Masaryk University
Faculty of Informatics

Analyze and improve image filters in GNOME Photos

Bachelor's Thesis

Samuel Zachara

Brno, Spring 2020

Masaryk University
Faculty of Informatics

Analyze and improve image filters in GNOME Photos

Bachelor's Thesis

Samuel Zachara

Brno, Spring 2020

This is where a copy of the official signed thesis assignment and a copy of the Statement of an Author is located in the printed version of the document.

Declaration

Hereby I declare that this paper is my original authorial work, which I have worked out on my own. All sources, references, and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Samuel Zachara

Advisor: RNDr. Adam Rambousek, Ph.D.

Acknowledgements

I would like to thank Debarshi Ray for guiding me with all parts of the thesis and for his time helping me with the thesis. I would also like to thank RNDr. Adam Rambousek, Ph.D. for advising and supervising my thesis.

Abstract

This thesis is focused on reverse-engineering the Clarendon filter from popular social network Instagram and implementing it inside the Gnome's Photos application. At the beginning of the thesis, a detailed reconstruction of the filter is made. After that, a search for the most optimal implementation of the filter follows. The result of the thesis is the implementation of the filter algorithm inside the Gnome's Photos application.

Keywords

Image filter reconstruction, Clarendon filter, GNOME Photos

Contents

List of Tables

List of Figures

Context

We live in the age when it is easier than ever to connect with other people through so-called social networks. Many types of these media platforms are being developed due to ever-growing interest from its customers or users. While some social networking sites are used mainly for texting, sending emoticons, or small compressed images and videos, others make it possible to share your daily routines to all of your contacts. These are called stories and they are short videos available to view for everyone for 24 hours. One of the most famous networks that offer this type of functionality is called Instagram.

The original purpose and the idea of Instagram is, however, a little different. During the days when Instagram stories are not available yet, people are using this app for sharing their nicest photographs from their holidays and adventures. One of the breakthrough features of Instagram, however, is the fact you can use many different photo filters to enhance the look of your photo. These range from black and white filters, ones that offer washed-out colors to vivid and contrast heavy filters.

Somewhat similar photo filters can be found in Linux. More specifically inside desktop environment called Gnome. This desktop environment's native photo editing program is simply called Photos. Only a small set of filters to enhance the look of your image is available, and most of them are copies of Instagram filters. These copies are similar to their original counterparts and most of them are created with some filter color reproduction software. Such software is based on simple three-dimensional lookup tables which are represented by two-dimensional hald images. Hald image contains one particular color pattern. All colors are represented by this pattern. Colors that are not present in the hald image are calculated by approximation. The basic problem of this method is, that this procedure can only caption transformations used in photo filter, where no different techniques like vignetting, watermarks, scratches or gradients are used [1].

In the thesis, one of the Instagram filters is reverse-engineered as precisely and scientifically as possible. After this analysis, the best ways this filter can be implemented inside Gnome photos are explored and later implemented. For this purpose, Clarendon filter is chosen.

This is one of the most used Instagram filters, and subjectively one of my favorites. It is used mainly with sun-sets and colorful pictures, since it adds light to light areas (also called highlights) and darkens dark areas (also called shadows) [2]. This filter also introduces a slight color shift to colder tones. Another reason for this filter to to be chosen is the fact that it is a default filter in the Instagram app. For all of the reasons above, Gnome users are missing out on this filter.

One of the Instagram filters, specifically Hefe filter, is already implemented using similar techniques used in this. This work, done by Corey Hoard is the main inspiration for this thesis and helps with understanding all the necessary parts of reverse-engineering the filter [6]. However, many parts are just briefly described in this work, and a large part of the process is to be done in this thesis.

The key part of this thesis that follows analysis, is the part where some of the data generated from experiments are fitted with equations in numerical computing environment Matlab. This is one of the more challenging parts of the thesis since data generated from the analysis part is often complex and can only be represented by 3D graphs. For this purpose, very helpful seems to be Matlab documentation [12]. Another important source of information is the work Advanced surface fitting techniques. As stated in this article, "In most commercial packages the user has to specify various fitting parameters and test the results before accepting the final surface" [3]. This helps us not only understand the theory of fitting but also guides us to choose the correct parameters and variables when proceeding with a surface fit.

Introduction

At the time of writing this thesis, Instagram is one of the most popular social networking sites in the world. The majority of people have at least heard about this social network, and many of them use it regularly. As the study shows: "more than 95 million photographs are uploaded to Instagram every single day" [4]. Another research based on more than 40 million photographs uploaded to Instagram reveals that: "almost 20 percent of all the images are processed with one of the Instagram filters" [5]. Since Clarendon filter is the most popular Instagram filter and is used on almost 25 percent of all filtered images posted on this network, there is strong motivation to broaden the set of the filters available inside Gnome Photos.

The intention with this thesis is not only to give people a chance to use the Clarendon filter outside of Instagram, but also to show the process of reverse engineering the photo filter to make a starting point for others to continue with other filters. This second purpose is very important because this could resolve in a wider selection of filters inside the Gnome photos application and, therefore, more frequent use of filters on images imported from mobile phones and cameras. This could be the simplest way to enhance photographs without using complicated photo editing software, which sometimes can be time consuming and exhausting.

The first part of the thesis consists of the reverse engineering and studying the Clarendon filter. In this part, key discoveries are made which shape the final implementation of the filter. The second part of the thesis contains the creation of a mathematical representation of the filter. The next part is focused on the final implementation of the filter inside the Photos application. This implementation uses the C language. At the end of the thesis, the final comparison of the developed filter with the Clarendon filter is shown. A conclusion is made if it is possible to distinguish the two filters with a naked eye by an average human.

1 Reverse engineering the Clarendon filter

1.1 Introduction to the Clarendon filter

Before a detailed analysis of the filter, we need to understand the basic principles of the filter. To research the filter successfully, a first visual inspection of the filtered image is performed. For this purpose, the picture is transformed with the Clarendon filter and placed side by side with the original one to highlight the key techniques and the features of the filter (Figure 1.1).

After a close analysis with the naked eye, several things stand out. The first main difference is that colors in the right image seem to be more vibrant. The green color is more pronounced. Several more differences are noticeable, such as dark areas getting even darker in the right image and the light areas appear to be even brighter. This indicates the manipulation of the highlights and shadows by the filter. This resolves in a more vibrant image overall and increases the contrast of the image. The last thing that can be noticed from these two images is a slight blue color shift. To confirm the analysis from this image, the same inspection is performed with one more picture.

In another two images (Figure 1.2), there are very similar differences as in the set of images. In the bottom side of the image, even stronger blue color shift is visible. This indicates that red-colored pixels are transformed to contain a lot more blue color. This effect is most noticeable in darker parts of the image. All of the other differences are the same as with the first two pictures.

This filter analysis with the naked eye is an important step to understand the basic principles of the Clarendon filter. Based on this knowledge, it is possible to create several experiments to research the filter in more detail. These experiments work with precise data to eliminate possible errors caused by inspection with the naked eye.

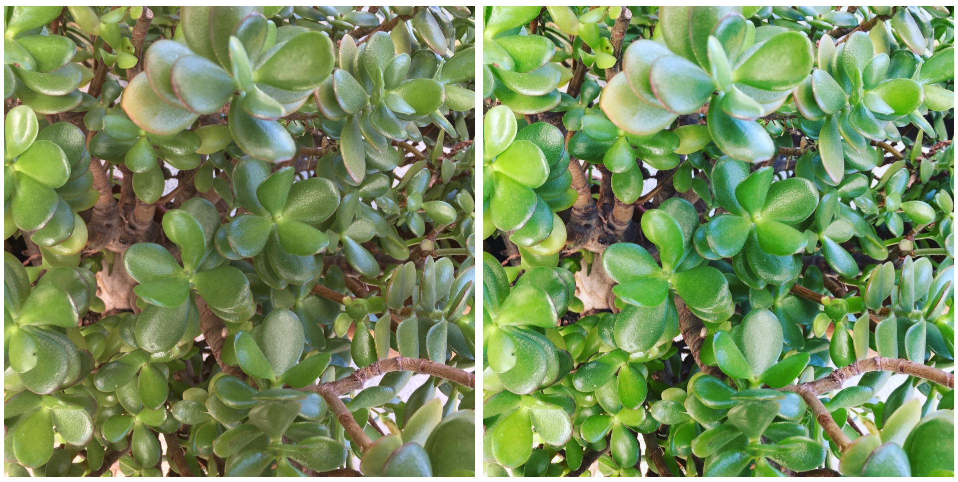


Figure 1.1: On the left side is original image without any processing. On the right side is the same image after being transformed with Clarendon filter.

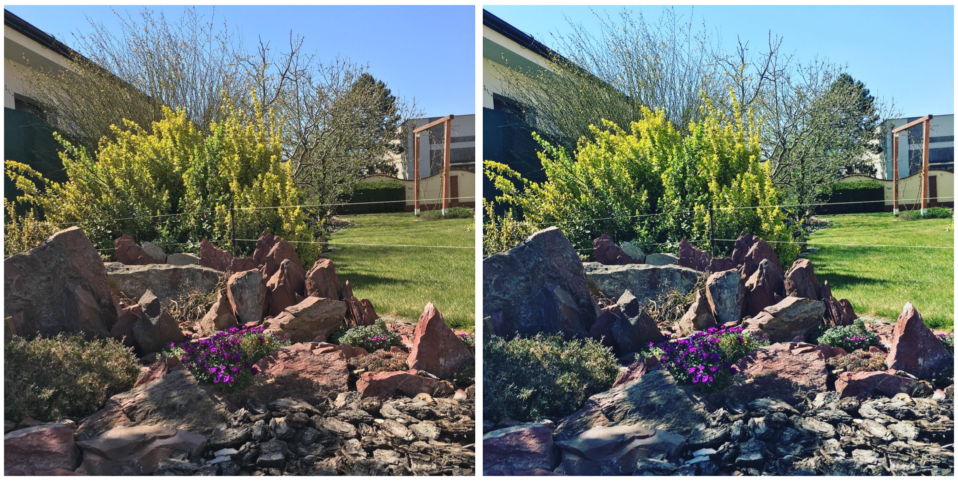


Figure 1.2: On the left side is original image without any processing. On the right side is the same image after being transformed with Clarendon filter.

1.2 Experiments

1.2.1 Introduction to experiments

This part of the thesis is designed to analyze the filter in more detail and divide it into smaller operations. Each experiment is designed to analyze one type of behavior of the filter. Every experiment consists of three parts. In the first part, called purpose, a general explanation of the importance of the experiment is given. This part is followed directly by the implementation of the experiment. All of the experiments are implemented in Python. Python is used for its ease-of-use. For manipulation with the images, Python's imaging library PIL is used. For generating graphs and visualizing data from the experiments, Python's plotting library Matplotlib is used. The last part of each experiment is an outcome. In this part, the general conclusion from the experiment is summed up.

The main purpose of these experiments is to be able to create an effective and precise algorithm to reproduce the Clarendon filter. This algorithm is based on a pixel by pixel transformation of the image to achieve the desired effect. Since many factors can increase the complexity of the algorithm, each experiment plays a role in minimizing the resources needed to perform this transformation.

There are several possible inputs that can play a role in transforming the input color of a pixel into the output color. One of these is the position of the pixel in the image. There are two types of operations that use the position of the pixel for calculating the output value of the pixel. The first group comprises pixel position-dependent operations, that create different output values for pixels with the same color in different positions of the image. The second group is formed by inter-pixel dependent operations, whose output value for a single pixel depends on the values of the neighboring pixels. The most important input, however, is the color of the pixel itself.

This means that currently to a create a correct output for one input pixel from the image a function with the these inputs is needed: the position of the pixel, the colors of other pixels in the image, and the color of the pixel itself. It would be very complex and difficult to create such a function. In the following experiments, the main goal is to eliminate as many inputs as possible to make the process of creating this function much simpler.

Each of the experiments uses images for the research. For this purpose, an image data set is generated with a simple Python script [Script 1]. This script is based on the script used for generating a similar image set in the work Reverse engineering Instagram's Hefe filter by Corey Hoard [6]. These images are designed to highlight the specific properties of the image filter. All of these images are processed with the Clarendon filter,

1.2.2 Detecting color space

Purpose

There are more ways of representing a color in the image. To be able to display different specters of visible colors various color spaces are used. Color space, also known as the color model, is an abstract mathematical model which simply describes the range of colors as tuples of numbers [7]. Many different color models are currently used for color representation. One of the most popular is the RGB color model.

The RGB color model uses three values to represent color. As the name suggests, the values are red, green, and blue. There are more standards of RGB color model. For our testing, the sRGB color space is used. The reason is that sRGB uses three 8 bit values, totaling 24 bit. It means that with sRGB we can display more than 16 million colors. This is enough for our purpose of reproducing the filter.

An alternative representation of color to the RGB model is the HSV model. Letter H stands for hue, S for saturation, and V for value. The main advantage of this color model is the intuitive changing from one color to another by manipulating the saturation or the hue.

From the visual analysis of the filter, it seems that some kind of tone mapping is present. This tone mapping has to be reproduced in the final filter algorithm. The reproduction of the filter can be done using the separation of the color into individual channels from some color model. For ease of use and the simple 3 times 8-bit structure, the sRGB color space is tested first to see the possibility of using it in the transformation.

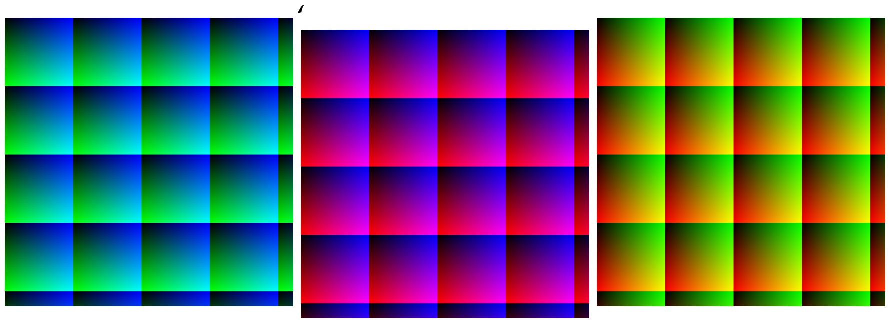


Figure 1.3: These are three images, each separating two out of three RGB channels into color plane

Implementation

The transfer function of the Clarendon filter is plotted for individual RGB channels in the following way. Pixels with values ranging from 0 to 255 for every RGB channel are taken from images in Figure 1.3. These values are mapped to an x-axis of the graph in Figure 1.4. The same pixels are taken from filtered pairs of these images and values of each channel are mapped to the y-axis of the same graph [Script 2]. In this way, the transformation function is visible and can be analyzed. Please note that this transformation function plot, does not represent the whole Clarendon filter since there can be many more factors that shape the function, as described in the introduction to the experiments. This plot is used only to get the general idea of the way it can look, which provides useful information for this experiment.

Mappings of individual channels are shaped into a regular curve with small deviations of the values from the general shape of the function. This means that creating a function to reproduce a similar mapping with our custom filter algorithm would be reasonably easy. Due to this discovery, the RGB color model is chosen for the reproduction of the filter. For this reason, no other color space needs to be tested. In all remaining experiments, the RGB color model is used for testing purposes. Separating color from individual pixels into sRGB channels seems like the best option since sixteen million colors is plenty to create a trustworthy reproduction of the filter while being space and performance efficient.

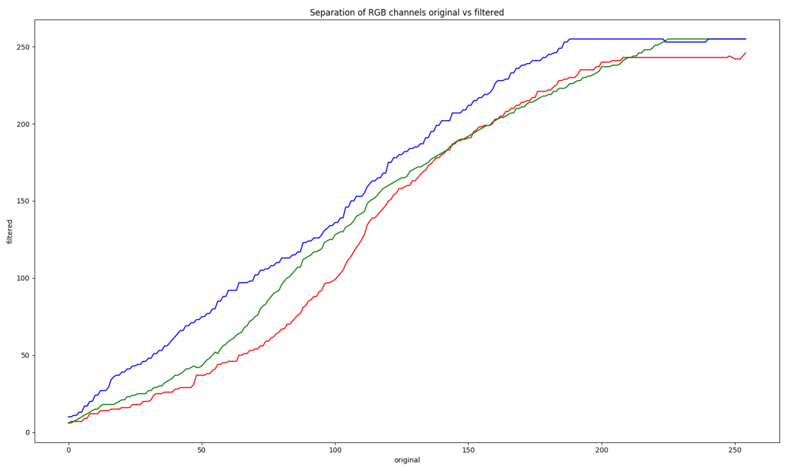


Figure 1.4: Separation of RGB channels

Outcome

It is apparent from this experiment, that the Clarendon filter applies some type of tone mapping using the RGB color model. This discovery is very useful for future experiments and analyses of the filter. Now, the general idea is that tone mapping present in the filter, will be reproduced using the RGB color model.

1.2.3 Detecting pixel position dependency

Purpose

In some of the filters from the Instagram filter library, pixel position-dependent operations can be found. One example of such a filter is the Hefe filter [6]. This means that there is a possibility that the Clarendon filter uses some type of pixel position-dependent operation, which is the reason why further analysis needs to be made.

A pixel position-dependent operation is in our case any type of image overlay. By image overlay, we can understand any operation that transforms pixels throughout the image differently with or without a regular pattern. For instance, vignetting can be considered as an image overlay. Vignetting creates a radial darkening in the corners of the image. As stated on the website of popular film cameras: "vignetting is an imaging phenomenon that happens with virtually every optical system" [8]. This phenomenon is so common in lenses with a wide aperture, that it is often considered a desired effect in the picture. This is the reason why vignetting is very popular in image processing in general, and thus used in some of the filters from Instagram.

By image overlay, we can also understand adding some type of artificial noise, scratches, or objects in front of the image. These techniques are often used to replicate noise in the picture, mimicking natural noise from film cameras. Despite often being considered an imperfection, many filters add noise artificially to replicate the unmistakable look from a film camera. When adding scratches and imperfections in front of the image, the vintage look can be achieved even with photographs taken with a smartphone camera.

The main goal of this experiment is to detect whether the Clarendon filter uses any type of overlay image over original images. It helps me to prove or disprove the presence of pixel position-dependent operations. If some type of image overlay is present when creating the filter algorithm, this overlay will be reproduced and applied over the image. If it is proven not to contain an image overlay, this type of operation can be eliminated from the final algorithm, making it one layer more simple.

Implementation

To separate one input (pixel position) from the other two inputs (color of the pixel, color and position of other pixels) simple images containing only one color are used in this experiment. For this purpose, 5 solid colored images are used. These images are processed with the Clarendon filter (Figure 1.5). If the image overlay is present in the filter, filtered images will not look uniform in color and at least in some of these images, some type of regular or irregular pattern would be visible.

After visual inspection of all these images transformed with the Clarendon filter, it seems that images are uniform in color, and with the naked eye no image overlay can be seen. To eliminate errors caused by the imperfection of the naked eye, a small Python script [Script 3]

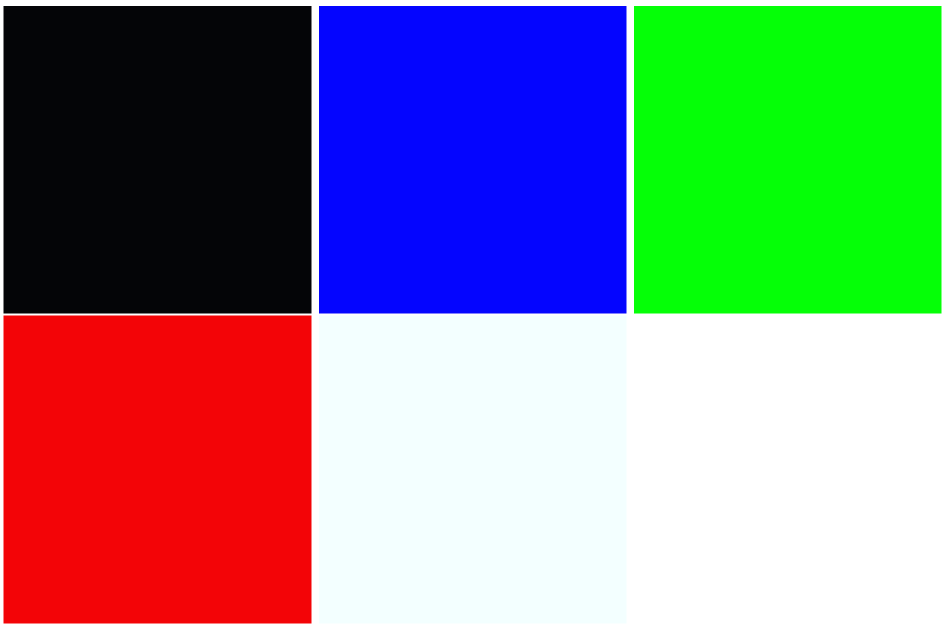


Figure 1.5: Images with solid colors used for detecting the image overlay

is used to determine the uniformity of the filtered images. This script takes each pixel from the image and compares its value to the other pixels. If all the pixels are the same, True is returned. This script is run with all 5 images.

Outcome

After running the script with all 5 images, the result is that all the pixels in each image are the same. This is a clear proof that the Clarendon filter does not use any type of image overlay. It means that the final filter algorithm will not include one extra step needed to reproduce image overlay. This helps with increasing the speed of the algorithm and decreasing the complexity of the filter.

1.2.4 Detecting inter-pixel dependency

Purpose

When manipulating the image, many types of inter-pixel dependent operations can be used. For testing purposes, these operations are divided into two groups.

- All operations that use information about neighboring pixels to make the color of the input pixel more similar to them belong to the first group of inter-pixel dependent operations. Some of the operations which belong to this group are blurring or decreasing clarity.
- Another group contains those inter-pixel dependent operations which use data about other pixels to make the color of the input pixel more different from its neighbors. This group comprises operations like sharpening and clarity increasing.

Since all of these inter-pixel dependent operations need extra inputs for transforming each pixel, they greatly increase the complexity of the filter. If the result seems to be, that some of these operations are present, they need to be replicated in the final algorithm. Implementation of such operations, however, is not a simple task, and the best approach would be to use some well-known algorithms. For example,

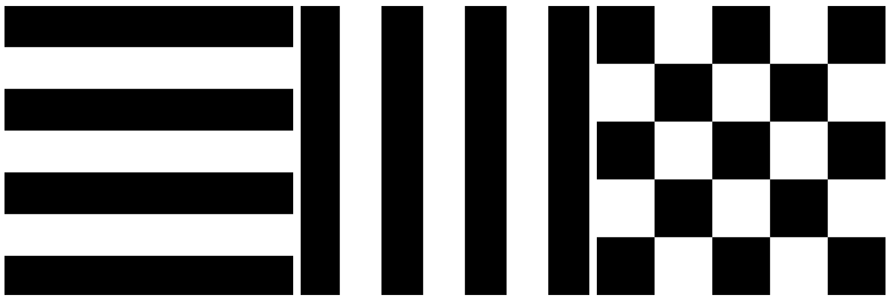


Figure 1.6: These are close ups of three patterns used in the six images

there are well known and widely used algorithms for blurring and other operations, as well. However, if the filter turns out to be inter-pixel independent, it would mean that the only remaining factor that affects the output value of a single pixel would be the color value of the pixel itself.

Implementation of detecting the first group of operations

In this part of the experiment, six images that consist of black and white pixels ordered in different patterns are used for the research. The main reason for this is that if some type of blur or similar operation is present, its effect can be seen best in images with strong contrast patterns. Three pairs of different patterns are present in these six images (Figure 1.6).

- Horizontal lines pattern.
- Vertical lines pattern.
- Alternating pattern.

From these six images, there are three pairs of complementary images which use the same pattern with swapped black and white colors. The main idea is to separate black and white pixels from these six images and compare them to the pixels with the same position from images with solid black and solid white color. From the six images, there are three pairs of complementary images, which use the same pattern with swapped black and white colors. This makes possible combining black pixels from one image with black pixels from the second image and create a new image consisting of only black pixels. The same is done for white pixels for all three pairs. In this way, six new images are generated. Three of them consist of black pixels and the other three of white pixels.[Script 4]

However, since all of these images are processed with the Clarendon filter, not all of the black or white pixels are equal. Now, each of the six images are compared with solid black and solid white images, which were also filtered with the Clarendon filter. In theory, if no blur or similar inter-pixel operation is present in the Clarendon filter, images should be the same. However, that is not the case this time. For this reason, the difference between the pixels in the compared images separated by RGB channels is counted [Script 5].

```
def count_error(pixel1, pixel2):
    (r1, g1, b1) = pixel1
    (r2, g2, b2) = pixel2
    error_red = ((abs(r1 - r2)/255) * 100)
    error_green = ((abs(g1 - g2)/255) * 100)
    error_blue = ((abs(b1 - b2)/255) * 100)
    return error_red, error_green, error_blue
```

In the code above, one function can be seen. This is the function used for comparing two pixels from two images. It takes two pixels as input parameters. At first, individual RGB color channels are extracted from the pixels and named accordingly. Then an error for each color channel is counted. This error is represented with a percentage. Specifically, the percentage of the difference between the RGB values of pixel1 compared to pixel2. 100 percent change would be value changing from 0 to 255. All of the errors are returned and used for future purposes.

Using this function, the average and maximum errors are counted for six images. Errors are counted for each RGB color channel separately. The average error is an arithmetic mean of all the errors. The maximum error is the maximum difference found in two corresponding pixels.

As results show (Table 1.1), the average error for black and white images circulates around 1 percent. This number is very insignificant

Table 1.1: Errors counted from images

Image	Channel	Combined black pixels		Combined white pixels	
		Average	Maximum	Average	Maximum
	Red	0.7843	1.1765	1.1765	1.5686
Vertical stripes	Green	0.7843	1.1765	1.5686	1.9608
	Blue	1.1765	1.5686	0.3922	0.7843
	Red	0.7843	1.1765	1.1765	1.1765
Horizontal stripes	Green	0.7843	1.1765	1.5686	1.5686
	Blue	1.1765	1.5686	0.3922	0.3922
	Red	1.1520	1.9608	1.7157	3.1373
Alternating pixels	Green	0.9804	3.1373	1.2255	3.9216
	Blue	1.0294	3.5294	0.5147	2.7451

at first sight. However, maximum errors in some cases reach 2 to 3 percent. This may indicate that some type of blurring kernel could be present. To understand these irregularities, a visual representation of the errors is needed.

For this purpose, the error is represented by a color. Since the errors range from zero to around three percent, it is represented in the following way. The zero percent error is represented by a blue color and three percent error, or more, by a red color. The error in between these two numbers is represented by the appropriate blend of these two colors. This creates a spectrum of colors, that can be differentiated with the naked eye. [Script 6]

For each of the six images compared previously, four error images are generated, totaling 24 images [Script 6]. Now, these images are analyzed. To see a three regularly repeating patterns in error images, three images are chosen. Since patterns are very dense and hard to see without a close look, images are scaled and cropped.

All of the error images show one of these three regular error patterns (Figure 1.7). All of these patterns are very similar to the patterns from the original images. The main difference is, that now the pattern is regularly repeating every 8 pixels. In the first of the three images, we can see a regular 8 by 8-pixel pattern. This does not look like a result of some blurring kernel. Another factor, that can create these

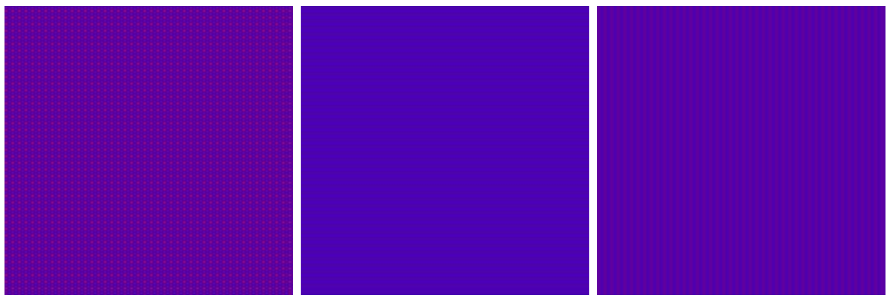


Figure 1.7: Three repeating patterns visible across all of the error images

patterns is a JPEG compression algorithm, used by Instagram after the filter is applied.

JPEG compression is a very common compression method used on the images. Since it is a lossy compression, some of the data from the image are lost. This compression algorithm consists of 5 main stages [9].

- RGB color space to YCbCr color space conversion
- Pre processing for DCT transformation
- DCT Transformation
- Co-efficient Quantization
- Lossless Encoding

Out of these five stages of the JPEG compression algorithm, pre-processing for DCT transformation, where DCT stands for a discrete cosine transformation, is the most significant to my error images. In this step of an algorithm, the image is divided into 8 by 8 pixel macroblocks. After this stage during the discrete cosine transformation similar artifacts as seen in the error images can be introduced.

The other argument, that the regular pattern is a result of JPEG compression, is that an error around one to two percent between two images is not noticeable with the naked eye. It means that it would not make sense to use the blurring kernel in the filter which is invisible to the naked eye.

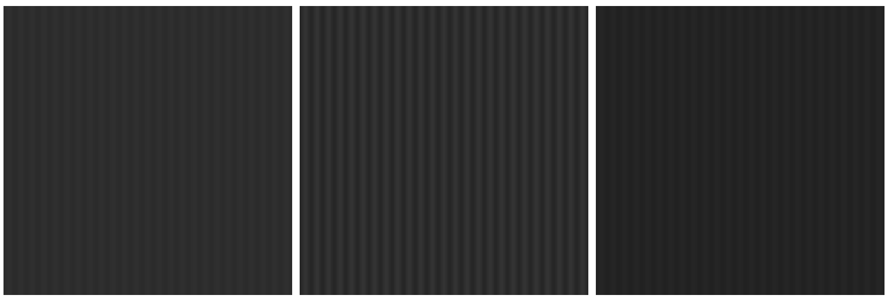


Figure 1.8: Three cropped images used for sharpening check. Left image is an original image in black and white. Middle one is sharpened image, and the right one is the image after being applied Clarendon filter and converted to black an white

Implementation of detecting the second group of operations

For detecting this group of operations, one of the previously generated error images is used. First, this image is converted into monochrome colors. This new monochrome image is processed in two ways. One image is sharpened with Gnome Photos software. Since Gnome Photos does not have an exact representation of sharpening, the slider is used for this purpose. The slider is set to about 25 percent. This is equivalent of sharpening, that can be done on regular photograph without introducing many side effects, like grain and noise. Then the image is saved. The second one is processed with the Clarendon filter and converted into monochrome. Now, these three images are visually compared (Figure 1.8).

When analyzing the three images (Figure 1.8), transitions between horizontal lines are most important to this experiment. The original image has very soft and subtle transitions when compared to the sharpened image. In the sharpened image, individual lines are more distinguishable. The difference between these two images is used as an example of how sharpening affects our image filtered with the Clarendon filter. When looking at this image, transitions between the lines seem very similar to the original image. This indicates that no sharpening was used on this image.

After visual analysis of the images, the assumption is made that the filter does not use sharpening. To prove it, transitions between lines in the previous three images are visualized. Since all three images are monochrome, the vlues of individual RGB channels for each pixel are the same. Visual representation is done by a graph, where the original value of one of the RGB channels is mapped onto the y-axis and the filtered value of the same channel is mapped onto the x-axis.

In the graph (Figure 1.9), three 16 pixel long horizontal lines of pixels from three images are visualized. Multiple graphs are created with different pixel positions in the images to ensure uniformity across the whole image. Graphs from different parts of the images all share the same results. The curve for the sharpened image is different from the other two. On the other hand, curves for the original image and the image processed with the Clarendon filter are very similar. The only difference between them is a slight shift on the y-axis. This is the result of the color manipulation of the Clarendon filter. Since both of these curves are very similar, we can say that the Clarendon filter does not use any sharpening algorithm.

Outcome

After completing both parts of the experiment, a conclusion can be made. In the first part of the experiment, the fact that the Clarendon filter does not use any type of blurring filter, nor any similar operation, was proven. In the second part, the non-presence of other inter-pixel dependent operations like sharpening inside the Clarendon filter is proven. After all of this testing, it can be said that the Clarendon filter does not use any type of inter-pixel dependent operations. This means that one more factor that could affect the output value of the final algorithm is eliminated. Since inter-pixel operations are last operations to require an input of other pixels, our algorithm for transforming input pixels into output pixels does not need any extra inputs other than RGB channel values of the one input pixels itself.

1.2.5 Detecting inter-channel dependency

Purpose

In the first experiment, the use of the RGB color model for the filter reproduction was proven to be a good option. After discoveries made in other experiments, it was proven that creating a tone mapping

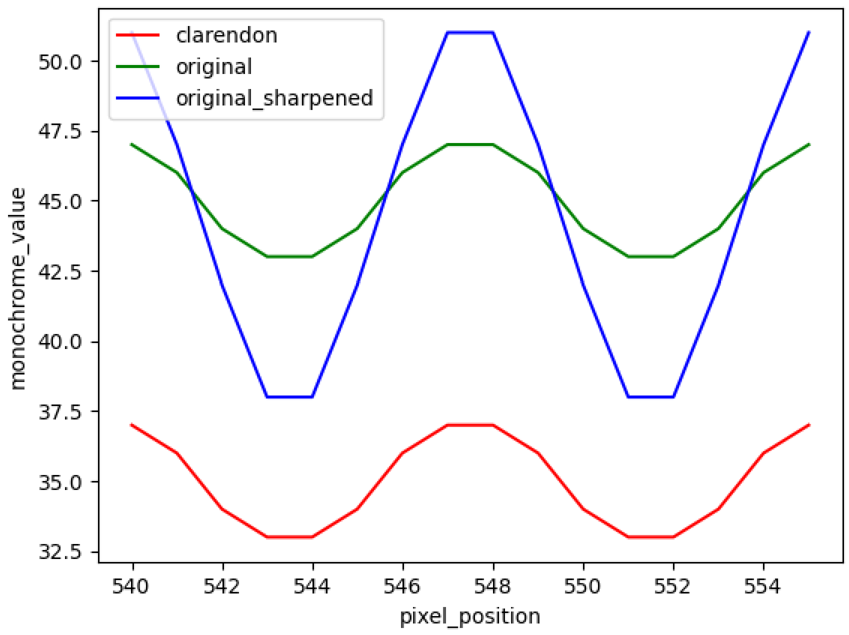


Figure 1.9: Curves generated from three images

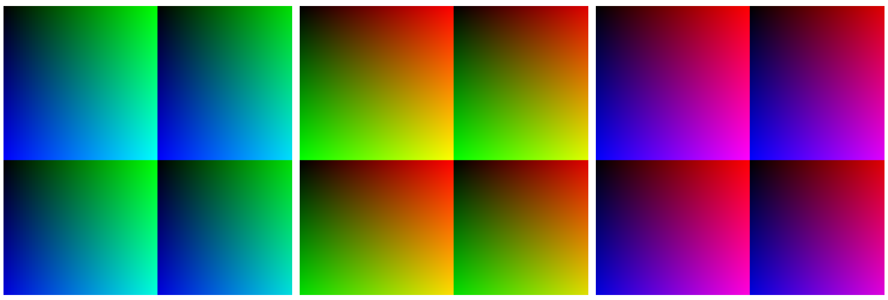


Figure 1.10: Three images, that each separate two RGB channels into color plane, to display all of the possible combinations with every two channels

is enough to reproduce the filter. The main question now remains what type of tone mapping can be applied to the image to mimic the Clarendon filter.

Since we are working with the sRGB color space, there are more than 16 million colors the image pixels can have. It would be almost impossible to create an algorithm with this many possible values that is effective and accurate enough for our purpose. One way how to simplify the algorithm is to prove that individual color channels are independent of each other. Even eliminating the dependency of one channel on another channel can help to create a better environment for creating an efficient filter algorithm.

For this purpose, images each separating two out of three RGB channels (Figure 1.10) can be used to unveil the relationship between the channels. From these images, it is possible to see what effect other channels have on each of the RGB channels.

Implementation

From the three images (Figure 1.10), it is possible to compare the input and output values of each channel for every possible value of the other two channels. In this way, dependency graphs can be created which visualize tone mapping functions for each channel with changing values of the other two channels. In a way similar to the first experiment, values of channels from original images are mapped onto

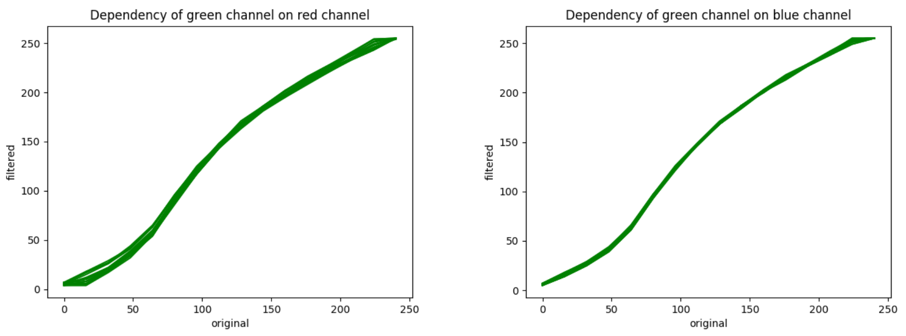


Figure 1.11: Graphs showing dependency of green channel on red and blue channels

the x-axis of the graphs and values of channels from filtered images onto the y-axis. Following these steps, six channel-dependency graphs are created (Figure 1.11, 1.12, 1.13) [Script 7].

From looking at the graphs showing the green channel dependency (Figure 1.11), the green channel seems to be independent on blue and red channels. Even though curves for different values of red and blue channels are not matching perfectly, this is the result of imperfections caused by compression used by Instagram. Since the differences between all of the curves are very small, it can be said that a green channel is only dependent on itself and no other channel. This means that the transformation equation for a green channel only needs an input of a green channel.

From the graphs showing the dependency of a red channel (Figure 1.12), it seems that a red channel is not dependent on a blue channel. On the other hand, the graph that shows the dependency of a red channel on a green channel has a wide range of different curves. This indicates that a red channel is dependent on a green channel. It means that for transforming an input value of a red channel to an output value, the equation needs to have inputs of red and green channels.

The last two graphs (Figure 1.13) show the dependency of a blue channel on green and red channels. It is apparent that a blue channel depends on a green channel. Another graph seems to indicate that a blue channel is independent on a red channel, excluding the very

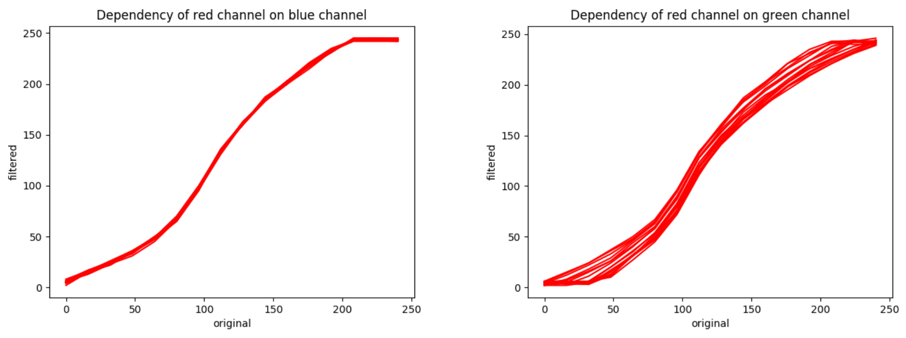


Figure 1.12: Graphs showing dependency of red channel on blue and green channels

bottom left (around first 40 values). However, variations in the curves are not significant and for this reason, a blue channel can be considered independent on a red channel. If the final filter turns out to be inaccurate because of these variations, it will have to be considered and reproduced with the algorithm.

Outcome

From this experiment, a few conclusions can be made. Not all individual RGB channels are independent of each other. Following dependencies were detected: a blue channel on a green channel and a red channel on a green channel. Other than these two dependencies, channels are only dependent on themselves. This is the final step in reverse-engineering the Clarendon filter. Now, we can proceed to develop the filter algorithm itself.

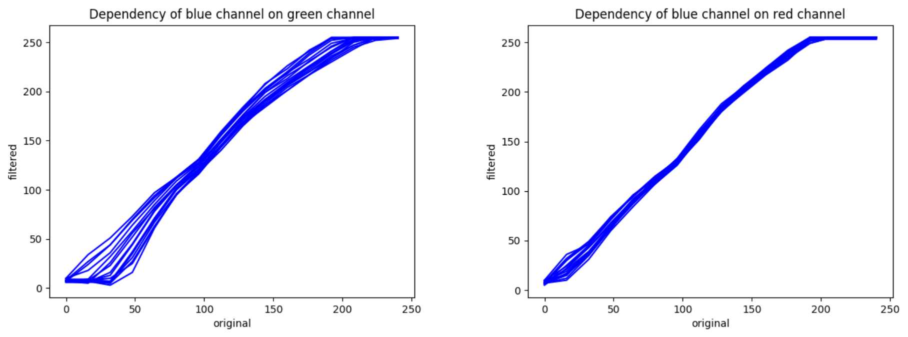


Figure 1.13: Graphs showing dependency of blue channel on red and green channels

2 Development of the filter algorithm

2.1 Introduction

From the previous experiments, we have enough data to create a reasonable algorithm that reproduces the Clarendon filter. Our algorithm takes one pixel from the image as an input. After that, the color of the pixel is separated into individual RGB channels. Each of these channels is filtered separately.

Now, the transformation of each input channel into an one output needs to be created. There are more ways of approaching this task. To make the usage of the filter convenient and fast, an individual approach is chosen for each channel. There are two methods of implementation, that are good candidates for the reproduction of the filter.

- Lookup tables
- Equation created by either polynomial or different fit

2.2 Algorithm using lookup tables

Using lookup tables is a technique very often used for color processing. This technique works in a way, that for each possible input value, the output value is pre-computed inside a data structure. This can resolve in unwieldy and very resource-heavy structures. In our case, let's take a blue channel for example. From the last experiment, it was proven to be dependent on a green channel and itself. This means that since we are working in sRGB color space, more than 65000 values need to be stored. There are a couple of techniques to decrease this number, such as pre-computing only certain values and using interpolation to compute values in between them.

Lookup tables are excellent for optimizing the evaluation of functions that are expensive to compute and inexpensive to cache [10]. This is unfortunately not our case with blue and red channels, since they require more than 65000 values inside the lookup table, if not using interpolation between the values. However, on the other hand, a green channel is only dependent on itself. This means that a simple lookup table with 256 values can be created. Thus, for red and blue channels lookup table is only a backup option, but for a green channel, it can be considered as one of the efficient ways of implementation.

To create this lookup table for a green channel, images used in detecting inter-channel dependency experiment are used (Figure 1.10). The goal is to create an array of 256 values. For every *n-th* position of this array, the output value for input with value *n* is saved. Data for this table are taken from two out of three images (Figure 1.10) since the third one does not contain pixels with values of a green channel other than zero. To eliminate possible inaccuracy, the output for each input value is calculated as the arithmetic mean on all filtered pixels from these two images with the same original value as our input. In practice, this means that for each pixel in both images with original green channel value, being for example 4, the output value is calculated as the arithmetic mean of the same pixels from the Clarendon filtered pairs of the images. The lookup table for a green channel is saved for future testing.

2.3 Algorithm using equations

The main advantage of using equations to transfer the input value into the output value is, that equations are not cache-heavy and if created correctly, they can also be fast and precise. Using the equations makes the most sense with red and blue channels since having output values of these channels is resource-heavy. From our three images (Figure 1.10), we have enough data to create lookup tables with every possible combination of any two channels. In this case, we are trying to represent this data not with the lookup table, but with the equation. This means, that to create the equation, we need to create a fit for our data.

Fitting curves and surfaces is a complex process. For this reason, the Matlab software is chosen to perform these fits.

Matlab is a programming platform designed specifically for engineers and scientists [11]. Inside this software, curve fitting toolbox can be found. This toolbox can create various types of fits on 2D or 3D data. To create our equations polynomial fitting is used. When

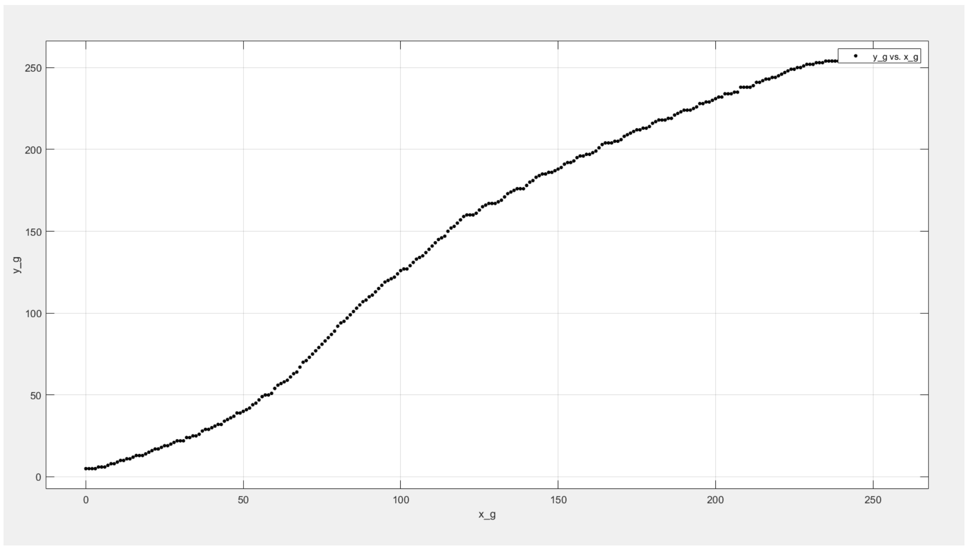


Figure 2.1: Plot of data points for the equation for a green channel

creating polynomial fits with the curve fitting toolbox, the polynomial degree of the equation can be adjusted with real-time results. Using this tool, different fits for each channel can be created and later tested for error and speed when compared to the original Clarendon filter.

Creating fit for the green channel is very simple to create as compared with blue and red channels. First, data from the previously created lookup table are imported to Matlab as two arrays. One array contains original values (values from 0 to 255) and another array contains values after Clarendon is applied. In the curve fitting tool, the first array of values is assigned to the x-axis of the graph and the second array is assigned to the y-axis. This results in a plot (Figure 2.1).

After selecting the type of fit to polynomial, several degrees of the equation are chosen to produce different fits. The residual plot of each fit is analyzed. The root mean squared errors are inspected. For several reasonable polynomials, these errors can be seem in table 2.1.

As we can see, the average residual value is greatly decreasing with polynomial fits with degrees from one to four. After that, the accuracy of polynomial fits is increasing slower. This means that since the fourth-degree polynomial fit has an arithmetic mean of residu-

Table 2.1: Root mean square error for different polynomial degrees of equation for fitting green channel curve

Degree	1	2	3	4	5	6	7	8
RMSE	14.2074	8.9676	5.5105	3.1180	2.6435	1.5472	1.1298	1.1313

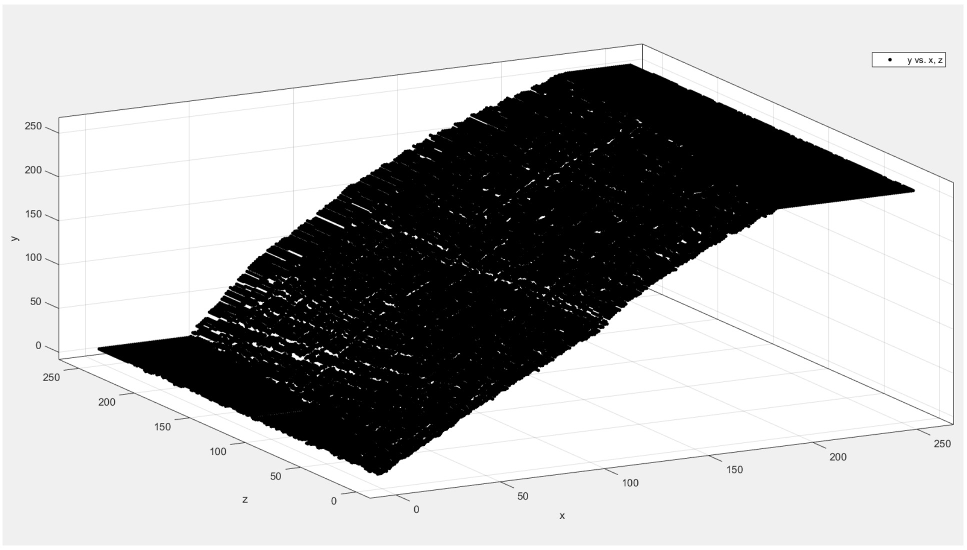


Figure 2.2: Plot of data points for the equation for a blue channel

als only about 3 (which is about one percent), it should be accurate enough to reproduce the green channel correctly.

Creating polynomial fits for blue and red channels is a little bit different from the green channel. In this case, the 3D surface needs to be fitted, since the output value of these channels is dependent on two values. The approach, is however, very similar. First, all of the possible values for each channel are imported to Matlab. Then, in a curve fitting tool for both red and blue channel fit, the z-axis is assigned desired output values. The original values of blue and red channels are assigned to the x-axis. Since both of them are dependent on the green channel, it is assigned to the y-axis. In this way, two 3D graphs are created (Figure 2.2, 2.3).

After experimenting with degrees of x and y variables, the following root mean square errors were found (table 2.2). For both blue

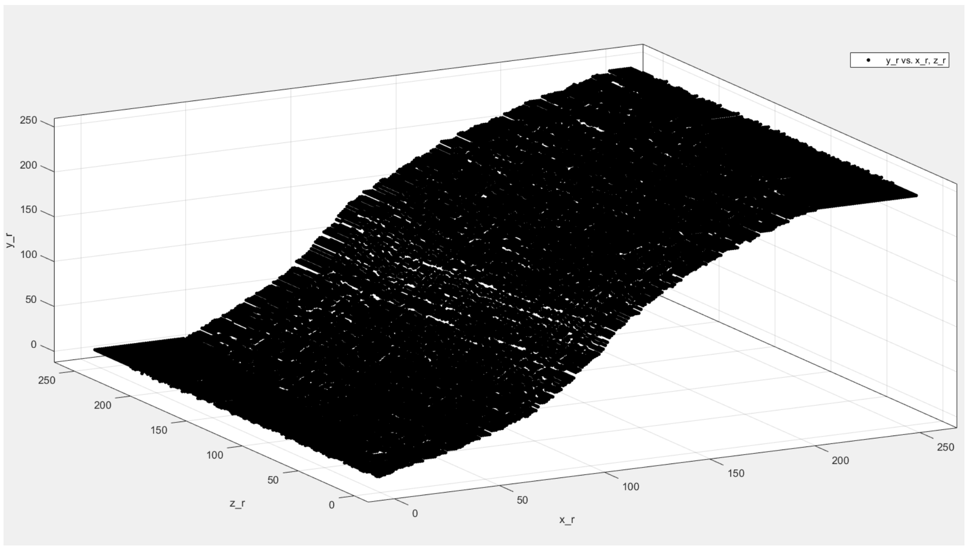


Figure 2.3: Plot of data points for the equation for a red channel

and red channels, the chosen degree of y is one. When increasing the degree of y, no significant improvements in accuracy are visible. It is very helpful to have degree of y set to 1, since increasing it would greatly increase the number of coefficients in total. A reasonable proportion of accuracy to complexity seems to have polynomial fits with the third and fourth degree of x variable for both red and blue channels. Equations from these four fits are saved and are used in future testing.

Table 2.2: Root mean square error for different polynomial degrees of equation for fitting blue and red channel surfaces

Degree of x	Degree of y	RMSE blue channel fit	RMSE red channel fit
1	1	17.9725	16.7423
1	2	17.7851	16.7294
2	1	10.4671	15.4491
2	2	10.4272	15.4355
3	1	5.7903	6.2367
3	2	5.628	6.1684
4	1	5.0810	4.5350
4	2	4.8832	4.4392
5	1	4.1453	3.9950
5	2	3.8999	3.8857

3 Implementing the algorithm inside Gnome Photos

3.1 Introduction to Gnome development

Gnome photos is an open-source photo management application for Linux. It can be used for many different purposes. It can be a beautiful photo viewing application with an intuitive user interface. Users can also choose to edit their photographs with a few built-in controls. Under the enhancement tab, some color filters can be found. Most of these are created with lookup tables, reproducing filters from Instagram. One of the filters is the Caap filter. This filter is a reproduction of the Hefe filter from Instagram. This work was done by Corey Hoard [6] who has already been mentioned a few times in my thesis. Currently, there are 6 filters to choose from. The goal is to add our version of the Clarendon filter to these filters.

All of these filters are implemented using the gegl C library. To ensure uniformity, the Clarendon filter is reproduced using the same library. The main principle of gegl are operations. Operations can be understood as a single action performed on the image. Some of the built-in operations of gegl are operations like load, save, saturation, contrast, etc. The goal is, to create a "Clarendon" operation, which applies our filter algorithm on each pixel of the image.

For the development of the application, a Gnome builder is used. This is a special environment for building gnome applications. It supports functionality like cloning a project from the git repository and creating and customizing files inside the repository. With this tool, it is easy to build the project with one click of a button. The application can be also easily exported as an image.

3.2 Implementation of the 4 algorithms

After cloning the gnome-photos repository [13] inside the Gnome builder, one new header file and one c file are created. These will contain the functionality of the new filter. The files are named *photos-operation-insta-clarendon*, following the naming scheme. Inside the c file, necessary objects and functions for the working gegl operation are created. The final filter algorithm will be implemented inside the function with the suffix "process".

Other files inside the same directory are modified to be able to work with a new operation. Build files are changed. The front end of the application is slightly improved, to be able to display a new filter. For now, after clicking on a new filter option, no action is done, since the filter is not implemented yet. Since the name of the filter should be renamed compared to the original name of the filter in the Instagram application, the name Trencin is chosen to represent this filter. Trencin is the name of a beautiful town in Slovakia with a picturesque scenery with a big castle and a historical old town center. Using the filter works great with this scenery.

The algorithm is implemented in four ways.

1. Green channel transformed with a lookup table, red and blue channels with the first type of equation
2. Green channel transformed with a lookup table, red and blue channels with the second type of equation
3. All channels transformed with equations, red and blue channels with the first type of equation
4. All channels transformed with equations, red and blue channels with the second type of equation

Lookup table for transforming green channel is saved to array inside the code. This lookup table was created inside filter algorithm chapter. Equations for red, green and blue channels following.

$$f_g(g) = 6.87 - (0.1453)g + (0.02435)g^2 - (1.355e-4)g^3 + (2.267e-7)g^4 \tag{3.1}$$

$$f_r(r,g) = 18.37 - (1.05)r - (0.0276)g + (0.03275)r^2 - (0.001056)rg - (0.000152)r^3 + (2.006e-6)r^2g + (2.091e-7)r^4 + (9.682e-9)r^3g \tag{3.2}$$

Table 3.1: Accuracy of individual algorithms separated by channels

Implementation	Red channel	Green channel	Blue channel	Combined
Algorithm 1	2.7696	1.8605	2.2544	2.2948
Algorithm 2	1.9795	1.8610	2.0963	1.9789
Algorithm 3	2.7094	1.9799	2.2654	2.3182
Algorithm 4	1.9773	1.9690	2.0933	2.0132

$$f_b(b,g) = 13.3 - (0.4149)b - (0.08369)g + (0.01699)b^2 - (0.001413)bg - (9.235e-5)b^3 + (1.239e-5)b^2g + (1.334e-7)b^4 + (2.221e-8)b^3g \tag{3.3}$$

The filter is implemented in these four ways. For each implementation, the same ten images are processed with the filter. 40 new filtered images are created after the testing. The testing analyzed two things.

First, an important cognition is that applying the filter on 24-megapixel images is very fast. There is no noticeable difference between equations with lower or higher polynomial degree. Also, the time difference between the lookup table and the equation is not noticeable. This means that time does not have to be taken into consideration with these implementations.

For testing the accuracy of each algorithm, all of the images are compared with the same images processed with the Clarendon filter inside Instagram. The results are displayed in the table (Table 3.1).

For creating the table (Table 3.1) same Python script for counting the error as in experiments was used [Script 5]. For each of the 10 images for every implementation mean error for each RGB channel is counted. Only mean errors are shown due to fact, that results were very consistent for all images.

3.3 Choosing one final algorithm

As we can see (Table 3.1), using a higher degree of equations helps improve the average accuracy of the algorithm by about one percent. This means that higher degree polynomial equations will be used in the final algorithm to make it as accurate as possible since no speed penalty is introduced. The difference between using the lookup table and the equation for transforming the green channel is very small. Using the equation is theoretically more resource-heavy on the processor. However, on the other hand, using the lookup table with static integer values inside array means that this information about the green curve is kept inside the cache memory. Since no real performance difference was apparent and in order to make the code look more uniform and simple, the green channel is chosen to be implemented using the equation.

This means that the final algorithm uses three separate equations for three RGB channels [Implementation]. The accuracy error as shown in the table (Table 3.1) is around two percent. When analyzing images filtered with Clarendon versus our filter, no difference can be spotted. However, to prove that the 2 percent difference is not visible to the naked eye, a small questionnaire is created.

4 Testing the visual accuracy of the naked eye

Our reproduction of the Clarendon filter seems quite accurate. However, the prove that the new filter is indistinguishable from the original Clarendon filter with the naked eye can be made. For this purpose, a questionnaire to research the capabilities of the naked eye is created.

Three images (Figure 4.1, 4.2, 4.3) are filtered with the Clarendon filter and our new filter. These are placed side by side in the survey. The respondents are expected to answer a simple question if the two images seem identical to them or not. To get the idea of the accuracy of the naked eye, the same three images are processed with two other algorithms.

These two algorithms are using equations fitted to the same data as our filter algorithm. The only difference is, that two new algorithms contain equations with different polynomial degrees. The first one is created using an equation with a first polynomial degree for all variables for all channels. Three equations are the following.

$$f_g(g) = 0.3595 + (1.142)g \tag{4.1}$$

$$f_r(r, g) = -9.369 + (1.181)r - (0.07837f)g \tag{4.2}$$

$$f_b(b, g) = 15.49 + (1.168)b - (0.0873)g \tag{4.3}$$

The other new algorithm is created using equations with these polynomial degrees. For the green channel, a second degree is chosen. For equation transforming a red channel, third degree of red variable, and the first degree of green variable is used. The blue equation uses the second and first degrees of blue and green variables. Three equations are the following.

$$f_g(g) = -23.93 + (1.715)g - (0.00225)g^2 \tag{4.4}$$

$$\begin{aligned} f_r(r, g) = 5.028 - (0.01739)r - (0.01967)g + (0.01482)r^2 - \\ (0.001433)rg - (4.411e - 5)r^3 + (5.709e - 6)r^2 g \end{aligned} \tag{4.5}$$

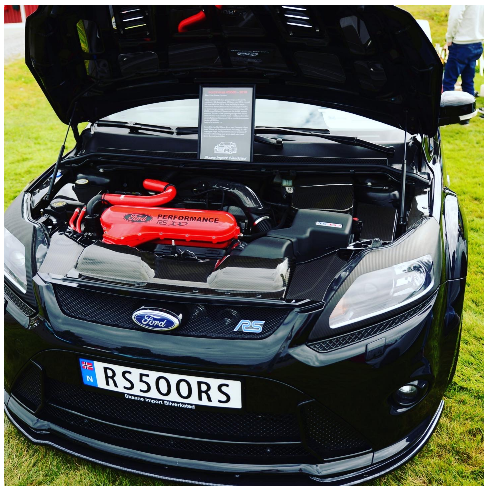


Figure 4.1: Image 1 used in the questionnaire

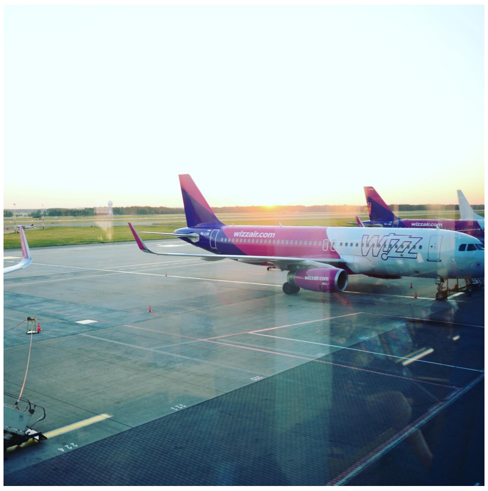


Figure 4.2: Image 2 used in the questionnaire

Figure 4.3: Image 3 used in the questionnaire

$$f_b(b,g) = -9.131 + (1.863f)b - (0.1438f)g - (0.00295)b^2 + (0.0004439)bg \tag{4.6}$$

Six images generated by filtering the original images with these two algorithms are also placed side by side with the images filtered with the Clarendon filter. In total, we have 9 questions then. Three more questions are added, each with two same images to see if the respondents can correctly detect these as the same images. For each filtered image from the questionnaire, the error percentage is counted in comparison with the Clarendon filter. These errors are put to the table (Table 4.1). As we can see, Three filter algorithms have slightly different average errors when compared with the Clarendon filter. These differences in errors are used to determine the capabilities of the naked eye.

In addition to deciding, if every two pairs of images are the same, the respondents are asked to explain what are the differences between the two images, if the images seem different to them. This can give us an idea, of what are the weak parts of our final algorithm. Each questionnaire was done with supervision. For each question, 15 seconds to analyze the images is given. After this time, enough time is given for

Table 4.1: Errors for three images used in the questionnaire for each filter algorithm

Implementation	Image 1	Image 2	Image 3
Our final algorithm	2.02	1.18	1.75
Algorithm 1	3.55	3.15	3.73
Algorithm 2	3.07	2.25	2.26

Table 4.2: Statistics from the questionnaire showing percentage of answers assuming the two images were the same

	Image 1	Image 2	Image 3	Total
Our reproduction	93.3%	80%	73.3%	82.2%
Algorithm 1	6.6%	0%	0%	2.2%
Algorithm 2	80%	33.3%	33.3%	48.8%
Two same images	80%	86.6%	93.3%	86.6%

writing the answer. A total of fifteen respondents participated in this survey. Results separated by the individual images and by different algorithms are shown in table(Table 4.2).

From the table (Table 4.2) a number of interesting observations can be made. *Algorithm 1*, which showed about 3.5 percent error on average compared to the Clarendon filter, produced images, that were clearly identified as different from the Clarendon filter. Only about 2.2 percent of images were identified as the same. This means that if enough time is given to study the images, the naked eye can easily spot the 3.5 percent difference.

Another interesting observation is, that only about 86.6 percent of the exact same images were correctly identified as the same. This can be a result of the fact, that the respondents were specifically told, that they should look for the smallest details in the images and the brain forced them into thinking, that they see differences between the images that were not present.

When we look at *Image 1* statistics, the first thing that is noticed is the fact, that a larger percentage of respondents identified two images as the same when comparing the Clarendon filter with our final implementation of the filter than when comparing two exactly the same images. This is a clear sign, that Image 1 filtered with the Clarendon filter, and our reproduction of this filter is so similar, that it is beyond the point the difference can be identified with the naked eye.

When looking at *Image 2* statistics, the difference between the perceived identity of two images between our reproduction of the Clarendon filter and the two same images is only 6 percent. This means the difference between our algorithm and the Clarendon filter can be also marked as indistinguishable with the naked eye.

On the other hand, statistics for *Image 3* are a little different. With a 20 percent difference between the Clarendon filter and our filter, more analysis is made to see why this particular image filtered with our filter seems a little different than with the Clarendon filter.

From analyzing the reasons given by the respondents of why they think *Image 3* filtered with our filter compared to the Clarendon filter was different, one answer is mentioned in almost all of them. About a third of the image consists of a very light and structured sky with detailed transitions between blue and grey colors (Figure 4.2). When looking at the images closely, a small difference can be seen between the way Clarendon filter and our filter processed the image. However, the difference is so small, that if images are not placed side by side but viewed one after each other, there is no chance a person would be able to tell the difference. For this reason, a conclusion is made. Our final filter algorithm is very accurate and in most cases, no difference can be seen with the naked eye when compared to the Clarendon filter. The only small weakness of our reproduction of the Clarendon filter is present when the filter is used on very bright but structured images.

5 Conclusion

5.1 Conclusion from the thesis

To sum up the thesis, there are a couple of important observations to be made. The main goal of this thesis, which was to create a reproduction of the Clarendon filter inside Gnome Photos application was successfully fulfilled. Furthermore, the approach was documented in detail, to be a helpful guide for any future similar reproductions. It can serve as a tool for not only similar reproductions of filters inside the Photos application but also any reverse engineering of any image filter in general. Various approaches were discussed in many different parts of the research. This work improves some of the techniques used in the work Reverse Engineering Instagram's "Hefe" Filter [6].

At the beginning of the thesis, the introduction to the Clarendon filter is made, to familiarize readers that are not using Instagram with the filter characteristics. After that follows the chapter with all experiments used for reverse-engineering the filter. First, the general idea of experiments is described in the introduction to experiments. Image test set generation is mentioned and the image set is available in the electronic appendix [Script 1], to show the reader exact data used for the experiments. Since this set was generated with very similar code as in Corey Hoard's work, it can be improved in the future to contain images that can resolve in even more in-depth reconstruction of a similar filter.

During all of the experiments, the main source of ideas and information was Corey Hoard's work [6]. However, there are key differences in this thesis compared to his work. The first experiment, which consists of the search for the best color model for the filter reproduction, resulted in a very similar way, with the RGB color model being the ideal candidate for this purpose. In future research of other filters, other color models could be analyzed and their performance tested. In this thesis, the results that the RGB color model offered were satisfying for our purpose.

When searching for pixel position-dependent operations in the second experiment, the presence of any type of image overlay was disproved. This was not the case with the Hefe filter, so if another filter is researched and it turns out to contain image overlay, the best option is to follow steps in Corey Hoard's work. [6]

The third experiment was designed for detecting inter-pixel dependent operations. During this experiment, the presence of these types of operations was also disproved. This experiment was improved in comparison to a similar experiment from Hefe reconstruction. Techniques used during the reconstruction of the Hefe filter were not designed to detect operations like sharpening. In this thesis, the second part of the experiment was added to detect the sharpening and to eliminate possible inaccuracy of the final algorithm. If some other filter is research in this way and turns out to contain blurring kernels or sharpening, it has to be reproduced in the final algorithm.

The last experiment was designed to detect a dependency between the individual RGB channels. This experiment is the main differentiating factor for the following implementation in comparison to a Corey Hoard's work. Contrary to the Hefe filter, the Clarendon filter does not have all of the RGB channels independent on each other. Dependencies of a blue channel on a green channel and a red channel on a green channel were detected. This shaped the following implementation. For future research of another filter, results from this experiment can be very helpful. Since none of the channels were dependent on all other channels in our case, this option is yet to be researched in the future with other filters.

During the development of the filter itself, multiple approaches are chosen. All of the decision making to create the algorithms are documented. From all of the possible implementations, best 4 are chosen. Red and blue channels are transformed using polynomial equations with different polynomial degrees, to find the best possible solution. Green channel is implemented using both lookup table and equation. All of the equations were created using software Matlab.

After creating these 4 algorithms, each algorithm is implemented into the Gnome Photos application. Used libraries and technologies are documented. This part can help future implementations of other filters since all necessary changes in the application framework are mentioned. Ten images are processed using Instagram's Clarendon filter and also using our 4 created algorithms. For each set of filtered images, average error compared to Clarendon filter separated by individual RGB channels is counted. After generating this data one of the four approaches is chosen as the final algorithm.

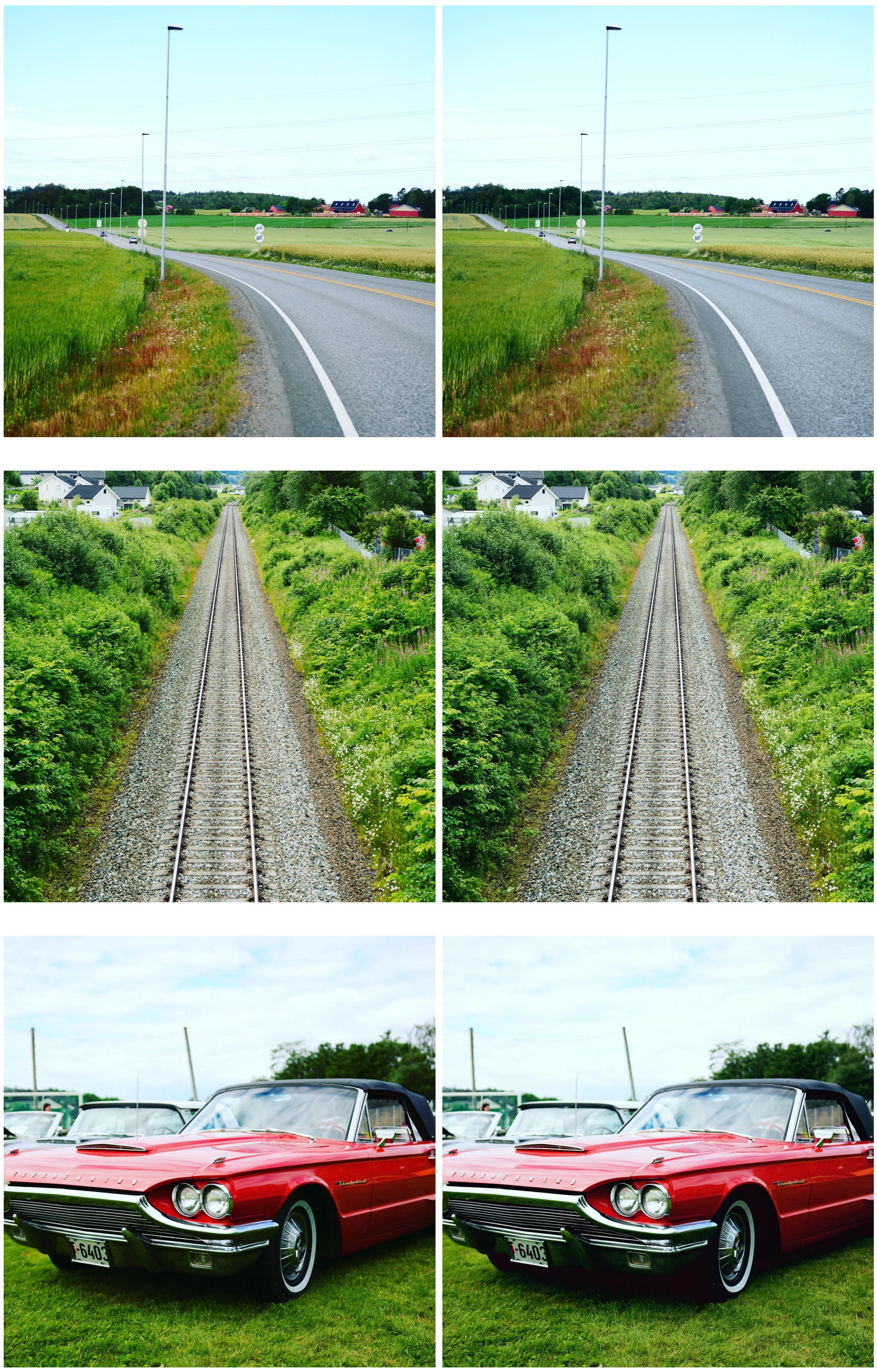
The final implementation is done by three equations. This approach turned out to be the best way for the filter to be reproduced. This part can serve as a strong motivation for future implementations of other filters inside the Gnome Photos application.

In the last chapter, the accuracy of the naked eye is tested. After researching a group of 15 people, a conclusion was made, that our reproduction of the Clarendon filter is so similar to the original Clarendon filter, that it is almost indistinguishable with the naked eye. This conclusion could be made since about 82 percent of images processed with our reproduction of the Clarendon filter were considered the same to the images filtered with the Clarendon filter. At the same time, only about 86 percent of the same images placed side by side were correctly identified as the same. For this reason, our reproduction of the filter can be considered very accurate.

5.2 Visual comparison of the Clarendon filter and our reproduction of the filter

In this section, 15 images filtered with both Clarendon, and our filter can be seen placed side by side. On the left original Clarendon filter is placed and on the right is the final reproduction of the filter.

RS500RS
PERFORMANCE
wizzair.com
WIZZ



AOH 280
AOH 280

Bibliography

1. *Color-filter-reconstruction*. Karpinsky Alexander, 2018. Available also from: `https://github.com/homm/color-filters-reconstruction`.
2. *Your Instagram filter cheat sheet* [online]. Lucille Zimmerman, 2019 [visited on 2020-01-05]. Available from: `http://lucillezimmerman.com/2017/01/07/instagramfilter/`.
3. *Advanced surface fitting techniques* [online]. V. Weiss, L. Andor, G. Renner, T. Várady, 2002 [visited on 2020-01-05]. Available from: `http://www.sciencedirect.com/science/article/pii/S0167839601000863`.
4. *Instagram marketing statistics for social media marketing gurus* [online]. 99 firms, 2019 [visited on 2020-01-10]. Available from: `https://99firms.com/blog/instagram-marketing-statistics`.
5. *Statistics: how filters are used by Instagram's most successful users* [online]. Kaptur, 2017 [visited on 2020-01-12]. Available from: `https://kaptur.co/statistics-how-filters-are-used-by-instagrams-most-successful-users/`.
6. *Reverse Engineering Instagram's "Hefe" Filter* [online]. Corey Hoard, 2014 [visited on 2020-01-12]. Available from: `https://s3-eu-west-1.amazonaws.com/pfigshare-u-files/1726806/CoreyHoardReverseEngineer` `pdf`.
7. *What is Color Space* [online]. Arc soft, 2016 [visited on 2020-02-12]. Available from: `http://www.arcsoft.com/topics/photostudio-darkroom/what-is-color-space.html`.
8. *Understanding lens vignetting* [online]. Red [visited on 2020-02-13]. Available from: `https://www.red.com/red-101/lens-vignetting`.
9. *JPEG Compression Algorithm* [online]. Danoja Dias, 2017 [visited on 2020-02-14]. Available from: `https://medium.com/breaktheloop/jpeg-compression-algorithm-969af03773da`.

10. *Chapter 24. Using Lookup Tables to Accelerate Color Transformations* [online]. Jeremy Selan, 2005 [visited on 2020-02-15]. Available from: `https://developer.nvidia.com/gpugems/gpugems2/part-iii-high-quality-rendering/chapter-24-using-lookup-tables-accelerate-color`.

11. *What is Matlab* [online]. MathWorks [visited on 2020-02-16]. Available from: `https://www.mathworks.com/discovery/what-is-matlab.html`.

12. *Matlab - the language of technical computing* [online]. MathWorks [visited on 2020-01-11]. Available from: `https://www.mathworks.com/help/matlab/`.

13. *Gnome-photos* [online]. Gnome [visited on 2020-05-21]. Available from: `https://gitlab.gnome.org/GNOME/gnome-photos`.

A An electronic appendix

This thesis contains an electronic appendix which can be found at https://is.muni.cz/auth/th/uy5lo//.
In the appendix, these files can be found:

- Python scripts used in the experiments
- Source code of the final implementation of the algorithm

A.1 Python scripts used in the experiments

In this part of the appendix, seven Python scripts are available. This scripts can by customized and used for reverse-engineering another image filter. Below are the descriptions for each of the scripts.
[Script 1] - Image test set generation.
[Script 2] - Separation of RGB channels plot
[Script 3] - Overlay detection
[Script 4] - Combining the black and white pixels
[Script 5] - Error computing
[Script 6] - Generating visual representation of the error
[Script 7] - RGB channels dependency graphs plot

A.2 Source code of the final implementation of the algorithm

In this part of the appendix, the source code of the final implementation of the filter can be found.
[Implementation] - Source code from the file *photos-operation-insta-clarendon.c* from the directory *src* of Gnome photos gitlab repository [13].