead to even better results, but it is much more complicated and slower, and therefore we decided to choose the simpler variant.

3.1 Segmentation guide

To create a segmentation guide, we use the method of diffusion curves [13]. This method is usually used to help artists draw a particular style. As we have minimal demands from this method, we do not need all its utilities, and we only focus on useful ones. The input to the method are curves (for our purpose, poly-lines), which have their specific color. Connecting landmark points generates these poly-lines. Then we use diffuse those curves into the image. The diffusion is expressed as a solution to a Poisson equation, and the colored curves are local constraints:

$$\Delta I = div\ \mathbf{w} \tag{3.1}$$

$$I(x, y) = C(x, y) \text{ if pixel (x, y) has a color value} \tag{3.2}$$

where Δ and *div* are the Laplace and divergence operators, respectively. I is the output image of this equation and C is the image with drawn curves. Equation 3.2 tells us that we do not want to change values that are already stored in the image, and equation 3.1 indicates the smooth diffusion in the image we want. In our case, w, the gradient field is always constant as we do not need any advantages of having a different gradient field. Computing these equations means solving a large linear system, which can be very time-demanding. That is why we use the Gauss-Seidel iteration method. This method iteratively changes values in the image until the solution converges, or if an exact result is not required, we can specify a constant number of iterations. In each iteration, we go through each pixel in the image and create a new image using the following formula

$$\bar{I}(x, y) = \frac{1}{4}(I(x+1, y) + I(x-1, y) + I(x, y+1) + I(x, y-1) - b(x, y)). \tag{3.3}$$

Here b stands for the divergence, and as mentioned before, the gradient field is constant. Therefore divergence in any point is equal to zero. $\bar{I}$ Is the output image of the iteration, and I is the input image of iteration. As we use large fields and it would take too many iterations to converge, we first downscale the image, find a solution, and then upscale the solution and run some iterations until we get the original resolution. In the down-scaled images, we only need few iterations to get a good result, and when we upscale, we use more iterations to propagate more minor changes. How segmentation changes with each upscale can be seen in Figure 3.2.

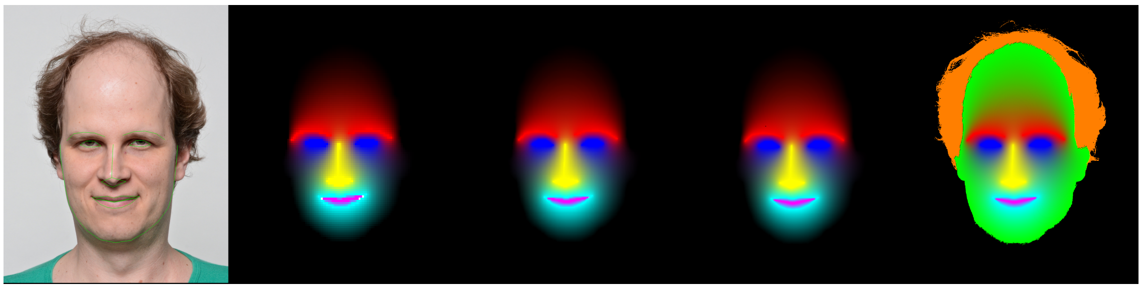


Figure 3.2: Pictures from left to right: the original image with landmarks displayed, the first iteration of creation of segmentation guide down-scaled 4x, the second iteration down-scaled 2x, the last iteration with no scale, the segmentation channel. Segmentation guide in downscaled iterations without skin and hair. We can see that all three scales are fairly similar, and the upscaled iterations only do small local changes.

3.2 Positional guide

To produce the positional guide, we need to find a transformation for each original image pixel. To find this transformation, we used a method of image deformation using moving least squares [14]. This method is used to deform the image by moving some of its pixels (called control points). We use it to deform the original image by moving its landmarks to positions of the style example's landmarks. However, we do not use the deformation to change the original image but store it as the positional guide. How the deformation changes the style example to be more spatially similar to the original image can be seen in Figure 3.4.

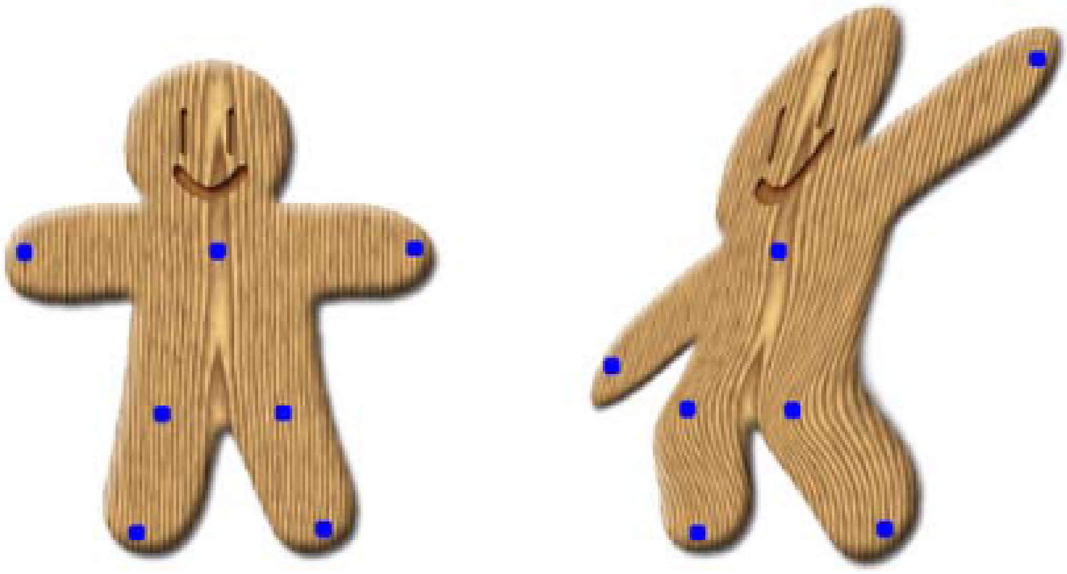


Figure 3.3: Deformation of a picture by its moving control points with affine matrix transformation for each pixel. Source [14].

Finding the perfect deformation is defined as finding the best affine transformation $l_v(x)$ that minimizes

$$\sum_i w_i |l_v(p_i) - q_i|^2 \tag{3.4}$$

where p_i and q_i are position vectors of a control points in the first and second image respectively. The point v is a point in the original image that we are trying to find a transformation for. The variable w_i is a weight of p_i with the respect to v and it is defined

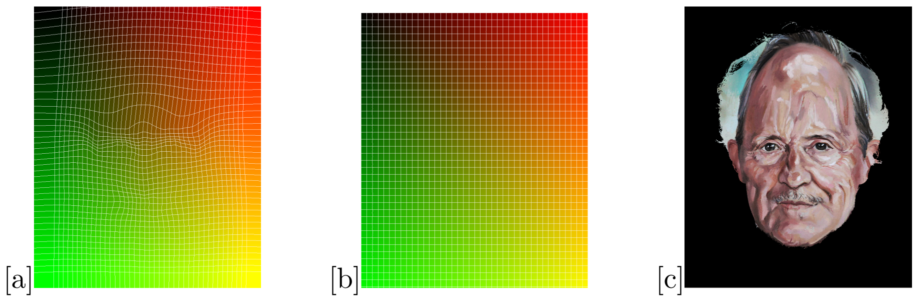


Figure 3.4: Visualisation of the deformation effect of the positional guide on the style example.

as

$$w_i = \frac{1}{|p_i - v|^{2\alpha}}. \tag{3.5}$$

It can be seen that it is the one over the distance between those two points to the power of α, which is a constant. That help to make closer control points more important than those that are far away from v. As mentioned before, $l_v(x)$ is an affine transformation, and therefore we can define it as

$$l_v(x) = xM + T \tag{3.6}$$

where T is a translation vector and M is a linear transformation matrix. To compute T, we need to find how is the whole image moved. We compute weighted centroids, and knowing M, we get the formulae

$$T = q_* - p_* M, \tag{3.7}$$

$$p_* = \frac{\sum_i w_i p_i}{\sum_i w_i}, \tag{3.8}$$

$$q_* = \frac{\sum_i w_i q_i}{\sum_i w_i}. \tag{3.9}$$

As we do not know the matrix M, we compute it first and then compute T. For the linear transformation matrix, we use the following formula

$$M = (\sum_i \widehat{p}_i^T w_i \widehat{p}_i)^{-1} \sum_j w_j \widehat{p}_j^T \widehat{q}_j, \tag{3.10}$$

where $\widehat{q}_i$ and $\widehat{p}_i$ are offsets from centroid and are computed as $\widehat{x}_i = x_i - x_*$.

3.3 Appearance guide

To create an appearance guide, we need to create grayscaled images from the inputs and then use a method of Shih et al. to change the global intensity levels and local contrast values in the grayscaled original image to be equal to those of the grayscaled image of the style example. To adjust the image, we used the style transfer method for headshot portraits [15]. The basis of this method is to decompose both images into Laplacian stacks, compute local energy maps of both images and lastly transfer local energy and residual of the style example grayscaled image. Laplacian stack of a grayscale image I can be defined as an ordered set of images L, where

$$L_i[I] = \begin{cases} I - I \bigotimes G(2), & \text{if } i = 0 \\ I \bigotimes G(2^i) - I \bigotimes G(2^{i+1}), & \text{if } i > 0. \end{cases} \tag{3.11}$$

Here $i = 0...n$, where n is number of levels of Laplacian stacks, $\bigotimes$ is the convolution operator, $G(\sigma)$ is a 2D normalized Gaussian kernel of standard deviation σ. In the first iteration, we subtract the blurred image from the original, which results in an image where the high-frequency features of the image are present. For each $i > 0$, we create a blurred image and subtract it from the blurred image created in the last iteration. This results in lower frequency information in the image. The last part of the decomposition is a residual R, which is defined as

$$R[I] = I \bigotimes G(2^n). \tag{3.12}$$

When we look at this decomposition, we can see that sum of all the images in Laplacian stack added to the residual creates the former image. This information is later used to create a new Laplacian stack with a residual and then add everything together to get a new image. Visualisation of Laplacian stack and residual can be seen in Figures 9, 11. To adjust the global intensity level of the original image, we use the warped residual of the example as a residual of the output image. The next step is to change the Laplacian stack of the original image to work better with the example residual and change local contrast values. This problem can be solved by creating maps of the local energy of both images. To get these maps, we use the following formula:

$$S_i[I] = L_i^2[I] \bigotimes G(2^{i+1}). \tag{3.13}$$

That means we need to create squared Laplacian stack and then blur the values with a Gaussian kernel of the given standard deviation. Comparison of a Laplacian stack to a local energy map is in Figure 10. In the next step, we use the positional guide to warp the local energy map and residual of the example. With warped data we can create output Laplacian stack using formula:

$$L_i[O] = L_i[I] * \text{Gain}, \tag{3.14}$$

$$\text{Gain} = \sqrt{\frac{S_i[E]}{S_i[I] + \epsilon}}. \tag{3.15}$$

Here I is the original image, E is the style example, and ϵ is a small constant to avoid division by zero. The square root in the formula makes up for the square in the formula of the local maps in 3.13.

If we used Gain as defined in formula 3.15, we would get a good result for many typical portraits, but there would be difficulties if we would try to use this method on an image with glasses or a mole and where the other image would not have it. That would create a peak in the Gain image, which would create visual artifacts in the final image. For example, there would be slightly visible glasses even though the person in the original portrait did not have them. To remove this problem, we use robust gain, which replaces Gain in the formula 3.14. Robust gain is defined as

$$\text{RobustGain} = max(min(Gain, \theta_h), \theta_l) \bigotimes G(\beta 2^i), \tag{3.16}$$

where θ_l and θ_h restrict the range of the original Gain, and β is a constant that changes the deviation of the Gauss kernel. The convolution with Gauss kernel blurs spikes in the Gain and makes the whole Gain image more smooth. The final step is to add the output Laplacian stack to the warped residual of the style example, and we get the output image.

As we do not want to have the background affect the appearance guide, we use a head mask, which is given as an input. It is also necessary to alter the convolution with Gauss kernel in order to work well with the mask. Therefore, we replace convolutions in formulae 3.11, 3.12, 3.13 and 3.16 as follows:

$$\text{Image} \bigotimes G \to \frac{(Image * Mask) \otimes G}{Mask \otimes G}. \tag{3.17}$$

For the constants used in before-mentioned formulae, we assign constants $\theta_h = 2.8, \theta_l = 0.9, \beta = 3$ and $n = 6$ as recommended in article [15].

3.4 Synthesis algorithm

After creating the guide channels, we need to use the synthesis algorithm to get the output image. We do not need a special algorithm for headshot style transfer, but we can use the one that can be used for numerous other usages. Multiple different algorithms retrieve a high-quality output and are usually based on similar principles. The two important algorithms for this thesis are of Kaspar et al. [16] and of Fišer et al. [17].

The basis of both algorithms is the nearest neighbor field (NNF), an array of data with the size of the original or style example image (depends on the algorithm). Each element of the array represents a patch in the image and the data in the element is the patch in the other image. These patch correspondences are chosen in a way that the difference between them is minimal according to the energy function 2.1. An algorithm that can find a nearest neighbour field is not tied to the synthesis algorithm, but the patch match algorithm is usually recommended. This algorithm will be explained in-depth in section 3.4.3. The aim of those algorithms is to find the best possible NNF for each patch in the original image, and then by averaging colors of patches around each pixel, we get the output image. In practice, algorithms do not use a patch for each pixel but place them with an offset of half of the patch size. Both algorithms of Kaspar et al. and of Fišer et al. use the nearest neighbor field, but they use it differently. How they use it will be explained in the following sections.

3.4.1 Algorithm of Fišer et al.

The basis of this algorithm is the EM-like iterative method, which finds an NNF for the currently found solution and then changes the solution according to the NNF. The NNF that this algorithm uses stores the best patch of the original image for each style example patch. This might be counter-intuitive because the output image is made out of style example patches in the original image. We do not use this NNF data to apply it directly to the output image, but we first have to throw away erroneous patches and then use the rest to build up the output image. Erroneous patches exist, because the NNF filling algorithm not always works perfectly and also because some patches of the style example might not be well represented in the original image. Therefore, these patches should not be used to produce the output image. To find out which patches are still feasible and which are not, we need to introduce a filter method. With filled NNF and having the energy of each element in the NNF, we sort all the elements in NNF by their energy. The more energy the element has, the worse the patch assignment is. To find a line where the energy is too high, and elements with higher energy should not be utilized, we have to approximate the function $f(x)$, where x is the index of the element and the value of the function is its energy. In practice, functions like this usually have a hyperbolic shape. The first elements have very small errors, and once the elements begin being erroneous, the energy function rapidly increases. The hyperbola, which we will be fitting can be defined as $f(x) = (a - bx)^{-1}$. The point, where the hyperbola starts being too steep is known as a knee point. In this hyperbola, the knee point is positioned in the point of value $f(x)' = 1$ and the point has index $k = \sqrt{1/b} + a/b$. Visualization of this is method is shown in Figure 3.5. Points above index k are not used. We also set a feasible error budget $T = \sum_{x=0}^{k} f(x)$.

With this budget, we can define the feasibility constraint of the problem as

$$\sum_{p \in A} \min_{q \in B} E(A, B, A', B', p, q, w, n) < T. \tag{3.18}$$

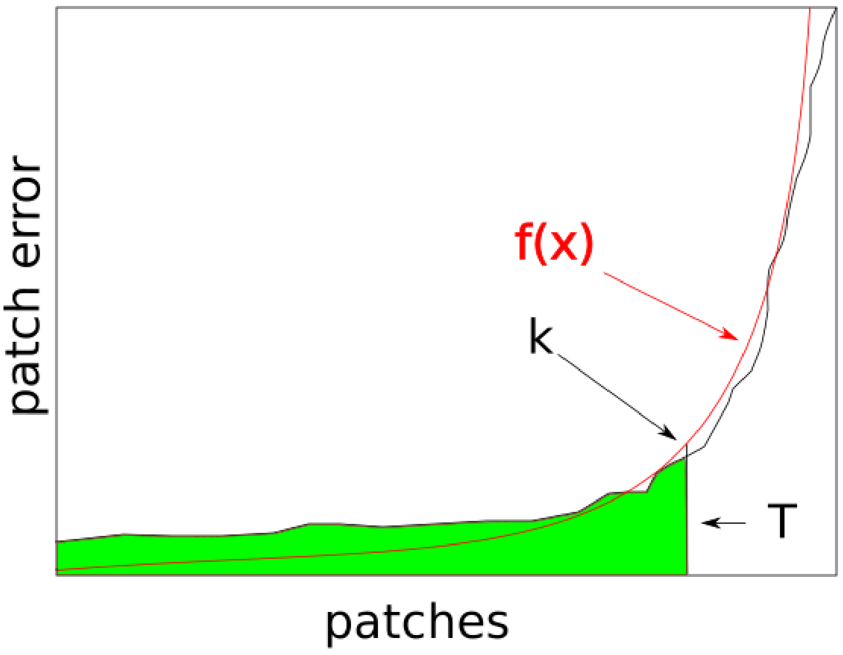


Figure 3.5: Visualisation of the patch error, fitting f(x) and setting error budget T. This image is based on an image from [17].

In each iteration, we first find the NNF using the patch match algorithm and then use good style example patches in the output image. We iterate until we have covered all or a little less than all (95%) patches. The last patches can be found by searching for the best possible patches in style example for the remaining patches of the original image with patch match algorithm. This process is sped up by running a coarse-to-fine synthesis.

This approach makes usage of patches more uniform in each iteration, we might choose each patch with good enough energy. By using a feasible error budget, we do not force uniformity. Therefore, if we need to use some part of the head more, for example, if the person in the original image has a bigger nose than the person in the style example, we can. Usually, the proportions of faces are similar, and therefore this method does not need many iterations to fill the output image.

3.4.2 Algorithm of Kaspar et al.

This algorithm also utilizes an EM-like iterative method, but it has a different approach to make patch usage uniform. It uses the NNF, which finds the best patch for each patch in the original image and then uses this NNF to create the output image out of it simply by averaging colors from each patch. This happens in every iteration. To enforce uniformity, this algorithm changes the formula for energy 2.1. It introduces new variables to increase the patch usage uniformity. The first variable is occurrence for a pixel with coordinates (x, y):

$$\Omega(x,y) = |\{s_i \mid (x,y) \in N(s_i)\}|, \tag{3.19}$$

where $N(s_i)$ is the set of pixels in the patch s_i. Occurrence tells us how many times pixel with specified coordinates was used in a patch that is used in the output image. To enforce uniformity, we want the occurrence in the output image to be as low as possible. The ideal occurrence for each pixel in the image is

$$\Omega_{best} = \frac{|O|}{|S|}N^2, \tag{3.20}$$

where $|O|$ and $|S|$ are areas of original image and style example respectively and N is the width of a patch. We define

$$\Omega(s_i) = \frac{\sum_{(x,y)\in s_i} \Omega(x,y)}{N^2} \tag{3.21}$$

as an occurrence in the patch, which is the average occurrence of pixels in the patch. With this defined variable we can finally alter the energy function 2.1 as

$$E(A,B,A',B',p,q,w,n) = ||A(p) - B(q)||^2 + \sum_{i=1}^{n} w_i||A'_i(p) - B'_i(q)||^2 + \lambda_{occ}\frac{\Omega(p)}{\Omega_{best}}, \tag{3.22}$$

where λ_{occ} is a controlling variable, with which we change how much we want to enforce the uniform patch usage. There is also one minor change in the patch match algorithm usage and it will be explained in the end of the following section. We also use this algorithm with coarse-to-fine synthesis to get results faster.

3.4.3 Patch match

Patch match is an algorithm, which for two images retrieves the NNF for patches between them. As we use the patch match differently in both previously mentioned algorithms, we will call used images A and B. For each patch in the image A, we search for the best corresponding patch in the image B. There are three basic steps in this algorithm: initialization, propagation, and random search. Initialization of the NNF might be completely random, which will help us to find different correspondences in each algorithm usage. If we have some previously known information, which might help the algorithm to start with better NNF, we can use that. One way to start is to use the previously found NNF, but the disadvantage is that we might have a lesser chance to find some global minimum(when talking about the energy in the synthesis) instead of searching for minor improvements around a local minimum. When we have initialized NNF, we iterate with improving techniques. The propagation improves the energy of the current patch by searching for good correspondences in neighbor patches. When we use propagation on a patch with coordinates (x, y), we look to data that is stored in NNF with coordinates $(x - 1, y)$ and $(x, y - 1)$. Then we have to translate this data to match the current patch. If both input images have the same sizes, we use the patches with translation $(1, 0)$ and $(0, 1)$, respectively. When both images have different sizes, we might need to scale those vectors. With this method, once one patch finds good patch correspondence, this good correspondence propagates throughout the whole image. Even though it might not be the optimal solution, it saves many steps, which would otherwise have been done by random search. Once we compute the energies, we decide if we want to keep the current patch correspondence or use one of the propagated ones. We compare energies and choose the match with the lowest energy. This step is portrayed in the Figure 3.6.

With this method, we can propagate good matches across the image, but we need a tool to find a good match in the first place and also to find a better solution if the one given by propagation is not optimal. To achieve those goals, we use random search. As the name suggests, we search the image with some randomness, and then we compare the found patch with the one we already have and store the better one. As we do not want to search in the whole image uniformly, we have multiple searches in each iteration, and we change the radius in which we search. First, we search the whole image. This helps us to find distant good matches, and even though they usually do not end up being the best match, it is necessary to search in all parts of the image. In the following searches, we half the search radius until the search radius is below one pixel. Patch coordinate that is chosen can be described with the following formula:

$$u_i = v_0 + w\alpha^i R_i \tag{3.23}$$

where u_i is the coordinates of the found patch, v_0 is the current coordinate patch stored in the NNF, R_i is a random vector with both coordinates having a value between -1 and 1. α is a constant with value 0.5, and i is the index of the random search starting at 0 and ending when the search radius is too small. Visualisation of this step is displayed in Figure 3.7.

After initializing the NNF, we alter between propagation and random search. The number of iterations needed might depend on the application, but usually, after 4 to 5 iterations, the image converges. The change required for the algorithm of Kaspar et al. is that when a new patch candidate is found in the random search phase, we might not use it once it has occurrence two times larger than the ideal one. The only exception is when the occurrence of the patch being replaced in the NNF is even higher than the one of the new patch candidate.

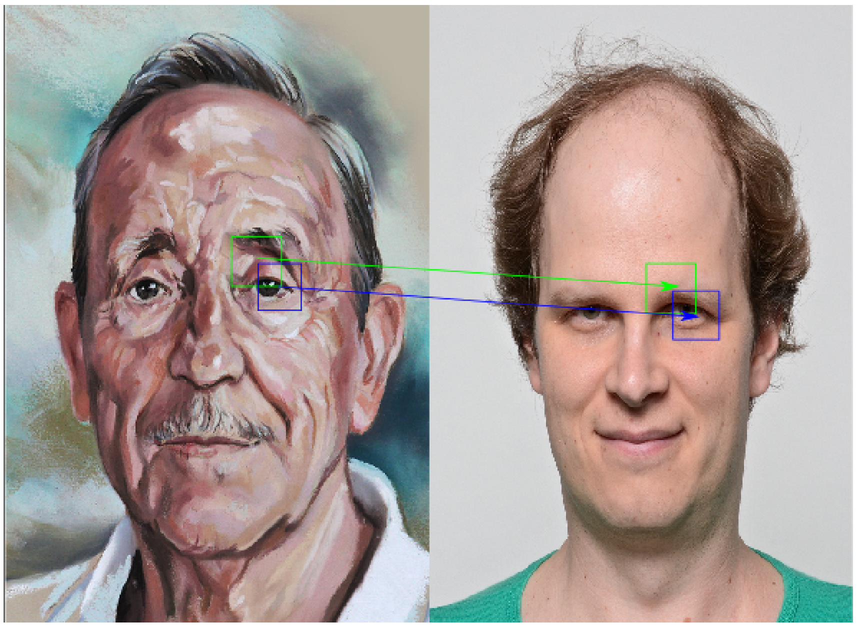


Figure 3.6: Visualisation of the propagation step, where we have a good correspondence for the green patch and by translating the match, we can get a good patch for the blue patch.

3.5 Implementation details

In this section, we will explain all the parts of implementation, which might not be understandable from their mathematical formulation in the previous sections. In the implementation of segmentation channel, we first had to transform landmark points in poly-lines to apply the previously mentioned method on them. We used a Bresenham algorithm [18] to connect specified pairs of points. Bresenham algorithm creates one pixel wide line without any anti-aliasing effects and runs in a linear time with respect to the distance of points. Appearance guide channel uses a lot of convolutions. Convolution is a very time demanding operation. For two functions defined by image of N pixels and Gauss kernel of M pixels, the convolution has time complexity $\mathcal{O}(N * M)$. To get better run times we use separability of a Gauss kernel to get a time complexity. Separability means, that having a Gauss function $G(x, y)$, we can express it in a way $G(x, y) = g(x) * g(y)$, which is evident from the Gauss function definition 3.24.

$$G(x, y) = \frac{1}{2\pi\sigma^2} exp^{-\frac{x^2+y^2}{2\sigma^2}} \tag{3.24}$$

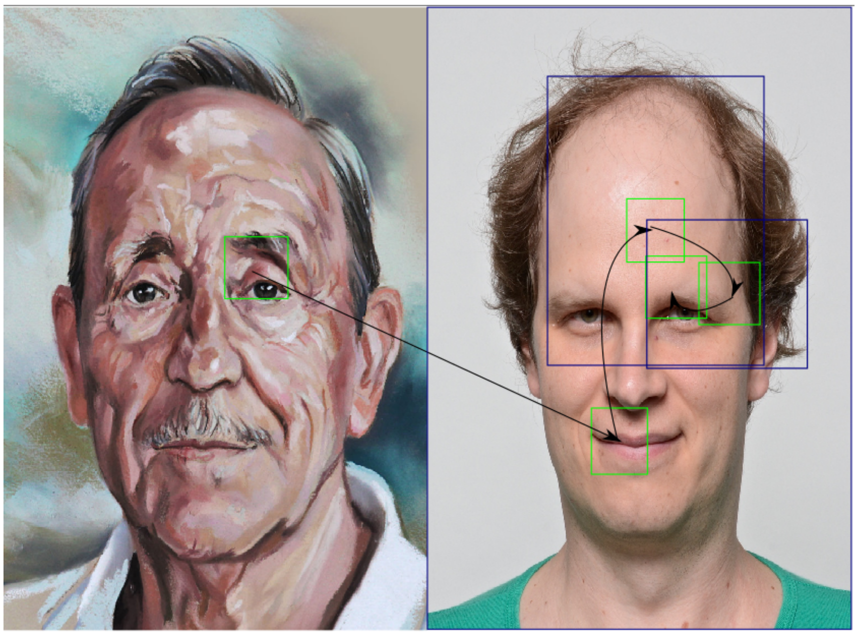


Figure 3.7: Visualisation of the random search step, where we have a bad correspondence for the green patch and by searching in shrinking radii (dark blue), we get progressively better results and end up with a good one. Arrows indicate how the best found patch moved after one random search in the iteration.

We use this quality to reduce the 2D convolution into two 1D convolutions, where one is for rows and one for columns. This change gives us a complexity of $\mathcal{O}(N * \sqrt{M})$, which can significantly speed up the algorithm. Most of the used methods use an algorithm with $\mathcal{O}(N)$ time complexity, because we have to compute an operation for each pixel. In none of these operations, we need a value of a previously computed pixel in the same iteration. This fact helps to easily parallelize algorithms by running each computation on a separate thread. The only algorithm with huge time complexity that left is the optimized convolution with each thread doing $\mathcal{O}(|G|)$, where G is our representation of Gauss kernel as an array. We used a technique to reduce number of elements in the array. We set a constant, which represents the smallest value, which can be represented by the normalized Gauss function. By changing this constant, we can fasten the algorithm by not using small function values that does not make any noticeable difference. To find the size of the gauss array, we had to use inverse function to the one dimensional Gauss function 3.25.

$$x = 2\sigma\sqrt{\ln \frac{\frac{1}{2\pi\sigma^2}}{G(x)}} \tag{3.25}$$

Chapter 4

Results

We implemented all three above-mentioned channel guides. We used implementation of the algorithm of Kaspar et al. implemented by Ondřej Jamriška called Ebsynth [2], with which we created output images for testing sets of style examples and original images. We aimed to generate output images looking as close to ones in Fišer et al. article as possible. The algorithm of Fišer et al. recommends setting weights of guide channels in energy function 2.1 as follows: segmentation and positional channels have weight equal to 5, appearance channel has a weight set to 1 except in the eye and mouth regions, where the value should be set to 5. As the used synthesis algorithm does not allow to change the weight depending on a mask, we had to use a special appearance guide channel for the eye and mouth, which was just the normal appearance guide influenced by teeth, mouth, and eye masks. This adjustment does not solve the problem entirely, but it makes better results than using only one weight with the appearance guide channel. How the special appearance guide channel looks in comparison to the normal appearance guide channel can be seen in Figure 4.1. How good are the results of guide channels and of the synthesis algorithm is explained in the following sections.

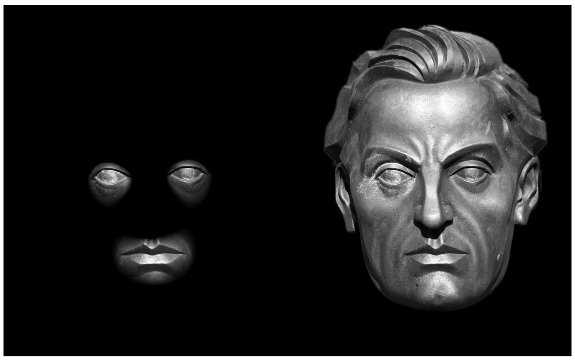


Figure 4.1: Comparison of special and normal appearance guide channels.

4.1 Segmentation channel

In Figure 4.2 we can see a comparison of a segmentation channel created by our implementation and one that is used as an example of the segmentation channel in portrait synthesis on the site of Ebsynth. The significant difference, which is highly noticeable, is the difference between inner mouth representation in magenta color. This major difference is caused by differently generated landmarks as our landmarks, provided externally, set the inner mouth as seen in our picture. As the person in the portrait does not show their teeth, the landmarks may be different, but this is not a problem as they are consistent within one usage. One smaller difference is that colors in our implementation seem to be more diffused into the skin color. This difference would be significant if we used one segmentation guide channel of our implementation and one out of the other implementation in the synthesis to get an output image. Some regions, which might be colored as the edge of eyebrows in our implementation, could be marked as skin segment in the other. But as long as the diffusion of segments is consistent within the implementation, this does not change the resulting image. One last slight difference is the blurred out edges of skin and hair regions, which we mentioned in the implementation chapter 3.

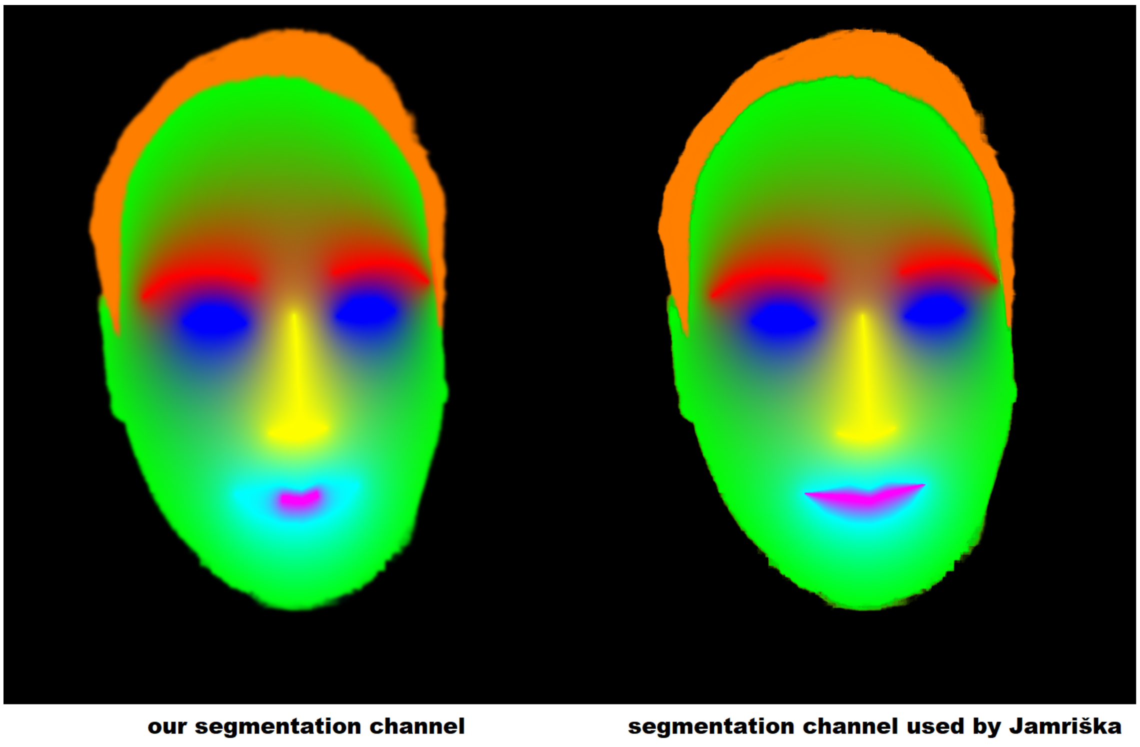


Figure 4.2: Comparison of segmentation channel created by our algorithm and a one that is used as an example by Ebsynth. Source of the second image [2].

4.2 Positional channel

Comparing positional channels is very difficult as images without visualization lines are tough to compare side by side, and the visualization with lines might be different in each implementation as their positions are not described. To observe the function of the positional guide, we provided side by side comparison of the deformed style example with the original image in Figure 4.3. We can see that the deformed parts of the face, which are near landmarks, are very well aligned with the ones in the original image's face. There are no sudden deformations, which would seem unnatural in the face region. Problematic parts might be seen in the forehead and hair area. As there is only one landmark in the area, the deformation far from any landmark appears unnatural. In our implementation, we use only one landmark for the forehead, situated at the highest point of the skin mask. A better guide channel would be created by having more landmarks or creating multiple new landmarks as the one mentioned above, but finding a landmark, which is in the same place in each portrait is not a straightforward task. Since even the one landmark we created had its artifacts as seen in Figure 4.4. As we can see, the highest skin point is not in the middle of a forehead in all faces, and in the example, it is on the right side of the forehead. This creates a deformation that puts the middle of the style example's forehead to the right side, and the right side of the forehead is shifted even further.

We chose simply to use an affine deformation in our positional guide channel algorithm, where using a similarity matrix might also be considered correct. Affine deformation can perform shear, which is not very usable in our usage. It can also do a non-uniform scaling, which can be helpful as heads are not of the same height to width ratio. The last change that could have been done to the algorithm of the positional guide is setting the alpha in the formula 3.5. We used $\alpha = 2$, but the constant value is not specified in any mentioned articles. The constant tells us how far the point has to be from its closest control point to be affected more by other control points. Setting this constant with a high or low value could create visual artifacts near control points, and with minor changes to the value, the result is nearly the same.

4.3 Appearance channel

To show how the algorithm changes the grayscaled version of the original image into its appearance guide, we offer the comparison in Figure 4.5. As can be seen, the global intensity levels are much closer to the image of the style example. The result is brighter, and the intensity difference between the left and right sides of the face is well visible.

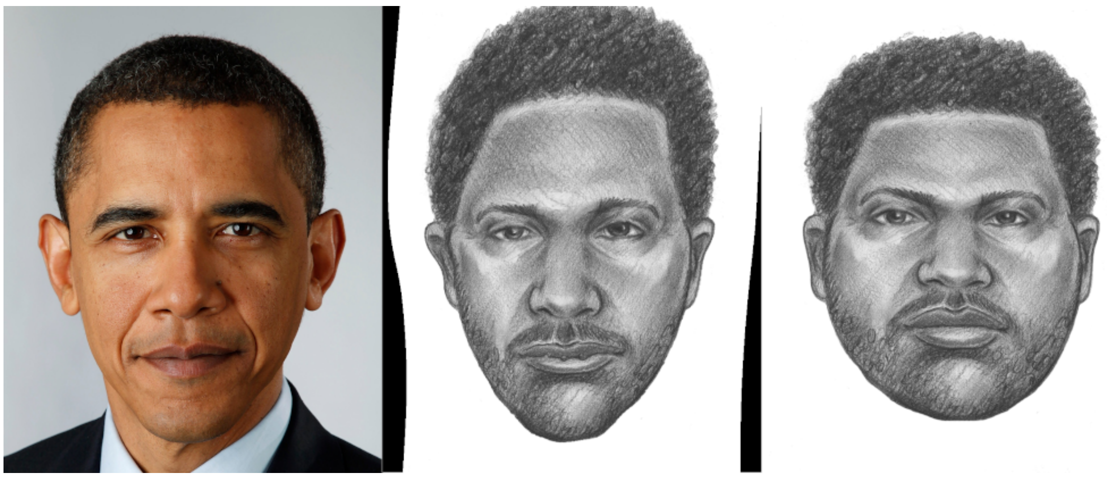


Figure 4.3: Visualisation of positional channel usage applying deformation of the style example on the original image.

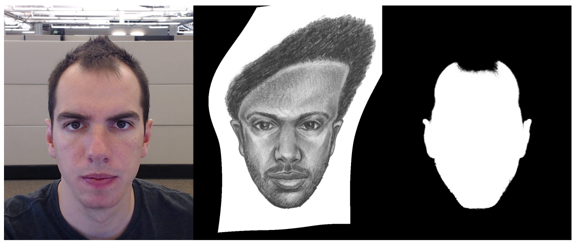


Figure 4.4: Visualisation of bad deformation caused by incorrectly placed forehead landmark.

Both these effects and many more are the results of the transfer of global intensity levels. The other effect that can be seen in the comparison is a change of local contrast values. These changes can be found in the eyes of the appearance channel, where the contrast between the sclera and cornea is higher, and reflections near the pupil are more visible. As the algorithm producing this channel does not have any adjustable variables that would modify the result, we cannot change the algorithm's outcome. The constants set in the implementation chapter 3 are recommended and set by authors of the algorithm, therefore, changing them would not lead to better results.

4.4 Results of the synthesis

Results of the synthesis using our guide channels can be seen in Figure 4.6. The choice of results was made to show the most common shortcomings of the implementation as well as good examples of synthesis. When looking at the first image, we can see that the output image synthesis went very well. One part of this image that might be improved is the dark parts above the eyes, which are present in the original image, but there are no dark parts above the eyes of the style example. The hair of the output does not seem perfect, but that

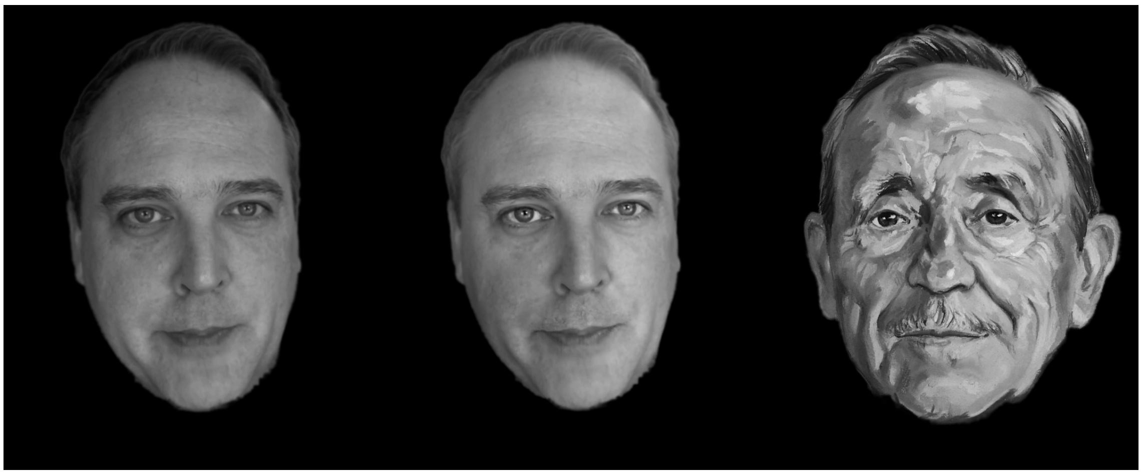


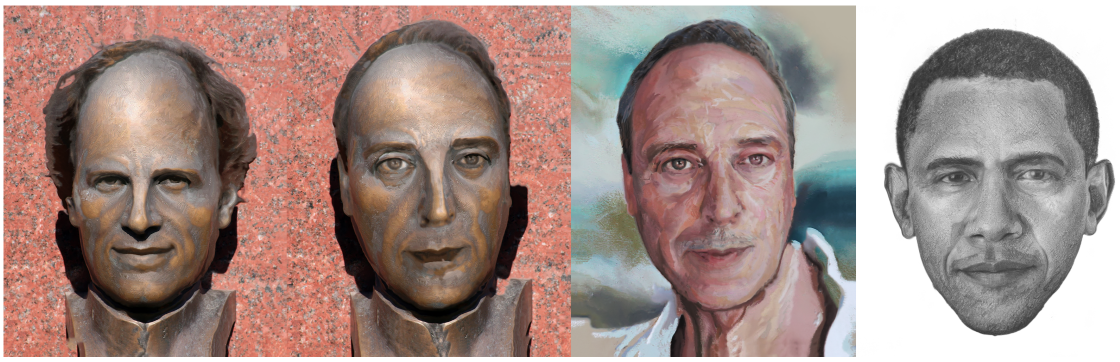
Figure 4.5: Visualization of changes that are made to the gray image of the original image to create its appearance guide channel. On the left is grayscaled image of the original image, in the center is the appearance guide channel of the original image and on the right is the grayscaled image of the style example.

is because, in the original image, there are individual hairs with different shades, which is not in the style example. In the second image, we can see how results are affected by using the second appearance channel instead of having a mask for weight. We can see the outline of the change of appearance channel, and the contrast between colors in those sections is visible. We can see that in both pictures the small structures from style example are preserved. The synthesis in the third image went well, but highly visible facial features, such as wrinkles in the style example, are not present in the result. Even though those features are not present in the original image, prominently painted wrinkles are part of the style used in the style example. The last synthesis also went very well, but we can see two minor flaws in the face. The first one is on the left of the left nostril, where we can see a shadow, which should not be present or at least this much visible. The second is the mustache, which seems like it does not fit. Although they are not significant deficiencies, finding a configuration of algorithms, which does not create them would be preferable.

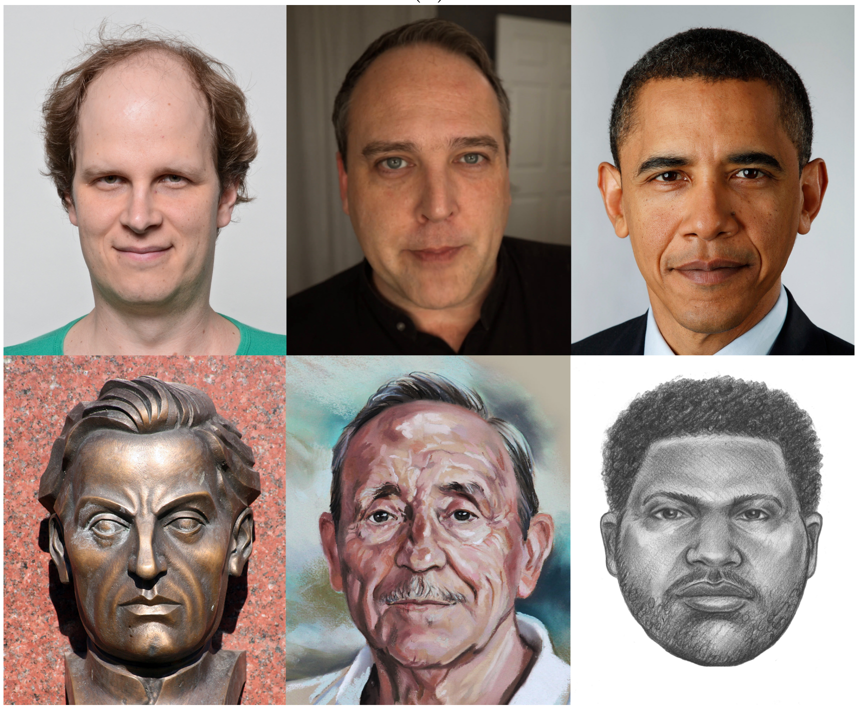
4.5 Performance results

The performance tests were done on a machine with following parameters:

- CPU: Intel i5-4670K @ 3.40GHz
- GPU: NVIDIA GeForce GTX 770
- RAM: 15.88 GB



(a)



(b)

Figure 4.6: a) Results of the synthesis. b) Original images in the top row and style examples in the bottom row.

- OS: Microsoft Windows 10
- Programming language: C++ 17
- Compiler: MSVC v142
- CUDA: v10.1

All performance data are present in Table 4.1. All present data were recorded on five different sets of images, each measured three times, and results were averaged with no significant outliner. The Visual studio profiler was used to capture these data. The whole process of loading input images, creating guide channels, and saving guide channels took 4045 ms. 35.89 % of that are input/output operations for images as well as transferring all the necessary data to GPU and afterward from GPU. If our algorithm was connected with the synthesis algorithm, we could have saved half of that time since we would not have to transfer results back to RAM and store them into a hard drive in the end. The time that was spent inside methods that create guide channels takes 64.10%. In this time are included all allocations and memory transfers that are needed for the algorithms. Most of this time took the appearance guide, which took 46.77% of the whole run, and the majority of the runtime of this algorithm is a convolution with Gauss kernel, an implementation of formula 3.17. The runtime of this part can be reduced by making the kernel smaller and thus forgetting parts of the array representing the kernel, which have minor effects on the final convolution. Using this method could get us time spent in this method down to 77% of its original runtime. This change can be experimented with by changing a constant in the code. Another suitable change could be to set a fixed size of the kernel array for each deviation of the Gauss, which could cause even more significant changes in time as well as changes to the final appearance guide. 15.30% of time was used on the segmentation guide. This time can also be altered by changing the number of iterations inside the Gauss-Seidel iterations. This change can modify the outcome of the algorithm, and the time spent in this method is not that significant, therefore, we kept it as it is. Only 1.81% were used on the positional guide, which has to calculate affine matrix for each pixel, and calculating a matrix is not dependant on the number of pixels and only has to process all landmarks. This process could be even more sped up by not calculating it for each pixel, but instead, we could find the matrix for each pixel in a specified grid and use interpolation to spread data across the whole image.

method	ms per call	N° calls	total time [ms]	% of whole run
Main	4045	1	4045	100
Guides	2593	1	2593	64.10
Appearance guide	1897	1	1897	46.77
Convolution with Gauss kernel	61	30	1830	45.21
Segmentation guide	310	2	620	15.30
Positional guide	38	2	76	1.81
io, memcpy	1452	1	1452	35.89

Table 4.1: Measured time data shown for different parts of the algorithm. For each part, time per call and number of calls is listed, as well as the total time and percentage of the whole run the step takes.

Conclusion

The main purpose of my thesis was to research style transfer methods, find a suitable method for the example-based stylization of portrait and implement the algorithm. As there are many implementations of the synthesis algorithms available, our main target was to build a fast implementation, which creates good guide channels for a synthesis algorithm. After completing the first versions of guide channel algorithms, we tried to implement the synthesis algorithm based on Fišer et al. method 3.4.1. Still, we could not make the algorithm perform well on given inputs. Therefore, we decided to redirect our focus to finalizing and polishing guide channel algorithms and using the available synthesis algorithm.

We implemented all three necessary guide channels with fast run times and good results. Even though we showed flaws in the results, we were able to point them out and find a method or a flaw in the algorithm, which caused these shortcomings. Using our implementations with a synthesis algorithm, whose primary focus is portrait style transfers, could improve run times and result quality, as described in the results section. We were able to run our algorithms on multiple sets of data without having a significant flaw in the result or without having one that could not be explained. Our implementation made it possible to alter results by changing constants within the algorithm. As all algorithms are well implemented, changes that would create better results should not have to change the implementation at all.

Bibliography

1. FIŠER, Jakub; JAMRIŠKA, Ondřej; SIMONS, David; SHECHTMAN, Eli; LU, Jingwan; ASENTE, Paul; LUKÁČ, Michal; SÝKORA, Daniel. Example-Based Synthesis of Stylized Facial Animations. *ACM Transactions on Graphics*. 2017, vol. 36, no. 4, pp. 1–11.
2. JAMRISKA, Ondrej. *Ebsynth: Fast Example-based Image Synthesis and Style Transfer* [`https://github.com/jamriska/ebsynth`]. GitHub, 2018.
3. HAEBERLI, Paul. Paint by Numbers: Abstract Image Representations. In: *Proceedings of the 17th Annual Conference on Computer Graphics and Interactive Techniques*. Dallas, TX, USA: Association for Computing Machinery, 1990, pp. 207–214. SIGGRAPH '90. ISBN 0897913442. Available from DOI: `10.1145/97879.97902`.
4. MEIER, Barbara J. Painterly Rendering for Animation. In: *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*. New York, NY, USA: Association for Computing Machinery, 1996, pp. 477–484. SIGGRAPH '96. ISBN 0897917464. Available from DOI: `10.1145/237170.237288`.
5. HAYS, James; ESSA, Irfan. Image and video based painterly animation. In: 2004, pp. 113–120. Available from DOI: `10.1145/987657.987676`.
6. BOUSSEAU, Adrien; KAPLAN, Matt; THOLLOT, Joëlle; SILLION, François X. Interactive Watercolor Rendering with Temporal Coherence and Abstraction. In: *Proceedings of the 4th International Symposium on Non-Photorealistic Animation and Rendering*. Annecy, France: Association for Computing Machinery, 2006, pp. 141–149. NPAR '06. ISBN 1595933573. Available from DOI: `10.1145/1124728.1124751`.
7. MEIER, Barbara J. Painterly Rendering for Animation. In: *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*. New York, NY, USA: Association for Computing Machinery, 1996, pp. 477–484. SIGGRAPH '96. ISBN 0897917464. Available from DOI: `10.1145/237170.237288`.

8. SELIM, Ahmed; ELGHARIB, Mohamed; DOYLE, Linda. Painting Style Transfer for Head Portraits Using Convolutional Neural Networks. *ACM Trans. Graph.* 2016, vol. 35, no. 4, pp. 1–18. ISSN 0730-0301. Available from DOI: `10.1145/2897824.2925968`.

9. GATYS, Leon A.; ECKER, Alexander S.; BETHGE, Matthias. A Neural Algorithm of Artistic Style. *CoRR*. 2015, vol. 16, no. 12, pp. 326–326. Available from arXiv: `1508.06576`.

10. FUTSCHIK, D.; CHAI, M.; CAO, C.; MA, C.; STOLIAR, A.; KOROLEV, S.; TULYAKOV, S.; KUČERA, M.; SÝKORA, D. Real-Time Patch-Based Stylization of Portraits Using Generative Adversarial Network. In: *Proceedings of the 8th ACM/Eurographics Expressive Symposium on Computational Aesthetics and Sketch Based Interfaces and Modeling and Non-Photorealistic Animation and Rendering*. Genoa, Italy: Eurographics Association, 2019, pp. 33–42. Expressive '19. Available from DOI: `10.2312/exp.20191074`.

11. ZHANG, Zhanpeng; LUO, Ping; LOY, Chen Change; TANG, Xiaoou. Facial Landmark Detection by Deep Multi-task Learning. In: FLEET, David; PAJDLA, Tomas; SCHIELE, Bernt; TUYTELAARS, Tinne (eds.). *Computer Vision – ECCV 2014*. Cham: Springer International Publishing, 2014, pp. 94–108. ISBN 978-3-319-10599-4.

12. SHEN, Xiaoyong; HERTZMANN, Aaron; JIA, Jiaya; PARIS, Sylvain; PRICE, Brian; SHECHTMAN, Eli; SACHS, Ian. Automatic Portrait Segmentation for Image Stylization. *Computer Graphics Forum*. 2016, vol. 35, pp. 93–102. Available from DOI: `10.1111/cgf.12814`.

13. ORZAN, Alexandrina; BOUSSEAU, Adrien; WINNEMÖLLER, Holger; BARLA, Pascal; THOLLOT, Joëlle; SALESIN, David. Diffusion Curves: A Vector Representation for Smooth-Shaded Images. In: *ACM SIGGRAPH 2008 Papers*. Los Angeles, California: Association for Computing Machinery, 2008, pp. 1–8. SIGGRAPH '08. ISBN 9781450301121. Available from DOI: `10.1145/1399504.1360691`.

14. SCHAEFER, Scott; MCPHAIL, Travis; WARREN, Joe. Image Deformation Using Moving Least Squares. *ACM Trans. Graph.* 2006, vol. 25, no. 3, pp. 533–540. ISSN 0730-0301. Available from DOI: `10.1145/1141911.1141920`.

15. SHIH, YiChang; PARIS, Sylvain; BARNES, Connelly; FREEMAN, William T.; DURAND, Frédo. Style Transfer for Headshot Portraits. *ACM Trans. Graph.* 2014, vol. 33, no. 4, pp. 1–14. ISSN 0730-0301. Available from DOI: `10.1145/2601097.2601137`.

16. KASPAR, Alexandre; NEUBERT, Boris; LISCHINSKI, Dani; PAULY, Mark; KOPF, Johannes. Self Tuning Texture Optimization. *Comput. Graph. Forum*. 2015, vol. 34, no. 2, pp. 349–359. ISSN 0167-7055. Available from DOI: `10.1111/cgf.12565`.

17. FIŠER, Jakub; JAMRIŠKA, Ondřej; LUKÁČ, Michal; SHECHTMAN, Eli; ASENTE, Paul; LU, Jingwan; SÝKORA, Daniel. StyLit: Illumination-Guided Example-Based Stylization of 3D Renderings. *ACM Transactions on Graphics*. 2016, vol. 35, no. 4, pp. 1–18.

18. ŽÁRA, Jiří; BENEŠ, Bedřich; SOCHOR, Jiří; FELKEL, Petr. *Moderní počítačová grafika, 2. vydání*. Computer press, 2005. ISBN 80-251-0454-0.

Attachments

A Contents of the enclosed CD

```
CD
├── Source codes ... Folder containing source files of the project.
├── README ... File containing information about compilation

│             requirements, program inputs and outputs.
├── document.pdf ... Pdf containing this document.
└── Results ... Folder containing a few results of used

                algorithms.
```

B Additional Images

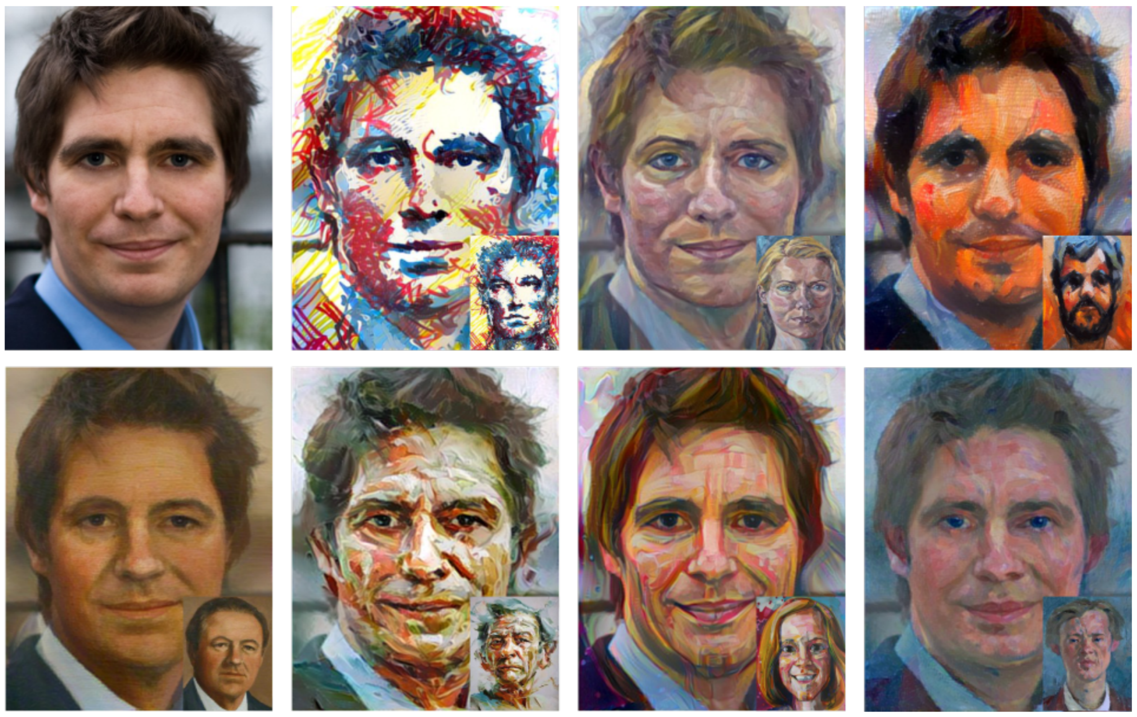


Figure 7: Example of the portrait style transfer using the Selim et al. algorithm. Source [8].

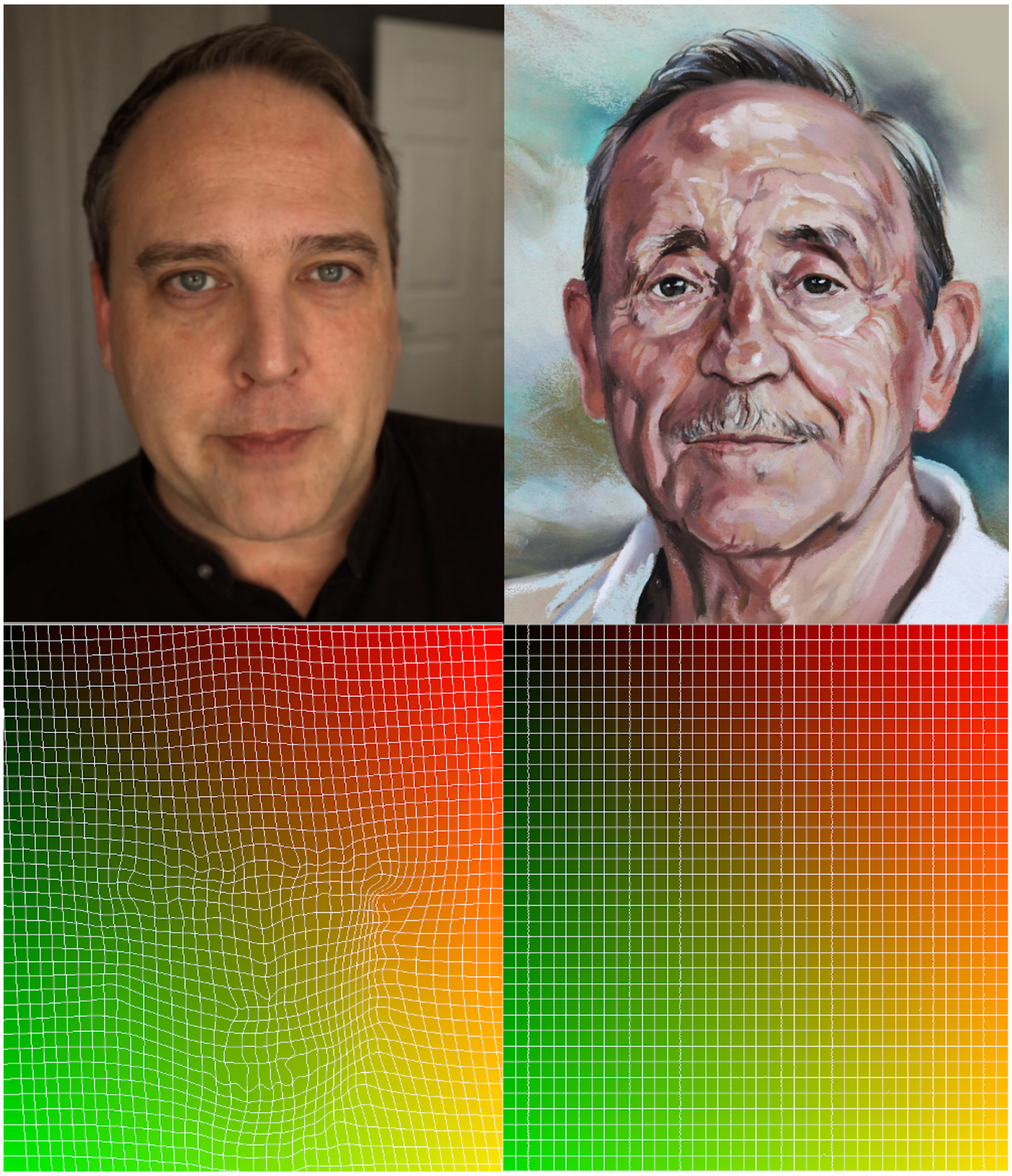


Figure 8: On top left and right, we can see original image and style example. The bottom two images are visualisations of their positional guides. Lines can help see the deformation in the positional guide of the original image. We can see that at the position of the right ear there is huge deformation, because the ear of style example is big and on the very right side of the image. In the guide we could also notice the rotation of the face as the person in the original image looks straight into the camera or a little to the right and the person in the style example looks a little to the left. Minor details like difference between lips and eyes can also be seen in the figure.

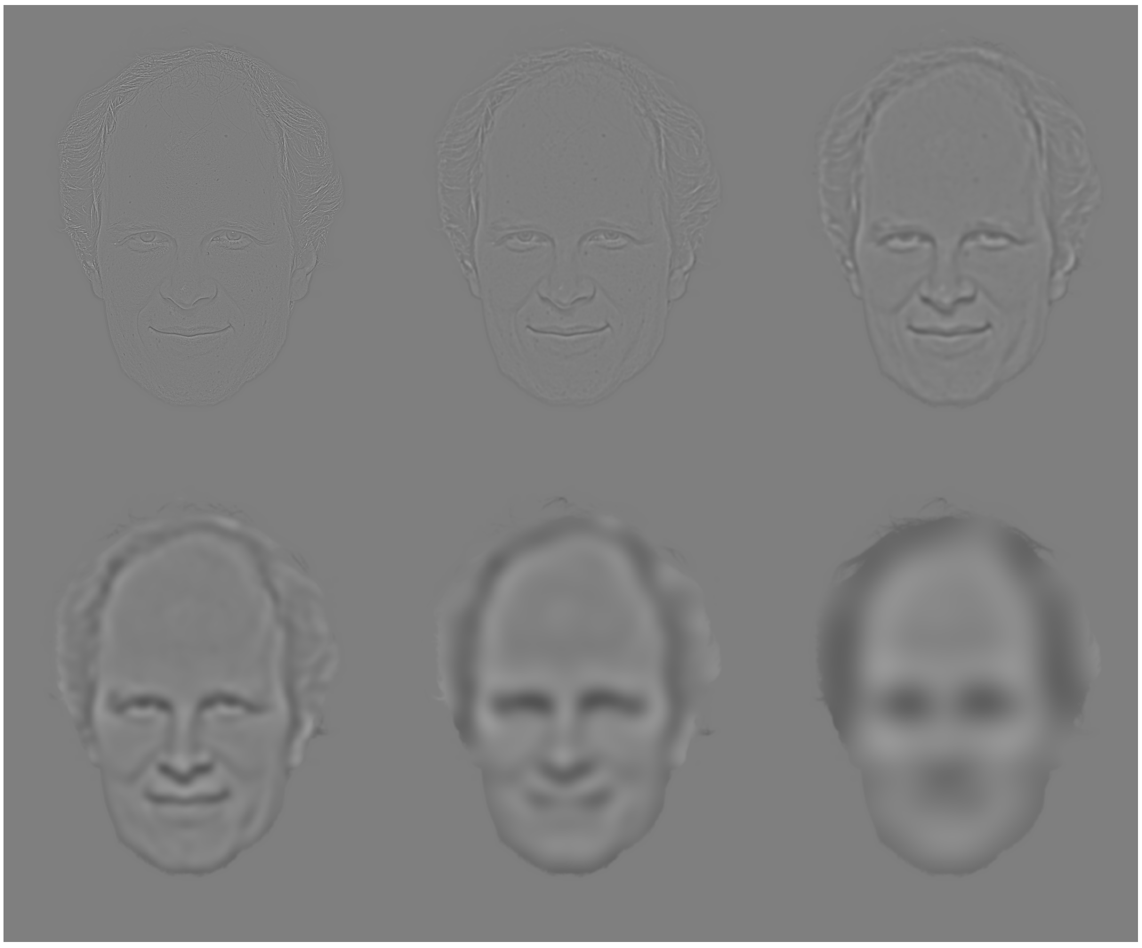


Figure 9: In this image we can see visualisation of decomposition of an original image. In the first image we can see only highlighted high frequency changes, such as hair lips or edges of eyes. As we move to deeper decompositions, low frequency details are seen.

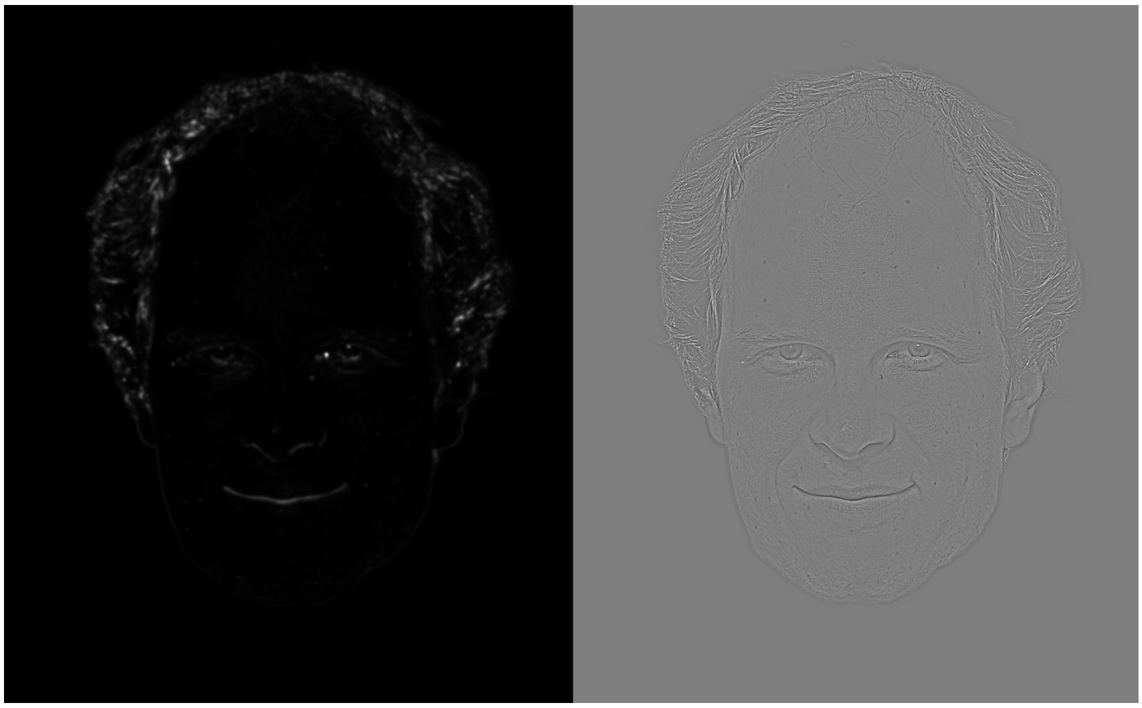


Figure 10: In this image we can see a comparison between a local energy map and the decomposition.

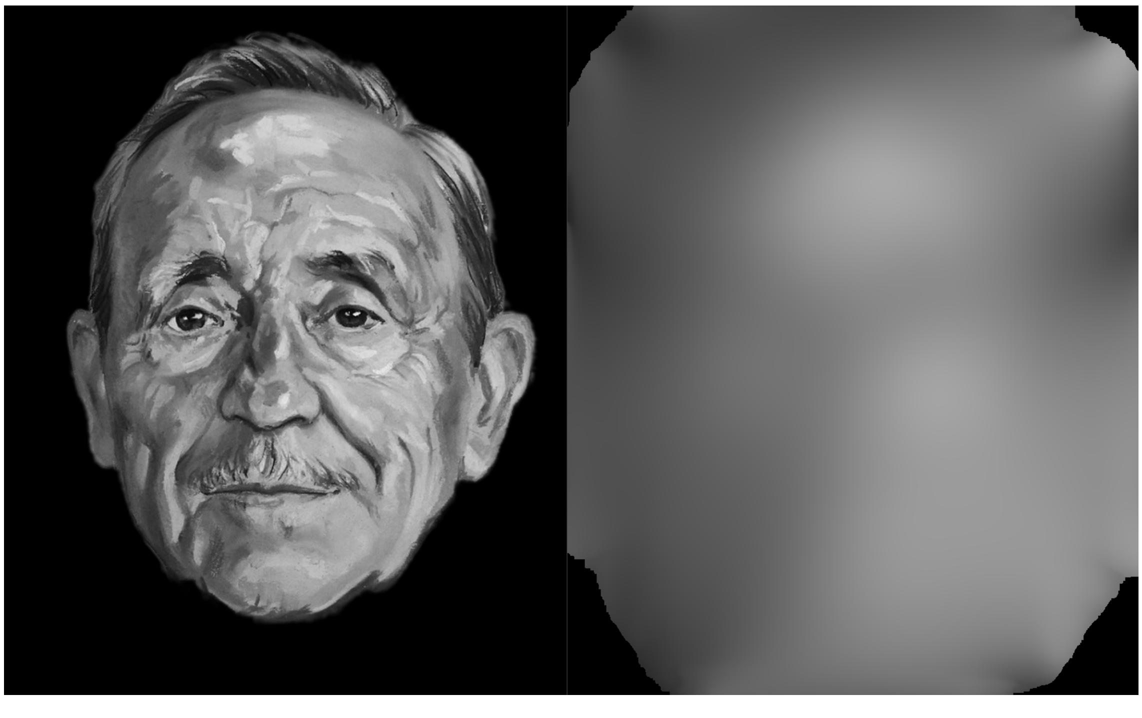


Figure 11: Comparison of a style example and its residual.