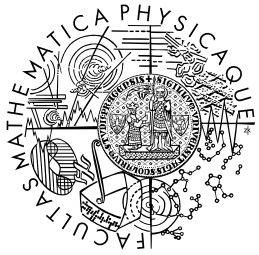


FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

BACHELOR THESIS

Martin Studna

Procedural generation of pencil drawings

Department of Software and Computer Science Education

Supervisor of the bachelor thesis:	doc. Ing. Jaroslav Křivánek Ph.D.
Study programme:	Computer Science
Study branch:	Programming and Software Systems

Prague 2019

I declare that I carried out this bachelor thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date ..... signature of the author

I would like to express my gratitude to doc. Ing. Jaroslav Křivánek PhD. for being an amazing supervisor. Without countless discussions and advices this work could not have happened. Special thanks to Filip Hauptfleisch who helped me with technical aspect of the project. Futhermore, I would like to thank also to everyone who helped me during my studies. I especially appreciate that my mother supported me through many hardships and tried her best to make appropriate conditions for my studies.

Title: Procedural generation of pencil drawings

Author: Martin Studna

Department: Department of Software and Computer Science Education

Supervisor: doc. Ing. Jaroslav Křivánek Ph.D., Department of Software and Computer Science Education

Abstract: The aim of the thesis is to implement a procedural method which transfers a natural image into a pencil drawing-like style. Our project is written in C++. It uses libraries like OpenCV for image processing and Eigen for linear algebra computations. Since neural networks are frequently questioned, as to whether or not they are better than procedural methods for artistic style reproduction, this work presents also a detailed comparison of both of these approaches. We have re-implemented a selected method for procedural generation of pencil drawing style, bringing several modifications. We compare results of the method with a recently released code for neural network-based drawing generation. The result of this subjective comparison indicates that neural networks maybe be better suited for the generation of pencil-like hatching texture to reproduce shading. On the other hand, the procedurally generated outlined produced by the implemented approach provide more natural renderings.

Keywords: Non-photorealistic rendering, procedural methods, pencil drawing, image processing, neural networks

Contents

1. Introduction

Our work deal with the problem of turning an image into a pencil drawing. Similar methods, which are concerned with converting an image into a painting, animated cartoon or technical illustration are part of the Non-photorealistic rendering area. NPR has wide application in the movie industry, game development, architectural illustrations or scientific visualizations.

Since the development of computers has significantly moved forward over the last two decades, it enabled scientists from the computer graphic area to produce more impresive results. We have been overwhelmed by the works of Picasso, Da Vinci or Michelangelo. However, it is really challenging for us to teach the computer to imitate the style of a well-known artist. In addition, it would be very interesting, if the computer could produce its unique painting or drawing style. In our thesis, we present the computer-generated pencil drawing algorithms

We are going to describe in more details a method designed by Lu et al. [2012] which we also implemented. Their method is based on line and tone extraction which processes these two parts of the image independently. They attempt to simulate that lines are drawn several times, inconsistently and in small parts. Futhermore, they divide tones into three tonal layers based on the amount of light that each pixel carries. We discuss how to adjust the tones of the images so the images correspond more to pencil drawings, how to generate outlines and how to make pencil texture fit the image.

More recently, neural networks were applied even in NPR and produced impressive results. There have been several researches which attempt to develop these methods and probably we are going to hear more about them.

The thesis is structured as follows: The second chapter provides the reader with a general overview of the existing methods in NPR, and computer-assisted pencil drawing generation in particular. Then, we discuss the algorithm design. Afterwards, we describe more the implementation details. Our motivation to use certain data structures, libraries, methods and parameters. Thereafter, we present our results. We also compare the results with another project that attempted to achieve the same goal, which is based on neural networks. In the last chapter, we make a conclusion.

2. Related work

Numerous methods have been developed to transfer an image into a pencil drawing. The most common approach is to process the outlines and tones of the image independently. But some of these methods concentrate only at one area. Still, it is important to mentioned them, because they could be probably combined in the future experiments. In the first section we present several procedural methods and in the second section we describe the methods based on neural networks.

2.1 Procedural methods

The first algorithm worth mentioning is the Jin Zhou and Baoxin Li [2005] method for automatically generating pencil-sketch like drawings from personal photos. Their approach is quite different from the others. Instead of using the popular Sobel operator as well as other conventional edge detectors, they propose the algorithm for gradient estimation. They claim that one of the drawbacks of the detectors is that only a small neighborhood is used to compute the gradient. This has the effect that the operators extract even the noisy segments. Their method produces probably the most visually appealing pencil-like lines from the methods we studied. Other methods generate unnecessarily too many edges for very detailed structures which significantly worsens the overall impression of the drawing.

On the other hand, Kang et al. [2007] propose a method that produces the Edge Tangent Flow field that preserves the flow of the salient image features. They then introduce the notion of flow-guided anisotropic filtering for detecting highly coherent, smooth and stylistic lines while suppressing noise. Even if the lines represent the shapes quite accurately, they are solid which is unnatural for pencil drawings because lines are usually drawn with small strokes.

Moreover, pencil drawing-like transfer can be applied even in real-time. Lee et al. [2006] designed the real-time technique for rendering 3D meshes in the pencil drawing style. They analyze the characteristics of pencil drawing and incorporate them into the rendering process. For object contours, they propose a multiple contour drawing technique that imitates trial-and-errors of human in contour drawing. For interior shading, they present a simple approach for mapping oriented textures onto an object surface. To improve the quality of pencil rendering, they generate and map pencil textures that reflect the properties of graphite pencils and paper.

Recently, Sun and Huang [2019] propose an extension to the existing automatic pencil drawing generation technique based on Line Integral Convolution. LIC is a texture based vector field visualization method. It takes a 2D vector field and a white noise image as the input, and generates an image which has been smeared out in the direction of the vector field throught the convolution of the white noise and the low-pass filter kernels defined on the local streamline of the vector field Sun and Huang [2019]. Futhermore, they improve the LIC method with more accurate and rapid graph-based image segmentation method to divide the image into different regions.

Wen et al. [2006] introduce an NPR system to generate color sketches in a free-

hand drawing style. They use interactive segmentation algorithm. In addition, they apply two operations. First, they shrink the boundary of the color regions with a luminance-based algorithm emphasizing the highlights part, which enforces consistency with free-hand artist style of the sketch. Second, they propose color shift algorithm which is applied to image regions emphasize the main content of the sketch and acquires a visually pleasing combination of color. They choose the optimal combination by minimizing the energy function based on an artist drawn color database, artistic drawing rules and input image colors.

Our implementation is inspired by the work of Lu et al. [2012]. They combine the tone and stroke structures. First, they take eight directional line kernels and convolve them with gradient map. At last, they adjust the tone of the image and map texture on it.

More recently, Okawa et al. [2017] referred to the work of Lu et al. [2012]. Instead of using the Sobel operator they extracts the edges with the Canny operator. Moreover, they propose to distort the main edges strongly and the texture edges weakly because the main countours are drawn several times, but texture countours are drawn only once. Outlines of their images seem to be quite impressing. Their lines did not overlap the boundaries of the shapes as it was in Lu et al. [2012] case and it seems as the lines depict the objects more accurately. Still we cannot claim which results of which method resemble more to a pencil drawing. Some of the drawing style count with overlaping the objects.

2.2 Neural Style transfer methods

Gatys et al. [2015] were inspired by the power of Convolutional Neural Networks. It was the first work indicating that neural networks could be capable of extracting content information from an arbitrary photograph and style information from an artwork. They opened up a new field called Neural Style Transfer Jing et al. [2017].

Zhu et al. [2017] propose image-to-image translation method. Their goal is to learn the mapping between an input image and an output image using a training set of aligned image pairs. However, for many tasks, paired training data will not be available. They present an approach for learning to translate an image from a source domain X to a target domain Y in the absence of paired examples. Their goal is to learn a mapping $G : X \rightarrow Y$ such that the distribution of images from $G(X)$ is indistinguishable from the distribution Y using an adversarial loss. Because this mapping is highly under-constrained, they couple it with an inverse mapping $F : Y \rightarrow X$ and introduce a cycle consistency loss to push $F(G(X)) \approx X$ (and vice versa).

Recently Li et al. [2019] presented a method similar to that of Gatys et al. [2015] and even to Lu et al. [2012]. They train deep neural networks on outlines and tones. The model is trained on a dataset from pinterest. In addition, their framework allows to produce several styles.

Gao et al. [2018] present a framework named as PencilArt for the chromatic penciling style generation from wild photographs. The structural outline and textured map for composing the chromatic pencil drawing are generated, respectively. First, they take advantage of deep neural network to produce the structural outline with proper intensity variation and conciseness. Next, for the textured

map, they follow the painting process of artists to adjust the tone of input images to match the luminance histogram and pencil textures of real drawings.

3. Algorithm description

From all of the mentioned procedural methods, we concluded that it would be most suitable to implement the algorithm proposed by Lu et al. [2012]. They take into consideration several factors which are line extraction, tone adjustment and pencil texture mapping. Some of these approaches also took these points into account but they seemed to us clearly inferior to Lu et al. [2012] approach or they concentrate at only one characteristic of pencil drawings.

The technical description in this chapter follows the algorithm design proposed by Lu et al. [2012]. Moreover, we made some minor modifications which we will highlight.

Their overall framework consists of two main steps: pencil stroke generation and pencil tone drawing. Their effect complement each other. Specifically, stroke drawing aims at expressing general structures of the scene, while the tone drawing focuses more on shapes, shadow and shading than on the use of lines. These two steps are processed separately. It is noteworthy that there exist several pencil drawing styles but their framework produces only one.

3.1 Line Drawing

An important observation is that artists cannot always draw very long curves without any break. Strokes end often at points of curvature and junctions. In addition, there might be crosses at the junction of two lines in pencil drawing. These are critical evidences for human-drawn strokes Lu et al. [2012].

(a) Gradient (b) Line image

Figure 3.1

First, they detect the edges of the image. As we know, edges are places of the image where the pixel intensity changes rapidly. From a mathematical point of view, we can imagine that the image is a two dimensional continuous function. Therefore, the derivatives can compute local difference of some neighborhood. We can use similar approach as approximations to the true spatial derivatives in image processing.

For edge detection, we use the Scharr operator Levkine [2011]. It uses two gradient matrices that detect edges in horizontal and vertical direction. It is expressed as:

$$G = \sqrt{(G_x I)^2 + (G_y I)^2} \tag{3.1}$$

where I is the grayscale input. G_x and G_y are gradient operators in two directions. Usually the gradient maps are typically noisy and do not contain continuous edges immediately ready for stroke generation.

One issue of generating short lines for sketch is the estimation of line direction for each pixel. Naive classification to this end uses the gradient direction, which is sensitive to noise and thus is unstable. They propose a more robust strategy using local information Lu et al. [2012].

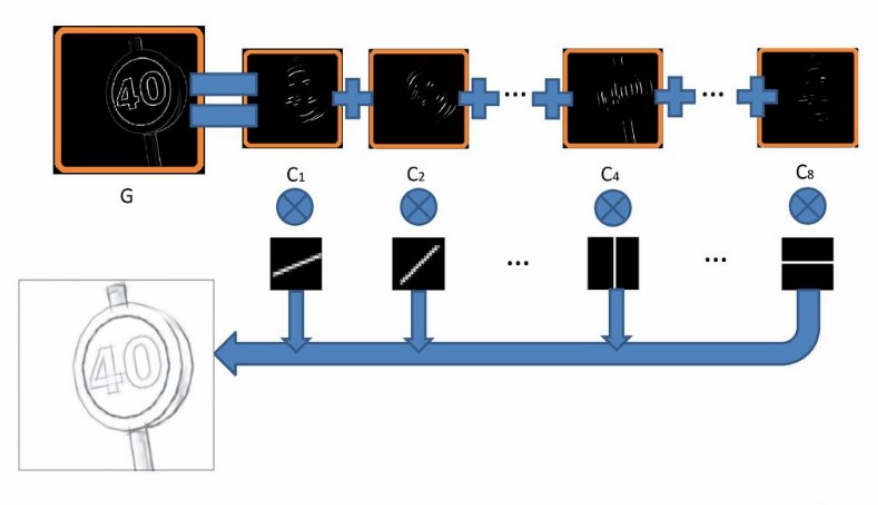


Figure 3.2

First, they choose eight reference line directions. Each of them represents directions at 45 degrees apart and denote the line segments as $\{\mathcal{L}_i\}, \in \{1..8\}$. The response map for a certain direction is computed as:

$$G_i = \mathcal{L}_i * G \tag{3.2}$$

where $\mathcal{L}_i$ is a line segment at the i^{th} direction. $*$ is the convolution operator, which groups gradient magnitudes along direction i to form the filter response map G_i.

Then, the classification is performed by selecting the maximum value among the responses in all directions and is denoted as

$$G_i(p) = \begin{cases} G(p) & \text{if } argmin_i\{G_i(p)\} = i. \\ 0 & \text{otherwise.} \end{cases} \tag{3.3}$$

where p is a pixel and C_i is the magnitude map for direction i. Their relationship with G is expressed as $\sum_{i=1}^{8} C_i = G$.

Line shaping Given the map set C_i, we generate lines at each pixel also by convolution. It is expressed as

$$S^{'} = \sum_{i=1}^{8} (\mathcal{L}_i * C_i). \tag{3.4}$$

Convolution aggregates nearby pixels along direction $\mathcal{L}_i$, which links the edge pixels that are not even connected in the original gradient map. This process is also very robust to noise and other image visual artifacts. Finally, we invert the pixel values and map them to $[0, 1]$.

3.2 Tone Drawing

Artists also use several styles of dense strokes such as hatching, stippling, blending or crosshatching. Many methods use image tones to generate hatching patterns. They propose that this process is not optimal because the image tones are usually significantly different from that of pencil sketch Lu et al. [2012].

(a) Grayscale (b) Tonal layers

(c) Tone map

Figure 3.3

We have a grayscale image whose tone values are in range 0 to 255. We divide these values in three layers: bright, mid-tonal and dark layer. They propose a parametric model which is written as:

$$p(v) = \frac{1}{Z} \sum_{i=1}^{3} w_i p_i(v), \tag{3.5}$$

where v is the tonal value and $p(v)$ is the probability that the tonal value occurs in the pencil drawing. w_s are the weights coarsely corresponding to the number of pixels in each tonal layer. Z is the normalization factor to make $\int_0^1 p(v)dv = 1$.

We set boundaries of tonal layers as follows: brightest from 225 to 255, mid-tonal values from 105 to 225 and dark values from 0 to 105. For very bright regions we apply Laplacian distribution, that is written as:

$$p_1(v) = \begin{cases} \frac{1}{\sigma_b} e^{-\frac{1-v}{\sigma_b}} & \text{if } v \leq 1. \\ 0 & \text{otherwise.} \end{cases} \tag{3.6}$$

As a result, Laplacian distribution makes the brightest layer even brighter. This causes that it resemble more to the paper. On the other hand, we want

mid tonal layer to capture all important details in the image. We apply uniform distribution so the gray values mostly stay unmodified. It is expressed as:

$$p_2(v) = \begin{cases} \frac{1}{\mu_b - \mu_a} & \text{if } \mu_a \leq v \leq \mu_b. \\ 0 & \text{otherwise.} \end{cases} \tag{3.7}$$

Finally, we use normal distribution on dark strokes. It is denoted as:

$$p_3(v) = \frac{1}{\sqrt{2\pi\sigma_d}} e^{-\frac{(v-\mu_d)^2}{2\sigma_d^2}} \tag{3.8}$$

Lu et al. [2012] claim that usually variation of pixel values in the dark layer is in general much larger than that in the bright layer. After we compute distribution for each tonal layer, we superpose them.

Our resulting tonal image is computed with histogram matching between the tonal distribution defined in eq. 3.5 and the tonal distribution of the grayscale image. Therefore, we need to compute a probability density function from the histogram of the grayscale image. The pdf is written as

$$p_r(r_j) = \frac{n_j}{n} \tag{3.9}$$

where n_j is the frequency of the grayscale value r_j and n is the total number of pixels in the image. Afterwards, we map each pdf to its cumulative distribution function which are denoted as:

$$S(r_k) = \sum_{j=0}^{k} p_r(r_j), \ k = 0, 1, 2, 3, ... \tag{3.10}$$

$$G(z_k) = \sum_{j=0}^{k} p_z(z_j), \ k = 0, 1, 2, 3, ..., L \tag{3.11}$$

where L is the total number of gray levels. The idea is to map each r value in X to the z value that has the same probability in the desired pdf: $S(r_j) = G(z_i)$ or $z = G^{-1}(S(r))$. Besides, there might be a case where we will not get exactly an equality. We find the smallest absolute difference between $S(x_1)$ and $G(x_2)$. In other words, for a mapping function M, for each entry x_1, we must find an intesity x_2 such that:

$$M(x_1) = arg \min_{x_2 \in [0,255]} |F_1(x_1) - F_2(x_2)| \, \forall x_1 \in [0, 255] \tag{3.12}$$

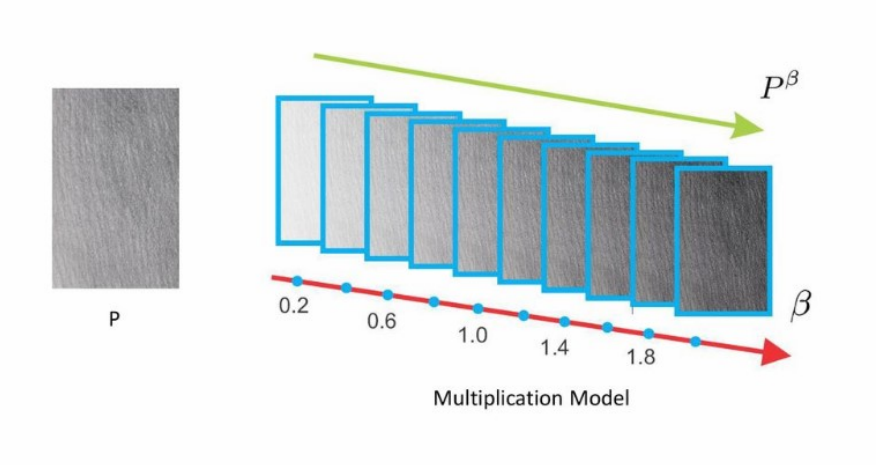
Once, we receive the mapping function, we apply it on the grayscale image.

$$I(x) = M(x) \tag{3.13}$$

where I is our grayscale image, x tonal value and M the mapping function.

3.3 Pencil Texture Rendering

Generating suitable pencil textures for image is difficult. Texture refers to pencil patterns without obvious direction which reveal only tone information Lu et al. [2012].



(a)

Figure 3.4

Our input image only needs one pencil pattern. We will denote the pencil pattern by P and the tonal image by J. Human drawing is generated by repeatedly drawing at the same place. They simulate the process using multiplication of strokes, which results in an exponential combination $P(x)^{\beta(x)} \approx J(x)$ or in the logarithm domain $\beta(x)lnP(x) \approx ln(x)$ corresponding to drawing pattern $P\beta$ times to approximate the local tone in J.

First, we need to find appropriate β which optimizes our texture so it can be used in our image. It is computed by solving

$$\beta^* = arg\,\min_{\beta} \|\beta lnP - lnJ\|_2^2 + \lambda\|\nabla\beta\|_2^2 \tag{3.14}$$

where λ is the weight with value 0.2. We transfer the equation to the system of linear equations with these steps:

We can make

$$\|\beta\|_2^2 = \|D_x\beta\|_2^2 + \|D_y\beta\|_2^2 \tag{1}$$

Then the optimization meet the following condition:

$$(\beta\, lnP - lnJ)^T lnP + \lambda\,(\beta\, D_x^T\, D_x + \beta^T\, D_y\, D_y) = 0 \tag{2}$$

Furthermore, both sides take the transpose

$$\beta\, lnP^2 - lnJ\, lnP + \lambda(D_x^T\, D_x + D_y\, D_y)\,\beta = 0 \tag{3}$$

Thus solvable

$$[\,\lambda\,(D_x^T\, D_x + D_y^T\, D_y) + lnP^T\, lnP\,]\beta = lnP^T\, lnJ \tag{4}$$

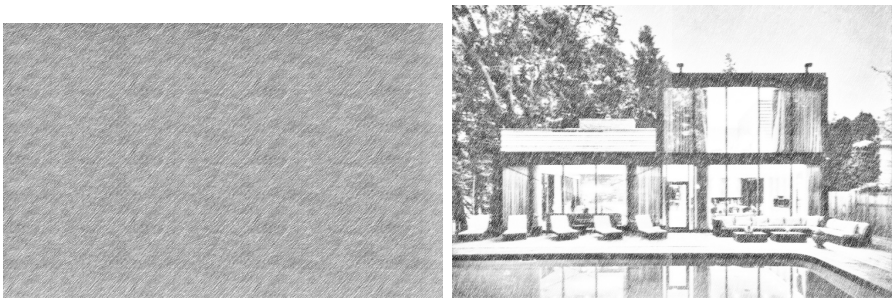
Transformation form

$$[\,diag(lnP^2) + \lambda(D_x^T\, D_x + D_y^T\, D_y)\,]\,\beta = lnJlnP \tag{5}$$

because $[\,diag(lnP^2) + \lambda(D_x^T\, D_x + D_y^T\, D_y)\,]$ is symmetric positive definite matrix, it can be solved by using the Conjugate gradient method.

The final pencil texture map T is computed through an exponential operation which is denoted as:

$$T = P^{\beta^*} \tag{3.15}$$



(a) Pencil Texture (b) Texture image

Figure 3.5

At last, we calculate the product of the line map and the texture map to receive the pencil drawing. It is written as:

$$R = S * T \tag{3.16}$$

(a) (b)

Figure 3.6: Pencil drawings

4. Implementation details

We developed our project in the C++ language on Windows platform. Furthermore, we included also a CMakeList, which makes our project portable even to UNIX-like systems.

For image manipulation, we use the OpenCV. OpenCV is cross-platform and open-source library mainly for image processing and computer vision. It contains many useful algorithms, for example, for edge detection or object recognition. In addition, we also use the Eigen library. It has methods for linear algebra, matrix and vector operations, geometrical transformations and numerical solvers. We especially use it to solve some difficult optimization problems.

Our project takes five input parameters. Path to image, path to specific pencil texture and three weights which control brightness in three tonal layers, which are passed to the eq. 3.5.

We have divided our project into four seperate parts. One is the main function and other three parts are line, tone and texture method.

```
// Sketches.h
void generate_lines(Mat gray, Mat& grad, Mat& dst);
void generate_tones(Mat src, Mat& dst, int w1, int w2, int w3);
void generate_texture(Mat tones, Mat texture, Mat& dst);
```

4.1 Line method

Before an image is passed to the line generation function, it is converted into grayscale. Each pixel is then represented with tonal value ranges from 0 to 255. Then, we apply a Gaussian blur to reduce the noise.

We made a minor change in the method proposed by Lu et al. [2012]. For edge detection, we use the Scharr operator instead of the Sobel operator. Edge detector consists of directional kernels. We wanted the kernels size to be 3x3. But in the OpenCV documentation of Sobel operator is written that if kernels have size 3x3, so Sobel produces some inaccuracies. The Scharr operator is almost the same, but it has different kernels that are written as follows:

$$G_x = \begin{bmatrix} -3 & 0 & +3 \\ -10 & 0 & +10 \\ -3 & 0 & +3 \end{bmatrix}, \quad G_y = \begin{bmatrix} -3 & -10 & -3 \\ 0 & 0 & 0 \\ +3 & +10 & +3 \end{bmatrix}$$

We use these kernels in eq. 3.1 where we use $addWeighted$ function for the sum of both directional gradients, because we followed implementation of the Scharr operator from OpenCV official example `https://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/sobel_derivatives/sobel_derivatives.html`

Lu et al. [2012] propose line kernels representing each line segment in 45 degrees direction apart. Then, they sum of $\mathcal{L}_i G$ as i goes from 1 to 8, as it is written in eq. 3.2. It is confusing that mentioned formula counts with eight kernels. If we represented eight directions with 45 degrees apart, we will need

only four kernels. We suggest a little modification that consider eight kernels which each of them represents direction of 22.5 degrees apart.

They also suggest that size of the line kernel should be 1/30 of input image. Our kernels has 5x5 size for all the input images.

4.2 Tone method

In our tone generation fuction, we produce histograms of input image, tonal layers and result. For study reasons, we also generate colored layer image. We want to see which pixels fell into which layer.

Before we compute the PDF of each layer, we normalize the values to $[0, 1]$. We use OpenCV function normalize, that is defined as:

```
normalize(Mat InputMat, Mat& InputOutputMat, double alpha, double
    beta, int norm_type);
```

Lu et al. [2012] propose three sets of weights that they used for generating the tonal images.

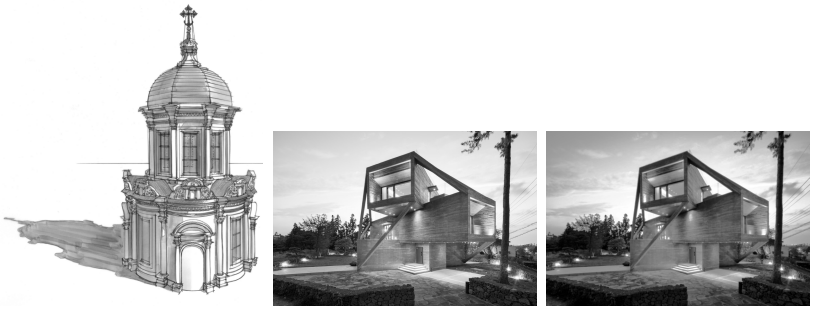
$$11 : 37 : 52, \quad 29 : 29 : 42, \quad 2 : 22 : 76 \tag{4.1}$$

For study purposes, we created a function that takes arbitrary weights. Moreover, we use mean values of probability distributions which were also suggested by Lu et al. [2012] which can be seen in fig. 4.1.

ω_1	ω_2	ω_3	σ_b	u_a	u_b	μ_d	σ_d
11	37	52	9	105	225	90	11

Figure 4.1: Parameters.

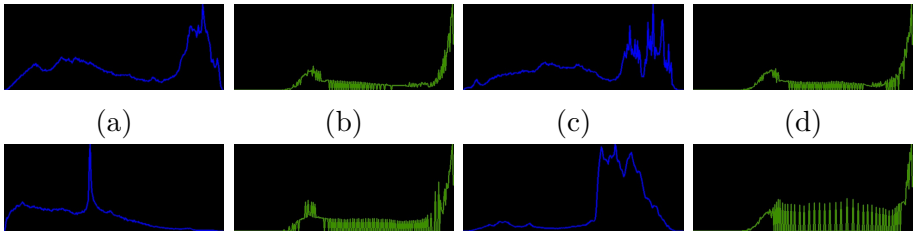
Firstly, we thought the weights can be evaluated from the number of pixels of each tonal layer of certain pencil drawing. Afterwards, we would estimate the mean values of each probability distribution function. But, unfortunately, our project did not produce any visually appealing results.



(a) Drawing (b) Grayscale (c) Tone map

Figure 4.2

In fig. 4.2 is the pencil drawing which we used for the estimation of the parameters and weights. We can notice that there is a small difference between the grayscale image and tonal image.



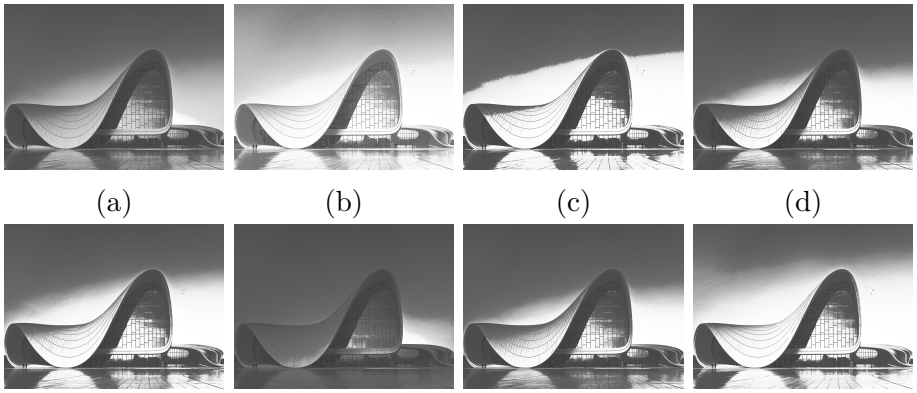
(a) (b) (c) (d)

(e) (f) (g) (h)

Figure 4.3: Histograms - blue: input, green: tonal image

In fig. 4.3 after applying probability distributions we can see how the histogram of our input image has changed. In this example we used the weights mentioned in the fig. 4.1.

In the fig. 4.4 we also present set of tone images. Each of them were generated with different weight values.



(a) (b) (c) (d)

(e) (f) (g) (h)

Figure 4.4: Tones.

4.3 Texture method

In our texture generation function, tonal image and pencil texture are taken as the input parameters.

First, we set the pencil texture image size to the size of the tonal image. If the pencil texture fits several times in the tonal image, we repeat the pencil texture. Then, we make one column vectors from both of our images.

Lu et al. [2012] propose to solve the system of linear equations with conjugate gradient method. It is recommended to transfer the matrices into sparse matrices, because CG is usually extremely efficient on normal equations. In addition, CG is especially useful for matrices that are symmetric and positive-definite.

Eigen implements sparse matrices with Compressed Column or row storage scheme. It is represented with three one-dimensional arrays. First contains all non-zeros values of the matrix. Second has column indices of the values of the matrix. Third contains indices of the first array of non-zero values that appeared first on the matrix row.

$$\begin{bmatrix} a & 0 & b \\ c & d & e \\ 0 & 0 & f \end{bmatrix}$$

$$values \begin{bmatrix} a & b & c & d & e & f \end{bmatrix}$$

$$columns \begin{bmatrix} 0 & 2 & 0 & 1 & 2 & 1 \end{bmatrix}$$

$$rowptr \begin{bmatrix} 0 & 2 & 5 \end{bmatrix}$$

For CG, it is also convenient to use some sort of preconditioner so that the algorithm is more effective. We diagonalize our directional kernels. Furthermore the diagonalization, our Dx kernel has minus ones on main diagonal and plus ones on tone map height diagonal. Dy also has minus ones on the main diagonal but it has plus ones on the plus one diagonal.

We make little optimization, when we compute natural logarithm of texture image and tone image. Both of our images contain zero values. Computing the natural logarithm of zero is infinity, which is not appropriate for our computations. We set zero values to 0.001f.

We also set the tolerance threshold of CG to $1e-6$ and maximum number of iterations to 60.

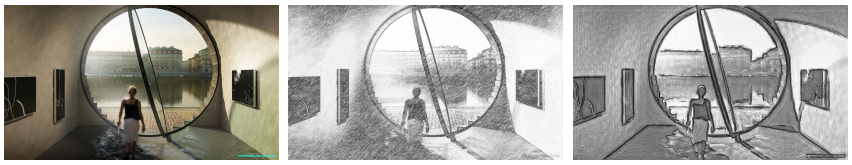
OpenCV has, unfortunately, poor documentation about conjugate gradient method solver. We could look at the source code of the solver, but it seemed to us too complicated. We decided that we are going to rely on other library. There are several libraries which have proper implementation of this method. We use library Eigen.

It is convenient that Eigen and OpenCV are compatible libraries. They have a method which can convert OpenCV matrices to Eigen matrices and otherwise.

By this approach, we solve the system of linear equations and transfer it back to OpenCV matrix data types. After that, we compute texture image power to β.

Finally, we calculate the product of the line map and the texture map to receive pencil drawing.

5. Results



(a) Input (b) Ours (c) Im2Pencil

(d) Input (e) Ours (f) Im2Pencil

Figure 5.1: Comparison of both methods.

Generating pencil drawings for images that we utilized in our project and Im2Pencil, we can see significant differences between the results. If we look at image (d), there is noise around domes. Testing the other images, we did not notice this kind of effect. We used pretrained model which was stored in the project. Perhaps if the model was trained differently, this effect would disappear.

On the other hand, our program does not produce this noise-like effect in any case. If we look at the images, Im2Pencil produces much more credible texture maps. It seems as if the texture is directly mapped on the surface of building.

If we compare the outlines, we have rather opposite opinion. Im2Pencil offers to select clean and rough style for the outlines. The outlines produced from Im2pencil do not really seem to us accurate and we are convinced that our implementation can produce better outlines.

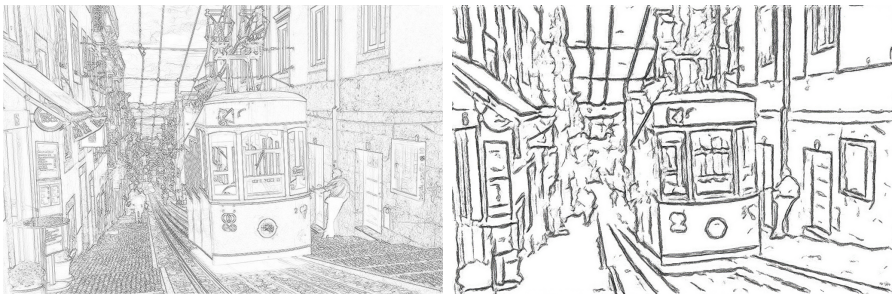
(a) Ours (b) Im2Pencil

Figure 5.2: Textures

On the contrary, Lu et al. [2012] method extends the texture over the size of the image and adjusting the proper tone of the texture so it can be applied into the certain place of the target image. It does not seem that the texture is directly mapped at the surface of a particular shape. Next disadvantage is that we use only one texture pattern for one image. We are convinced that artists use more than one pattern to create a pencil drawing and they apply it in several directions.

We tested the Im2Pencil even for high resolution images. Unfortunately, our machine was not able to produce any pencil drawing, because it could not have allocated enough memory on the GPU. We ran the project on Nvidia Geforce 860M with 2 GB of memory. Our implementation does not crash on runtime exception due to lack of memory even for big images. Moreover, it produces the results even faster.

In summary, based on our results and our research, we assume that neural networks have still a long way to go. Their results did not seem to us so much better than ours. In addition, their time and memory complexity is very high. We do not really see such a big advantage in collection a large amount of data for this purpose. We think that transfering an image to a sketch can be done with much more elegant way.



(a) Ours (b) Im2Pencil

Figure 5.3: Lines

If we look at line images in the fig. 5.2, we will see one significant difference. Li et al. [2019] method did not see that it handled thick lines in any case. They uses eXtended Difference of Gaussians to detect the edges.

xDoG is a feature enchancement algorithm that involves the subtraction of one blurred version of an original image from another Winnemöller [2011]. It is written as:

$$G(\hat{x}, \sigma, I) = \frac{1}{2\pi\sigma^2} \int I(x) e^{-\frac{\|\hat{x}-x\|^2}{2\sigma^2}} dx \tag{1}$$

$$D_x(\hat{x}, \sigma, k, \tau, I) = G(\hat{x}, \sigma, I) - \tau\, G(\hat{x}, k * \sigma, I) \tag{2}$$

$$E_x(\hat{x}, \sigma, k, \tau, e, \psi, I) = \begin{cases} 1, & \text{if } D_x(\hat{x}, \sigma, k, \tau, I) < e \\ 1 + \tanh(\psi * (D_x(\hat{x}, \sigma, k, \tau, I))), & \\ & \text{otherwise.} \end{cases} \tag{3}$$

where $\hat{x}$ is a two-valued coordinate, σ is the standard deviation of distribution, I is the source image, e shifts the detection threshold, τ changes the relative weighting between the larger and smaller Gaussians.

They even present their motivation using this kind of detector. They prepared a test, where they compare xDoG against boundary detector based on structured random forests and against the method for sketch simplification. As their research shows, xDoG is able to handle line thickness and smooth non-outline regions well. From all of the images we had tested on Im2Pencil, we did not see in any case that it handled thick lines properly Winnemöller [2011].

We used the Sobel operator and modified the edges with line kernels. Our implementation can handle even thick lines. But if we have more detailed structure, line kernels produce too many edges. One of these effects can be seen at fig. 5.2 (a), if we look at the pavement of the street.

Both of these methods generate tone images. Even if we show the results in fig. 5.3, we cannot really compare the images. There are applied operations which have completely different goal. We are adjusting tones in three tonal layers, but Im2Pencil uses Guided filter to remove details and smooth the image.

The Guided filter uses the content of a second image, called a guidance image, to influence the filtering. The guidance image can be the source image itself, a different version of the source image, or a completely different image He et al. [2013]. It is denoted as:

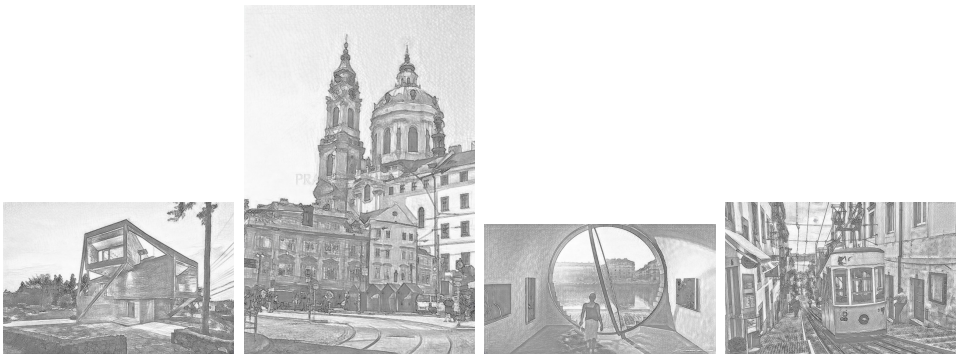
$$q_i = \sum_j W_{ij}(I)p_j \tag{5.1}$$

where i and j are pixel indexes. The filter kernel W_{ij} is a function of the guidance image I and independent of p.

(a) Grayscale (b) Ours (c) Im2Pencil

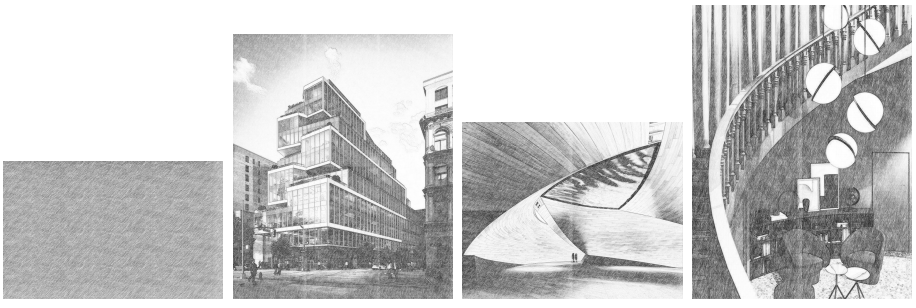
Figure 5.4: Tones

We mentioned that we were not very satisfied with the outlines. We thought that the results would be more impressive, if we superposed the line images from our project with texture images of Im2Pencil. In the fig. 6.5 we can see the results. We used weighted sum to superpose images of Im2Pencil and images from our project. Unfortunately, in the fig. 5.5 we do not see any significant improvements.

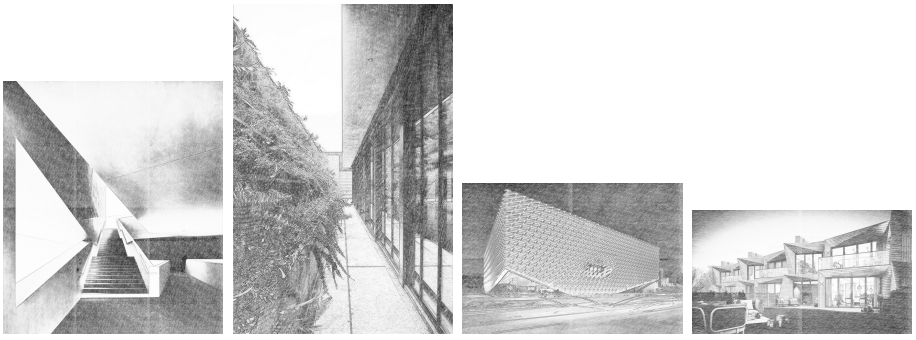


(a) (b) (c) (d)

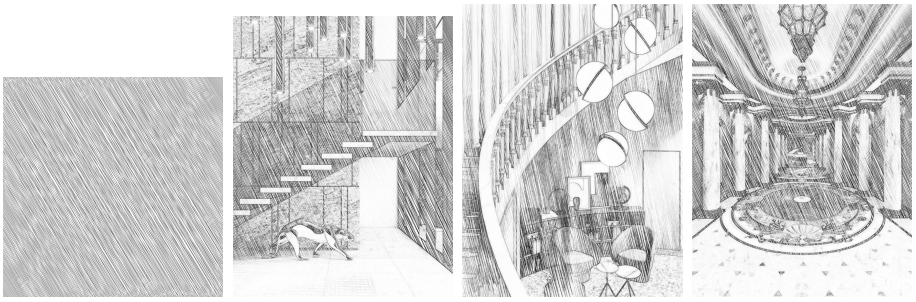
Figure 5.5: Combined



(a) (b) (c) (d)



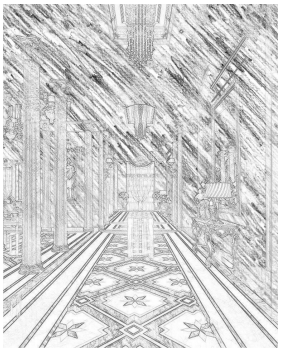
(e) (f) (g) (h)



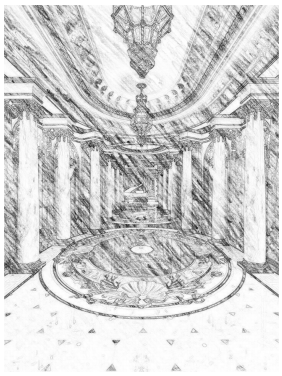
(a) (b) (c) (d)

(a)

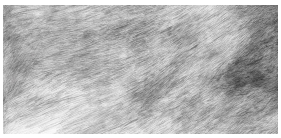
(b)



(c)

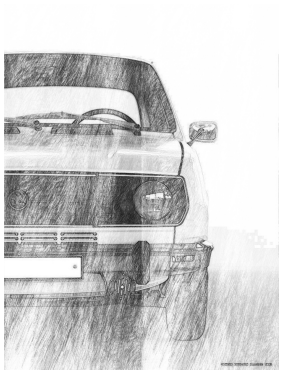


(d)



(e)

(f)



(g)

(h)

Conclusion

The goal of the thesis was to implement a program that procedurally generates a pencil drawing from a natural image. We attempted to describe each step of the algorithm used in our implementation. Then, we introduced how our project is structured and how it works in general.

We presented other possible approaches which are, for example, based on neural networks. In the discussion chapter, we showed several results of both of these projects. We described how these both methods differ.

There are more recent works which propose more advanced algorithms for pencil drawing-like line generation. We could probably combine Lu et al. [2012] method for tone extraction, texture optimalization with some of the line generation algorithms like xDog Winnemöller [2011] or with estimated gradient method proposed by Okawa et al. [2017].

Finally, we have suggested possible improvements of our implementation. Therefore, we described the reader what are the limitations of both of these methods.

Programmer's manual

In this chapter we describe the steps necessary to compile and run all programs mentioned in the attachments part. We include two projects. One is the PencilDrawing which is our core C++ program that transforms an input image into a pencil drawing style, and HtmlGenerator which is C# console application. For both of these projects we generated executables on the Windows platform.

PencilDrawing

As we have mentioned in the Implementation chapter, PencilDrawing takes five input parameters. Path to image, path to pencil texture and three weights, which adjust brightness in three tonal layers. We prepared one example image and pencil texture in the Data folder. Futhermore, our project contains a Visual Studio 2019 solution file, project file with .vcxproj extension and CMakeLists.txt. If the user attempts to start the project, first, he has to set correct paths to OpenCV and Eigen libraries. On the Windows platform these paths usually differ. For linux the CMakeLists contains the environment variables holding the path to all the necessary libraries. If the environment variables are set correctly, program compiles. Finally, if the project is executed succesfully, it will create the Results folder.

HtmlGenerator

For study purposes we designed the HtmlGenerator project to generate a HTML file in which we can find all the results of the PencilDrawing project. It is necessary to include PencilDrawing.exe file, the generated dependencies of the project and opencv_world410.dll to run the program. Futhermore, a file is taken as the input parameter where we specify on each line call arguments of our C++ program. After the HtmlGenerator ran succesfully, it creates folders with the results and the html file. The Application is, unfortunately, not portable to Unix-like systems, because it executes the program with .exe extension which is specific to windows. But HtmlGenerator was developed with the .netcore framework so if we make a little modification to our program, it can be executed with binaries for Unix environment.

Bibliography

Chengying Gao, Mengyue Tang, Xiangguo Liang, Zhuo Su, and Changqing Zou. Pencilart: A chromatic penciling style generation framework. *Computer Graphics Forum*, 37(6):395–409, 2018. doi: 10.1111/cgf.13334. URL `https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.13334`.

Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. A Neural Algorithm of Artistic Style. *arXiv e-prints*, art. arXiv:1508.06576, Aug 2015.

K. He, J. Sun, and X. Tang. Guided image filtering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(6):1397–1409, June 2013. ISSN 0162-8828. doi: 10.1109/TPAMI.2012.213.

Jin Zhou and Baoxin Li. Automatic generation of pencil-sketch like drawings from personal photos. In *2005 IEEE International Conference on Multimedia and Expo*, pages 1026–1029, July 2005. doi: 10.1109/ICME.2005.1521599.

Yongcheng Jing, Yezhou Yang, Zunlei Feng, Jingwen Ye, Yizhou Yu, and Mingli Song. Neural Style Transfer: A Review. *arXiv e-prints*, art. arXiv:1705.04058, May 2017.

Henry Kang, Seungyong Lee, and Charles K. Chui. Coherent line drawing. In *Proceedings of the 5th International Symposium on Non-photorealistic Animation and Rendering*, NPAR '07, pages 43–50, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-624-0. doi: 10.1145/1274871.1274878. URL `http://doi.acm.org/10.1145/1274871.1274878`.

Hyunjun Lee, Sungtae Kwon, and Seungyong Lee. Real-time pencil rendering. In *Proceedings of the 4th International Symposium on Non-photorealistic Animation and Rendering*, NPAR '06, pages 37–45, New York, NY, USA, 2006. ACM. ISBN 1-59593-357-3. doi: 10.1145/1124728.1124735. URL `http://doi.acm.org/10.1145/1124728.1124735`.

Guennadi (Henry) Levkine. Sobel and scharr gradient 5x5 convolution matrices. February 2011.

Yijun Li, Chen Fang, Aaron Hertzmann, Eli Shechtman, and Ming-Hsuan Yang. Im2Pencil: Controllable Pencil Illustration from Photographs. page arXiv:1903.08682, Mar 2019.

Cewu Lu, Li Xu, and Jiaya Jia. Combining sketch and tone for pencil drawing production. In *Proceedings of the Symposium on Non-Photorealistic Animation and Rendering*, NPAR '12, pages 65–73, Goslar Germany, Germany, 2012. Eurographics Association. ISBN 978-3-905673-90-6. URL `http://dl.acm.org/citation.cfm?id=2330147.2330161`.

R. Okawa, H. Yoshida, and Y. Iiguni. Automatic pencil sketch generation by using canny edges. In *2017 Fifteenth IAPR International Conference on Machine Vision Applications (MVA)*, pages 282–285, May 2017. doi: 10.23919/MVA.2017.7986856.

Shuo Sun and Dongwei Huang. Efficient region-based pencil drawing. 05 2019.

Fang Wen, Qing Luan, Lin Liang, Ying-Qing Xu, and Heung-Yeung Shum. Color sketch generation. In *Proceedings of the 4th International Symposium on Non-photorealistic Animation and Rendering*, NPAR '06, pages 47–54, New York, NY, USA, 2006. ACM. ISBN 1-59593-357-3. doi: 10.1145/1124728.1124737. URL http://doi.acm.org/10.1145/1124728.1124737.

Holger Winnemöller. Xdog: Advanced image stylization with extended difference-of-gaussians. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Non-Photorealistic Animation and Rendering*, NPAR '11, pages 147–156, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0907-3. doi: 10.1145/2024676.2024700. URL http://doi.acm.org/10.1145/2024676.2024700.

Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. *CoRR*, abs/1703.10593, 2017. URL http://arxiv.org/abs/1703.10593.

List of Figures

A. Attachments

Source code

A folder with the source code written in Visual Studio 2019. It contains a solution with all the files necessary for compilation and also a CMakeLists to build the project on UNIX-like platforms.

Executables

A folder with executables including other files needed to run them.

Examples

A folder with some images and pencil textures ready to be executed with the project.

The thesis

An electronic version of the thesis as .pdf.