POLITECNICO DI TORINO

Corso di Laurea Magistrale in Ingegneria Informatica
(Computer Engineering)

Master's degree thesis

Creative image processing algorithms for drawing robots: from raster images to vector-based graphics

Supervisors

Prof. Bartolomeo Montrucchio

Prof. Luigi De Russis

Company supervisor - Scribit

Dott. Danilo Ronchi

Candidate

Barbara Munoz Viamonte

Academic Year 2019/2020

DEDICATION

To my Dad Enrique, my Mom Evelyn and my little brother Gabriel who has taught me and still teach me and guide me to be a better person everyday. You guys are the best family ever.

Barbara.

AKNOWLEDGEMENTS

Firstly, I would like to express my sincere gratitude to my thesis supervisor at Scribit Danilo Ronchi for his enthusiasm, patience, helpful information and continuous support and guidance in the realization of this thesis.

Besides, I would like to thank my academic supervisors, Prof. Bartolomeo Montrucchio and Prof. Luigi De Russis for their insightful comments and encouragement.

I also wish to express my sincere thanks to the Politecnico di Torino for accepting me in this graduate program.

I am also grateful the start-up company Scribit, which provided me an opportunity to join their team as a thesis intern, and who made me feel as home giving me access to its facilities.

I would like to thank my cousin Elizeth and my friends for being close to me when I urgently needed to relax and laugh.

Last but not the least; I would like to thank my family: my parents and my brother for supporting me spiritually and throughout writing this thesis and my life in general.

LIST OF CONTENTS

INTRODUCTION

With the advances of mobile communication that went from simple telephony to a complex network of applications and services, driven by the existence of highly evolved devices commonly called "smartphones", entrepreneurs seek to surprise the growing mass of users with creative and inventive developments. Young companies emerge in order to propose these developments to the world market with the hope of having achieved a product which it can fall in love with.

When we look at the new smartphones proposed by the big manufacturers, it is obvious that there is an increasing emphasis on perfecting digital image capture by these devices; for the users the possibility of capturing better images with more resolution and definition at any time has become essential.

In 2018, a project called Scribit began to grow, proposing the creation of a vertical plotter to carry pre-processed images, using specialized software, to be printed on walls, glass and similar materials. The idea is really attractive since it makes available to an specific public the possibility of decorating among others office and restaurants walls with images or writings that can be printed and erased as desired.

In spite of entering into the concept of new technology, integrating together the plotter, an application on the Smartphone and the cloud processing, the product developed lacked of an essential element in order to meet a commercial success. It needed the ability to represent any image, not necessarily specialized, by means of the device. It is where the need emerges - subject of this thesis - the creation of algorithms able to transform and simplify any image in such way that it is designed and interpreted by the plotter in relatively short times and with a quality and definition attractive to the users.

This thesis shows the features, limitations and advantages of the system called Scribit, the premises that had to be taken into consideration based in these features,

and the algorithms developed to meet the company`s requirements in order to make the product available and attractive to any user.

Many preliminary studies were realized to achieve a balance and attunement between what is attractive and practical to the client and optimal, functional, flexible, realistic and possible to Scribit.

Every system has its pros and cons, which must be understood and studied to get the most out of it and its use, thus helping to create and develop special and adapted features for its use. An important factor kept in mind during the development of the thesis was the execution time. The algorithms developed meet the following requirements: reduced fast processing and image manipulation time and design.

Algorithms sought were fast processing and whose manipulating time was reduced. In other words, a compromise was always sought between obtaining a visibly pleasing and innovative result and analysis and design time within the stipulations and acceptable to users. The Scribit robot, as explained in the next chapters, uses markers to design, which require paying particular attention when designing is executed due to its restricted markers tips availability. This feature along with the language selected to be used for the representation of the graphic pattern, SVG, required the creation of algorithms based on continuous lines, figures, geometric elements and curves among other.

The SVG language gives us a great advantage. It is a language that allows representation in an understandable and compact way. It is a scalable and resizable vector language and allows a greater definition at reduced sizes.

CHAPTER I: CONTEXTS

1.1 Objectives

The aim of this project thesis is the creation, implementation and validation of algorithms that convert images into graphic patterns in order to improve the user experience, attract new clients and offer a new service by providing innovative and creative ideas and, at the same time, expanding the use of the Scribit robot.

The generation of these algorithms helps to massify the product, allows a simplified use to people with a few knowledge of graphic subject and understanding of illustration, graphic design, dedicated software, advanced designed tools and methods of image representation.

These algorithms hereinafter called “filters” are accessible and available through the mobile application and website. The user can obtain from images, designs or pictures various artistic effects that can be successively designed by Scribit on any whiteboard, plaster or glassy surface.

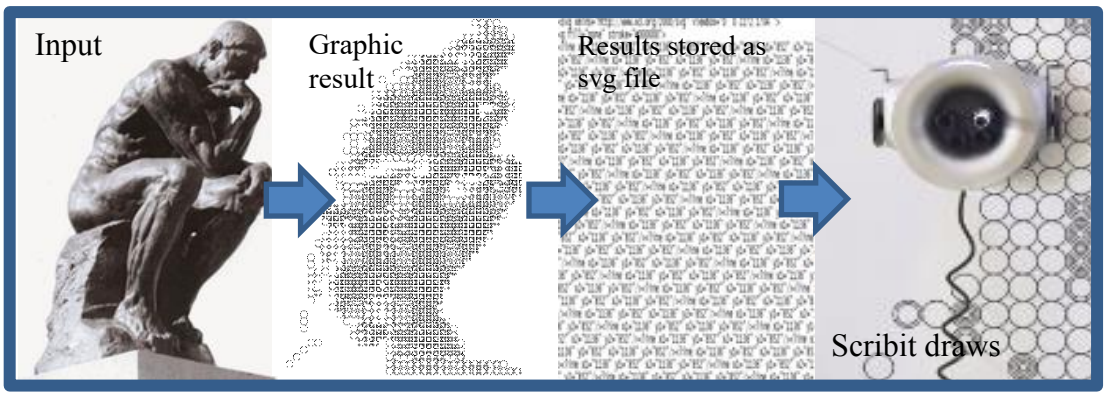

Figure 1

1.2 Scribit

Scribit is an intelligent write-and-erase robot able to draw any content on any vertical surface and operate between 4 combinations of color. It is based on the idea of a vertical plotter, enriched with the latest generation software that predisposes it to mass use. It transforms walls into interactive whiteboards and is capable to replicate any image and text to give a new look and perspective to shops, offices, homes and restaurants. Conceived by the international design and innovation studio Carlo Ratti Associati; this robot has dimensions of 6.6 inches x 3.15 inches; it is coated on the outside in aluminum and equipped with a shape that allows it to climb on the walls.
Two nails on the walls, a power outlet and an internet line are needed in order to start using Scribit.

1.2.1 History

Thanks to Carlo Ratti, Italian architect, engineer, inventor and teacher and Pietro Leoni, architect, the Project starts at the MIT Senseable City Lab (a research group that explores how the new technologies are changing the way people understand, design and explore the cities). Scribit development started in September 2017. It was funded on IndieGogo and KickStarter in June 2018 with a crowfinding campaing incubated by Makr Shakr. Led by Andrea Bulgarelli (CTO) and Andrea Baldereschi (CMO), it has raised more than $2 million and has managed to be known worldwide. Scribit is the result of a teamwork formed by marketing experts, engineers and product designers.

1.2.2 How it works

The system is composed of a hardware part; the vertical plotter. The mobile application and the server constitute the software part.

Hardware: Mechanics and assemblage

This robot has a circular shape or as defined by Kickstarter an "ear-like structure" with wheels on the sides that allow it to move through the walls.

Figure 2

kickstarter.com

It is based on a magnesium chassis properly designed to allow the free movement and oscillation of the power cord, a powerful motor, a drum, an electronic board, and a useful small ceramic heater which allows the erasing process. The drum has 4 holes and you can use from 1 to 4 markers. The plane containing the holes is a toothed wheel and it acts as a crown in a crown-pinion

system, in which the real pinion is attached to the shaft of the central stepper motor, which allows rotation and hence the functioning of the system.

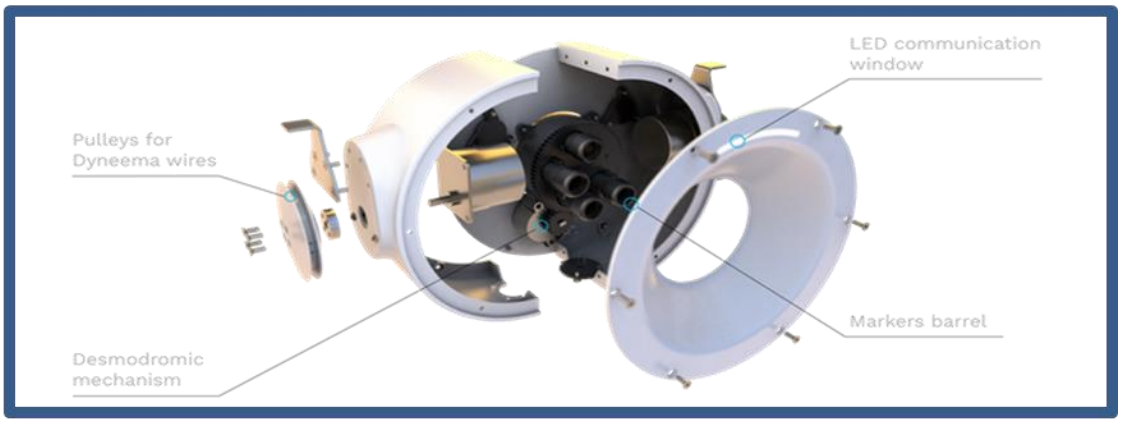

Figure 3

kickstarter.com

To ensure the marker ink is erased regardless of the room temperature conditions or the place where Scribit is being used, the ceramic heater must reach high as 130 degrees Celsius. This process is only activated when the erasing process is being carried executed. The relevant system parts are the isolated materials that fulfill the function of dissipating heat.

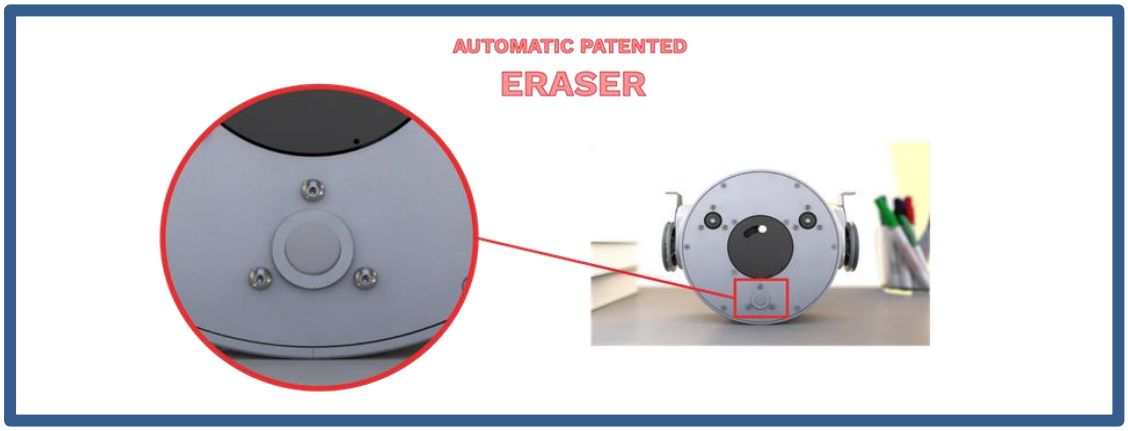

Figure 4

kickstarter.com

The electronic board was designed, created and developed ad hoc. It includes two processors capable of containing dedicated software. At its upper part the robot has an illuminated band with LEDs that modifies its color and frequency, allowing the user to know the Scribit current state.

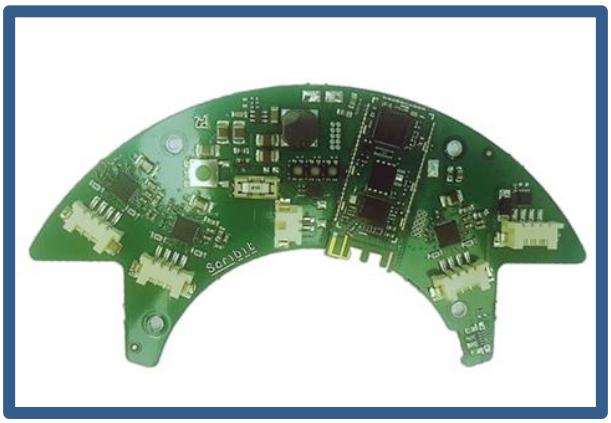

Figure 5

Figure 6

Software and intelligence

The mobile application is available for various platforms and is responsible for managing the installation, calibration, design selection and printing process, among other functions. Everything is based on a centralized architecture on servers, in which the user will have the possibility to link to his account more than one Scribit potentially installed anywhere in the world.

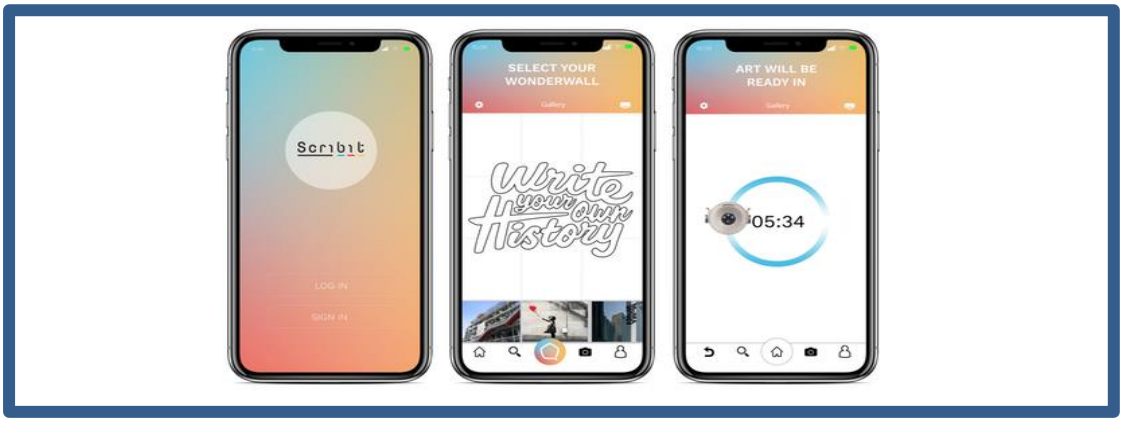

Figure 7

kickstarter.com

Following the step-by-step installation instructions the user will be able to register, create a profile and connect the device to the mobile application. Once the installation procedure is finished, the calibration phase is carried out (nowadays is done manually). Like many devices, Scribit needs to be calibrated so the user must enter the information necessary to complete this task. The wall material and dimensions are all the information the robot needs to execute a number of movements that will allow it to be well located on the selected surface.

Now the user is ready to select the design, graphic or widget he wants to reproduce on the wall. The application offers a wide variety of options the user

may pick as desired. Otherwise the user can load its own designs. Before starting printing, the user chooses colors, size (small, medium or large compared to the drawable area) and may also (in case walls are transparent) apply a mirror effect to the graphics. When the printing process has begun, the conversions from various input formats to GCODE are done through the use of remote functions, explicitly called on the platform Google cloud. The app also communicates with a NodeJS framework that exposes REST APIs used both to retrieve information related to users' graphics and preferences and for communication with the plotters linked to each one of them, such as: the estimated time and distance in meters that every marker will travel. The list is sent to the server which forwards it to the plotter via MQTT broker by changing its status, visible to the user by changing the LED animation and from the presence of the print interface on the application. From this moment it will be possible to pause or to stop printing completely. Once the process is finished, Scribit will return to the start position (the position decided in the calibration phase). Consequently, its status changes again.

If the user desires to start the cancellation routine, he simply return to the screen that controls the plotters synchronized with the user account.

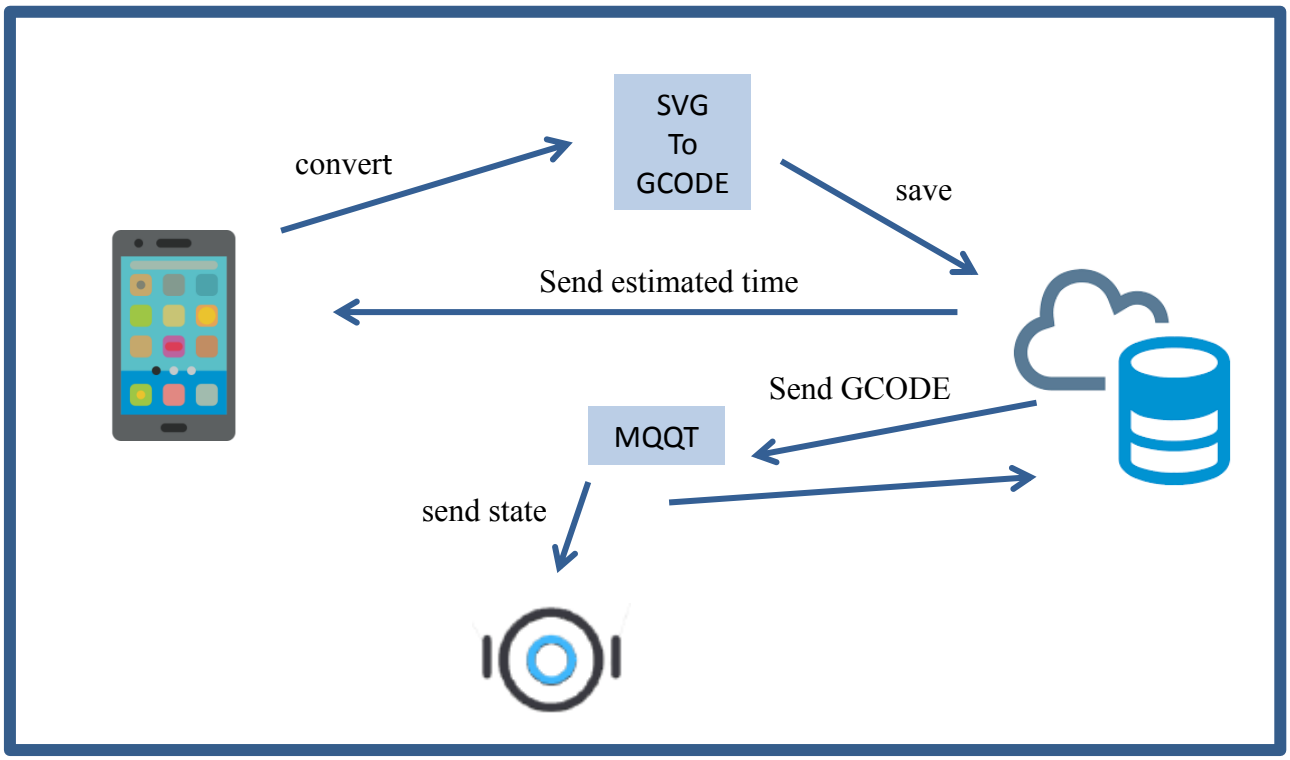

Figure 8

Printing process via app.

CHAPTER II: THEORICAL ASPECTS

2.1 DIGITAL IMAGE PROCESSING

The concept of image processing begins in the 20s in the newspaper industry in order to improve the quality of the images transmitted by cable. The notion of digital image processing instead comes to life with the development and expansion of digital computers. In fact, digital images demand plenty of computational power, performance and storage, that is why progress in the field has always been dependent on the development of digital computers and on the supporting technologies including display, transmission and data storage.

In the early 1960s the first computers powerful enough to accomplish significant image processing tasks emerged. The beginning of what we call digital image processing today can be tracked to the availability of those machines and to the start of the space program in the course of that period. The union of those two developments brought into focus the potential of digital image processing concepts.

The digital image processing impacts almost every technical and technological sector. There are numerous and varied areas of application:

- Nuclear medicine: Bone scans obtained by using gamma-ray imaging. These types of images are used to locate sites of bones pathology, such as infections or tumors. Radio waves are also use in MRI (Magnetic Resonance Imaging).
- Medical diagnostics and angiography: Using X-rays that designates an electromagnetic radiation, invisible to the human eye, capable of passing through opaque bodies and printing photographic films. Current digital systems allow obtaining and viewing the radiographic image directly on a computer without printing. The addition of a contrast medium allows enhancing contrast of the blood vessels and enables the specialist to see any irregularities or blockages.
- Fluorescence microscopy: It is an extraordinary method to inspect materials that can be made to fluoresce, either in their natural form or when treated with

applications such as lasers, biological imaging, lithography, industrial inspection and astronomical observation.

- Light microscopy used in pharmaceutical, microinspection and material characterization: It uses the infrared band in the conjunction with visual imaging.
- Radar: Application of imaging in the microwave band. The unique feature of imaging radar is its ability to collect data over virtually any region at any time, regardless of weather or ambient lighting conditions. It is sometimes the only way to explore unreachable regions on the surface of the Earth.
- Geological exploration: The most important commercial applications of image processing in geology are in mineral and oil exploration.

Digital Image processing is the utilization of a computer to process digital images through algorithms. Digital images are constituted of a finite number of elements having a particular value and position. These elements are known as pixels and are the smallest controllable units of a picture represented on the screen.

The pixel's sequence marks the coherence of the information presented, being a coherent matrix of information for digital use as a whole. In bitmap images, or graphic devices, each pixel is encoded by a set of bits of a certain length (the color depth); for example, a pixel with one byte (8 bits) can be encoded, so that each pixel supports up to 256 color variations, from 0 to 255. The images called "true color" usually use three bytes (24 bits) to define the color of a pixel, an image in which 32 bits are used to represent a pixel has the same amount of colors as that of 24 bits, since the other 8 bits are used for transparency purposes.

In order to visualize, store and process the numerical information represented in each pixel, it is important to know, in addition to the depth and brightness of the color, the color model to be used. For example, the RGB (Red-Green-Blue) color model allows us to create a color composed of the three primary colors according to the additive mixing system. Thus, the final color will depend on the proportion of each RGB component present in the pixel.

In this model 8 bits are commonly used when representing the proportion of each of the three primary color component. A component with 0 value means that it does not intervene in the mixture; and a component with a value of 255 (28 - 1) means that it does intervene by giving the maximum of that tone. Intermediate values provide the corresponding intensity. Most of the devices used with a computer use the RGB model.

An image can be processed in different domains. The spatial domain refers to the image plane itself, so image processing methods in this category are based on direct manipulation of pixels in an image. On the other hand, image processing in a transform domain involves first transforming an image into the transform domain, doing the processing there, and obtaining the inverse transform to bring the results back into the spatial domain.

2.1.1 Digital image fundamentals

Light is a particular type of electromagnetic radiation that can be sensed by the human eye. The colors that humans perceive in an object are determined by the nature of the light reflected from the object.

Monochromatic light

Light that lacks of color is called monochromatic (or achromatic) light. The only attribute of monochromatic light is its intensity or amount. Since the intensity of monochromatic light is perceived to change from black to gray and then to white, the term gray level is commonly used to denote monochromatic intensity. The range of measured values of monochromatic light from black to white is usually called the gray-scale, and monochromatic images are frequently referred to as gray-scale images **[1]**.

Chromatic light

Chromatic (color) light spans the electromagnetic energy spectrum and in addition to frequency, three basic quantities are used to describe the quality of a chromatic light source: luminance, radiance and brightness **[2]**.

Radiance

Is a measure of the amount of electromagnetic radiation leaving or arriving at a point on a surface. It is the total amount of energy that flows from the light source, and it is usually measured in watts (W) **[3].**

Luminance

Gives a measure of the amount of energy an observer perceives from a light source **[4]**.

Brightness

Is a subjective descriptor of light perception that is practically impossible to measure. It embodies the achromatic notion of intensity and is one of the key factors in describing color sensation **[5]**.

2.1.2 Basic intensity transformation functions

Intensity transformations are within the easiest image processing techniques. There are three elementary types of functions used for image improvement:

1. Linear: Negative and identity transformations.
2. Logarithmic: Log and invers-log transformations.
3. Power-law.

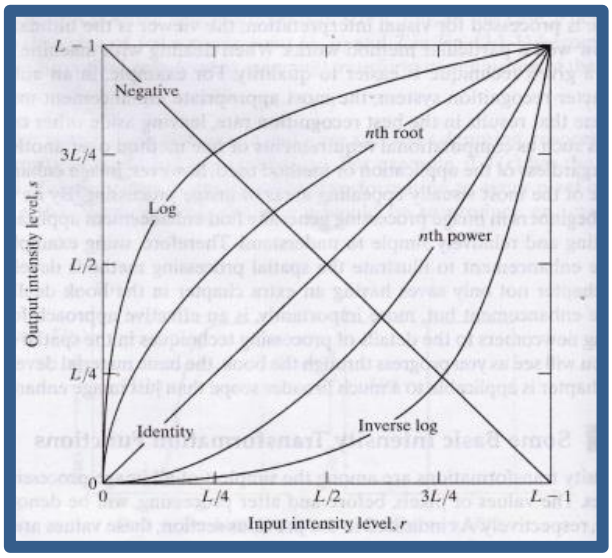

Figure 9

Gonzalez, Woods Digital Image Processing: Third Edition

Image negatives

The idea of this transformation is to reverse the order from white to black. It is appropriated for enhancing gray or white level detail embedded in dark regions. To obtain the negative of an image with intensity levels in the range [0, L-1] the follow expression needs to be apply:

$$s = L - 1 - r$$

where **r** and **s** are respectively the values of the pixels before and after processing.

Log transformations

The common formula of the log transformation is

$$s = c \log (1 + r)$$

where **r** and **s** are respectively the values of the pixels before and after processing and c is a constant.

This type of processing is used to compress the higher-level values while expanding the values of dark pixels. The log functions used in this transformation has the significant property of compressing the dynamic range of images with considerable variations in pixel values.

Power-law (Gamma) Transformations

Have the form

$$s = c\, r^{y}$$

where **c** and **y** are positive constants.

Power-law transformations are effective and beneficial in general-purpose contrast manipulation.

Piecewise-linear transformation functions

Contrast stretching, intensity-level slicing and bit-plane slicing are functions used to expand intensity levels range, highlight a specific range of intensities and highlight the contributions made to total image appearance by specific bits respectively.

2.1.3 Image enhancement and filtering

It is the set of techniques included in the preprocessing of images whose fundamental objective is to obtain, from an original image, another end whose

result is more suitable for a specific application, improving certain characteristics of the same that makes it possible to carry out processing operations on it.

The main objectives pursued with the application of filters are:

- Smooth the image: reduce the amount of intensity variations between neighboring pixels.
- Eliminate noise: eliminate those pixels whose intensity level is very different from that of their neighbors and whose origin may be both in the process of acquiring the image and in the transmission.
- Enhance borders: highlight the edges that are located in an image.
- Detect edges: detect pixels where there is a sharp change in the intensity function.

Therefore, filters are considered as those operations applied to the pixels of a digital image to optimize it, emphasize certain information or achieve a special effect on it. The filtering process can be carried out on the frequency and / or space domains.

Filtering in the frequency domain

Frequency filters process an image by working on the frequency domain in the Fourier Transform of the image. For this, it is modified following the corresponding Convolution Theorem:

1. Fourier Transform is applied.
2. It is subsequently multiplied by the filter function that has been chosen.
3. Re-transform it into the spatial domain using the Fourier Inverse Transform.

Convolution Theorem (frequency):

$$G(u, v) = F(u, v) * H(u, v)$$

Where ***F (u, v)*** is the Fourier transform of the original image and ***H (u, v)*** is the frequency attenuator filter.

Since the multiplication in Fourier space is identical to the convolution in the spatial domain, all filters could, in theory, be implemented as a spatial filter.

Types:

1. Low pass filter: attenuates high frequencies and keeps the lows unchanged. The result in the spatial domain is equivalent to that of a smoothing filter, where the high frequencies that are filtered correspond to the strong changes in intensity. It manages to reduce noise by softening existing transitions.
2. High pass filter: attenuates low frequencies keeping high frequencies invariable. Since the high frequencies correspond in the images to sudden changes in density, this type of filters is used, because among other advantages, it offers improvements in the detection of edges in the spatial domain, since these contain a large amount of said frequencies. Reinforce the contrasts found in the image.
3. Band pass filter: attenuates very high or very low frequencies while maintaining a midrange band.

Executing the filtering process in the frequency domain brings many advantages such as its simplicity to implement the method, fast when using the convolution theorem, easy association of the concept of frequency with certain characteristics of the image and flexibility in the design of filtering solutions. Nonetheless noise cannot be completely eliminated and it requires knowledge in several fields to develop an application for image processing.

Spatial filtering

Spatial domain procedures are more efficient computationally and demand less processing resources to carry out and apply.

The spatial domain process is denoted by:

$$g(x,y) = T[f(x,y)]$$

where ***g(x,y)*** is the output image, ***f(x,y)*** is the input image, and **T** is an operator on f defined over a neighborhood of point ***(x,y)***.

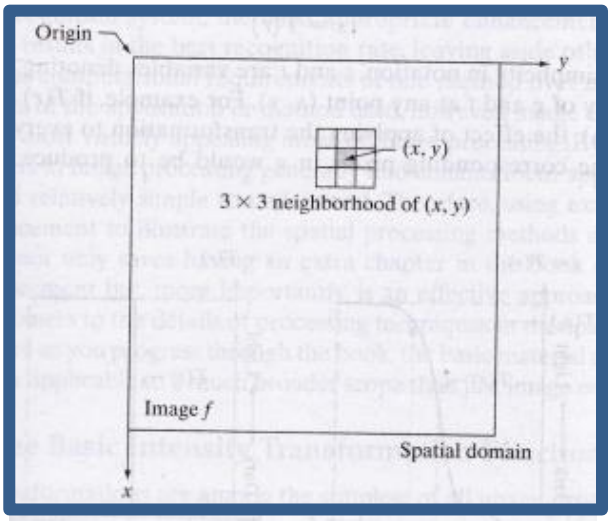

Figure 10

Gonzalez, Woods Digital Image Processing: Third Edition

The process of spatial filtering consists of moving the origin of the neighborhood from pixel to pixel and of applying the operator **T** to the pixels in the neighborhood to yield the output at that location. Thus, for any specific location ***(x,y),*** the value of the output image **g** at those coordinates is equal to the result of applying **T** to the neighborhood with origin at ***(x,y)*** in ***f***.

Types:

1. High pass filter (attenuation): intensifies details, edges and high frequency changes, while attenuating areas of uniform hue. This allows a better subsequent identification of the objects that are in the image, since the brightness becomes greater in the areas with higher frequencies, while the areas of low frequencies are darkened.
2. Low pass filter (smoothing): used to eliminate noise or small details of little interest since it only affects areas with many changes. The cutoff frequency is determined by the size of the core and its coefficients. Various cores are used; Average, Low pass in frequency, Medium, Median, Gaussian: approach to the Gaussian distribution.
3. Border enhancement by offset and difference: subtracts from the original image a shifted copy of it. Thus, it is possible to locate and highlight the existing edges and that they want to obtain according to the core model applied.
4. Edge detection and contour filters (Prewitt and Sobel): it is focused on the intensity differences that are given pixel by pixel. They are used to obtain the contours of objects and thus classify the existing forms within an image.
5. Edge enhancement using Laplace: enhances the edges in all directions (the results obtained can be considered as a sum of those obtained after applying all models of the previous type). The second derivative is used, which allows us to obtain better results.
6. Highlight edges with directional gradient: work with the intensity changes between adjacent pixels and is used to highlight more precisely the edges that are located in a given direction.

Some tasks require the application of several complementary techniques in order to achieve an acceptable result. So we can think of using many of these techniques together to achieve better effects, taking into account the order of execution and use of them.

2.2 SCALABLE VECTOR GRAPHICS

SVG is a widely-deployed royalty-free graphics format developed and maintained by the W3C (World Wide Web Consortium) SVG Working Group **[6]**. Scalable Vector Graphics (SVG) is an Extensible Markup Language (XML)-based vector image format for two-dimensional graphics with support for interactivity and animation. SVG images and their behaviors are defined in XML text files. This means that they can be searched, indexed, scripted, and compressed. As XML files, SVG images can be created and edited with any text editor, as well as with drawing software. All major modern web browsers have SVG rendering support. **[7]**.

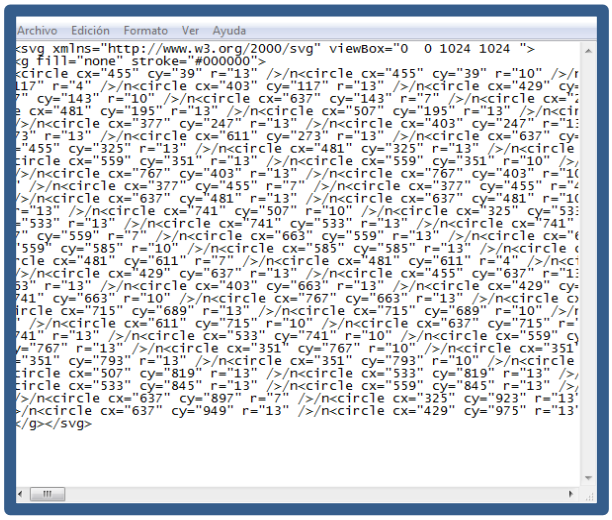

Figure 11

Colors can be applied to all visible SVG elements, either directly or via fill, stroke, and other properties. SVG allows for three types of graphic objects: vector graphic shapes (e.g., paths consisting of straight lines and curves), images and text **[8]**.

3.2.1 Paths

A path represents the outline of a shape which can be filled or stroked. Represent the geometry of the outline of an object, defined in terms of *moveto* (set a new current point), *lineto* (draw a straight line), *curveto* (draw a curve using a cubic Bézier), arc (elliptical or circular arc) and *closepath* (close the current shape by connecting to the last *moveto*) commands. A path is defined by including a 'path' element on which the d property specifies the path data **[9]**.

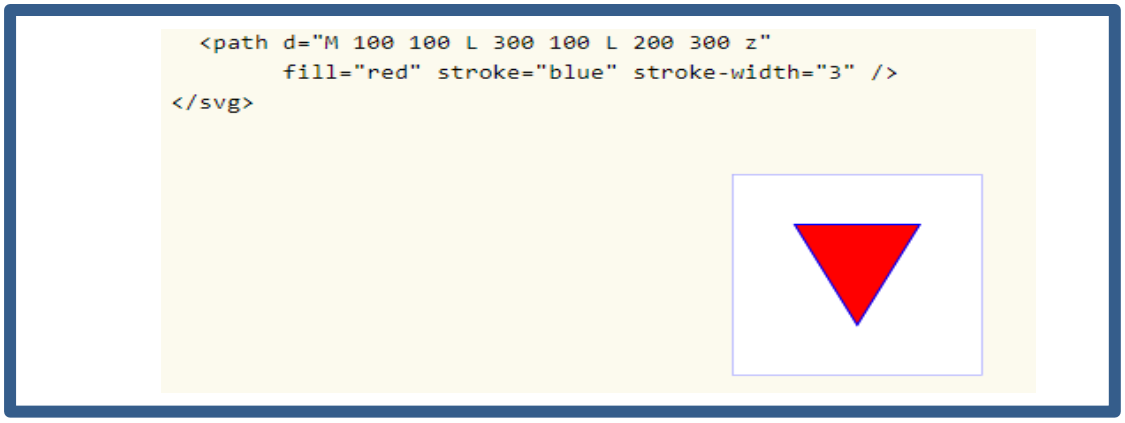

Figure 12

W3C (2019) "Paths"

3.2.2 Basic shapes

It is a graphic element that is defined by some combination of straight lines and curves. Specifically: 'circle', 'ellipse', 'line', 'path', 'polygon', 'polyline' and 'rect' **[10].**

SVG contains the following set of basic shape elements:

- Rectangles (created with the ‘rect’ element): The ‘rect’ element defines a rectangle which is axis-aligned with the current user coordinate system.

Rounded rectangles can be achieved by setting non-zero values for the rx and ry geometric properties.

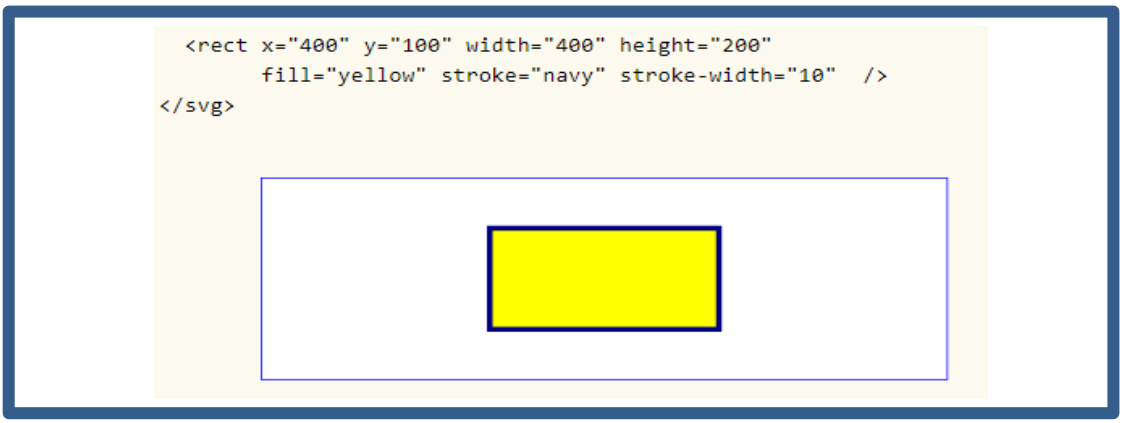

Figure 13

W3C (2019) “Shapes”

- Circles (created with the ‘circle’ element): The ‘circle’ element defines a circle based on a center point and a radius.

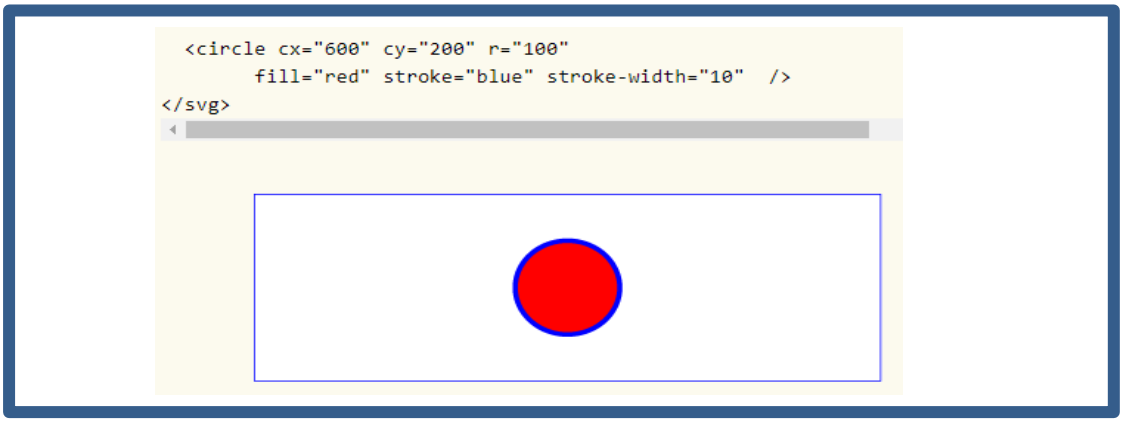

Figure 14

W3C (2019) “Shapes”

- Ellipses (created with the ‘ellipse’ element): The ‘ellipse’ element defines an ellipse which is axis-aligned with the current user coordinate system based on a center point and two radius.

```
  <ellipse transform="translate(900 200) rotate(-30)"
        rx="250" ry="100"
        fill="none" stroke="blue" stroke-width="20"  />

</svg>
```

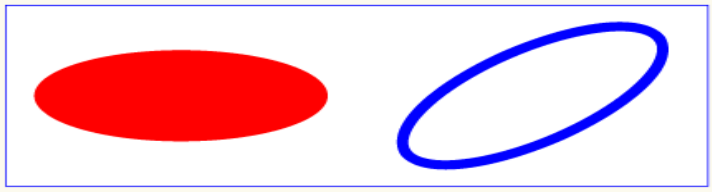

Figure 15

W3C (2019) “Shapes”

- Straight lines (created with the ‘line’ element): The ‘line’ element defines a line segment that starts at one point and ends at another.

```
<g stroke="green" >
  <line x1="100" y1="300" x2="300" y2="100"
          stroke-width="5"  />
  <line x1="300" y1="300" x2="500" y2="100"
          stroke-width="10"  />
  <line x1="500" y1="300" x2="700" y2="100"
          stroke-width="15"  />
  <line x1="700" y1="300" x2="900" y2="100"
          stroke-width="20"  />
  <line x1="900" y1="300" x2="1100" y2="100"
          stroke-width="25"  />
 </g>
</svg>
```

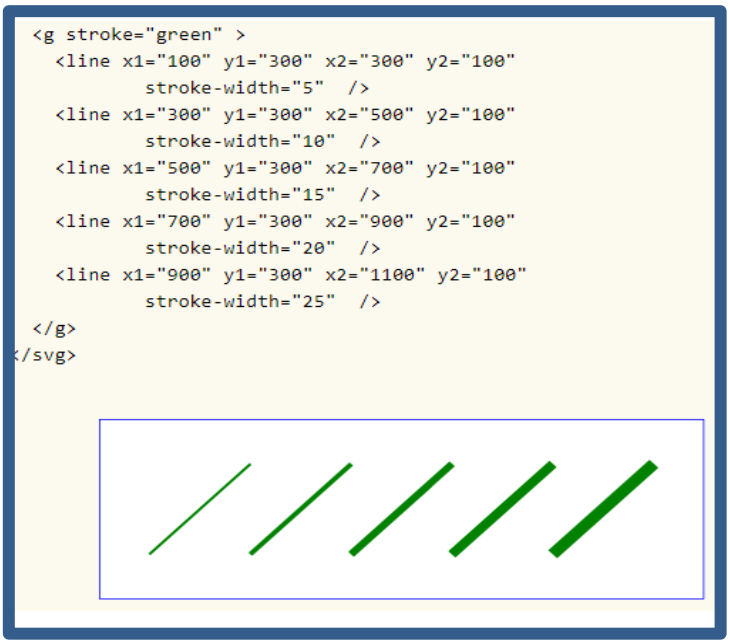

Figure 16

W3C (2019) “Shapes”

- Polylines (created with the 'polyline' element): The 'polyline' element defines a set of connected straight line segments. Typically, 'polyline' elements define open shapes.

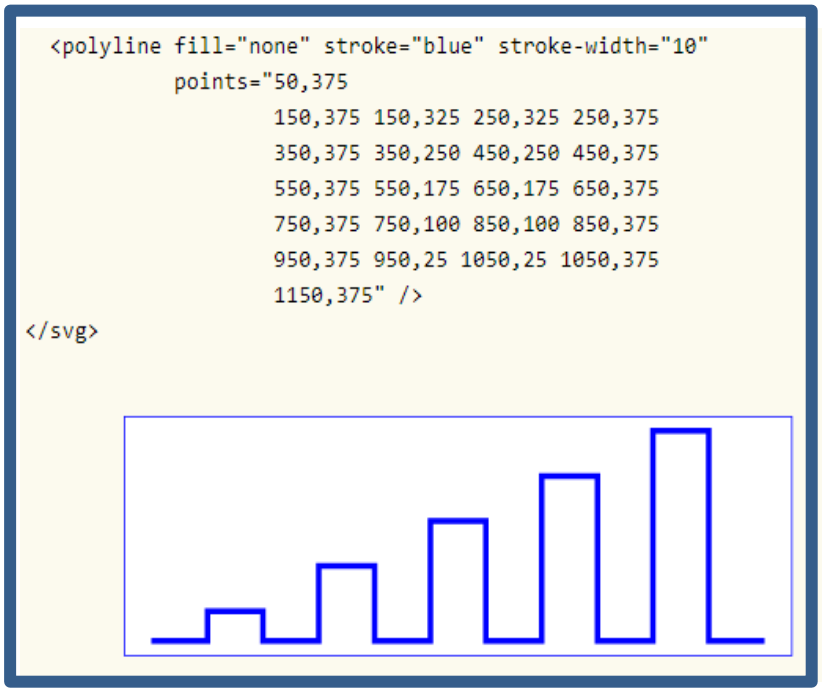

Figure 17

W3C (2019) "Shapes"

- Polygons (created with the 'polygon' element): The 'polygon' element defines a closed shape consisting of a set of connected straight line segments.

```
<polygon fill="red" stroke="blue" stroke-width="10"
          points="350,75  379,161 469,161 397,215
                  423,301 350,250 277,301 303,215
                  231,161 321,161" />
<polygon fill="lime" stroke="blue" stroke-width="10"
          points="850,75  958,137.5 958,262.5
                  850,325 742,262.6 742,137.5" />
/svg>
```

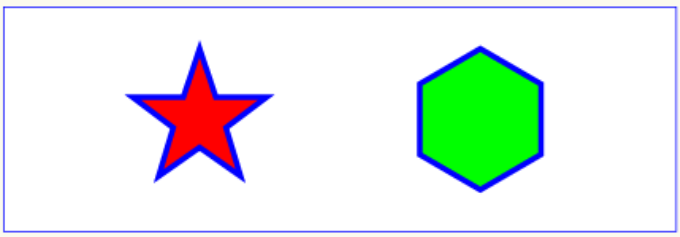

Figure 18

W3C (2019) "Shapes"

Mathematically, these shape elements are equivalent to a 'path' element that would construct the same shape. The basic shapes may be stroked, filled and used as clip paths. All of the properties available for 'path' elements also apply to the basic shapes.

3.2.3 Text

SVG's 'text' elements are rendered like other graphics elements **[11].** SVG text supports advanced typographic features including choice of typeface, use of discretionary, historical, or stylistic ligatures, text decorations (underlining, over-lining, etc.). It also supports international text processing needs such as:

- Horizontal and vertical orientation of text.
- Left-to-right or bidirectional text.

- Complex text layout where: there is not always a one-to-one correspondence between characters and glyphs, characters may change shape depending on location, and characters may be reordered or combined depending on context.
- Glyph alignment to different baselines.

Multi-language SVG content is possible by substituting different text strings based on the user's preferred language.

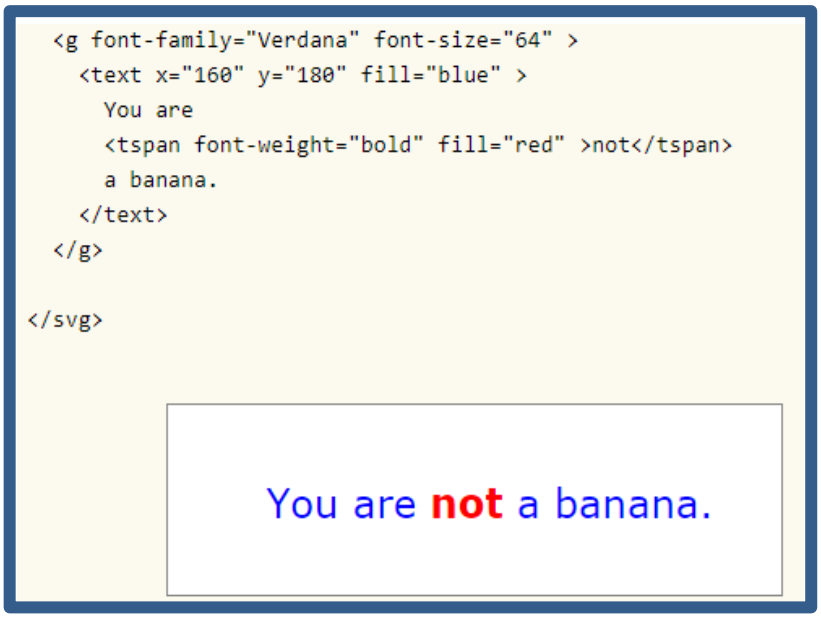

Figure 19

W3C (2019) "Text"

2.3 DELAUNAY TRIANGULATION

Delaunay's triangulations have important relevance and used in the field of computational geometry, especially in 3D computer graphics. It is named after Boris Delaunay for his work on this topic from 1934 **[12]**.

Delaunay triangulation is a network of convex () and related () triangles that meets Delaunay's condition.

Delaunay's condition explains that the circumscribed circumference of each triangle of the network must not contain any vertex of another triangle.

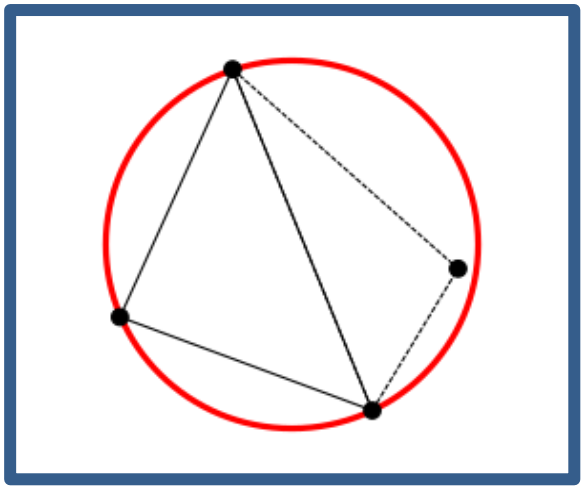

Figure 20

Vertex completely inside the circumscribed circumference. Delaunay's condition is not met. Wikipedia.

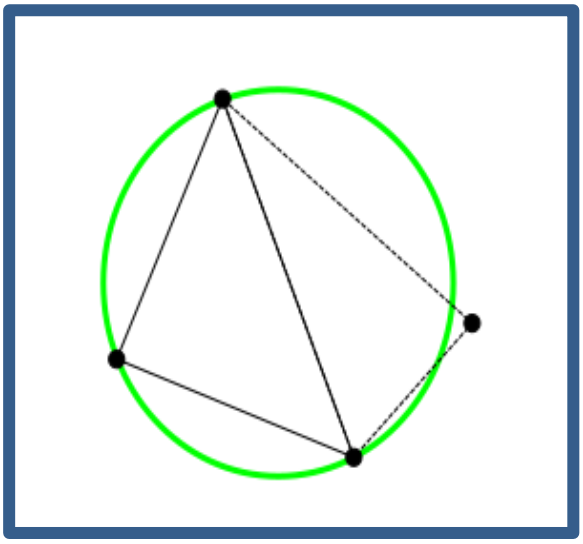

Figure 21

Vertex on the outside of the circumscribed circumference. Delaunay's condition is met. Wikipedia.

3.3.1 Properties

Delaunay triangulation of a set of points meets the following properties:

- The union of all simpleces[1] in the triangulation is the convex hull of the points.
- The minimum angle within all triangles is maximized, that is, to avoid obtaining results with angles that are too acute. This property is practical in computational geometry because it prevents rounding errors that may appear when performing calculations with arbitrary triangulations where very small angles may appear.
- Gabriel's graph[2] is a sub graph of the edges of the Delaunay triangulation; all the edges of Gabriel's graph belong to some triangle of the triangulation.

[1] Generalization of the notion of a triangle. https://en.wikipedia.org/wiki/Simplex

[2] The Gabriel graph of a set of points in the Euclidean plane expresses one notion of proximity or nearness of those points. https://en.wikipedia.org/wiki/Gabriel_graph

- The nearest neighbor's graph is a sub graph of the edges of the Delaunay triangulation; all the edges of the nearest neighbor's graph belong to some triangle of the triangulation.
- Delaunay's triangulation and the Voronoi diagram[3] of a series of points are dual graphs, so the construction of one is trivial from the other.
- In a graph constructed from the edges of the Delaunay triangulation, the shortest path between two points will never be greater than $\frac{4\pi}{3\sqrt{3}} \approx 2.418$ times the Euclidean distance between them.

3.3.2 Algorithms to calculate the Delaunay triangulation

There are several possible strategies to calculate Delaunay triangulation from a set of entry points:

- Brute force algorithm: Consists of generating all possible combinations of three points of the input set, and checking if any other point of the input set is inside the circumference that passes through said list of points. If the circumference does not contain any points, we can ensure that the list forms a triangle that belongs to the Delaunay triangulation.
- Edge Turn Algorithms (Flipping): It begins by creating any triangulation of the entry points and traverses all internal edges by checking if the pair of triangles adjacent to the edge meets the Delaunay condition in isolation. Otherwise, an edge rotation operation is applied, so that the Delaunay condition is gradually introduced into the triangulation. Unfortunately, the process can take O (n2) edge turns and only its convergence for 2-dimensional triangulations is assured.

[3] A Voronoi diagram is a partition of a plane into regions close to each of a given set of objects. https://en.wikipedia.org/wiki/Voronoi_diagram

- Bowyer-Watson algorithm: Is an incremental method where vertices are added to a trivial Delaunay triangulation that is corrected at each step to maintain Delaunay's condition. The selection of the following vertex has a great influence on the execution time of the algorithm, that is why there are several possibilities; the use of a tree, ordering by a coordinate or incidentally.
- Sweepline o sweep algorithms: It follows a similar principle to incremental construction, it builds a small part of the final triangulation and then continue adding vertices in the order in which they are swept by a line until the triangulation is complete.
- Divide and conquer: This algorithm divide the set of points into two parts of equal size, calculate the Delaunay triangulation for each part individually and then gather the two triangulations correcting the errors.

2.4 Bèzier Curves

A Bézier curve is a particular parametric curve, which has great use and application in computer graphics **[13].**

2.4.1 History

The Bézier curves were first published in 1962 by the French engineer Pierre Bézier and later, working in the Renault, he used them abundantly in the design of the different parts of the car. The curves were developed by Paul de Casteljau using the algorithm that bears his name. It is a numerically stable method to evaluate the Bézier curves.

$$\mathbf{B}(t) = \sum_{i=0}^{n} \binom{n}{i} (1-t)^{n-i} t^i \mathbf{P}_i$$

Figure 22

Explicit definition of a Bezier curve

2.4.2 Types, uses and applications

There are many curves between two points. Defining shapes geometrically is not too complicated. A point on the plane is defined by coordinates; Point A has coordinates (x1, y1) and a point B corresponds to (x2, y2). To draw a line between the two points, it is enough to have knowledge of their position. If rather than connecting them with a line they join with a curve, the elements of a Bézier curve arise; the points are called “nodes” or "anchor points". The

shape of the curve is defined by invisible points in the drawing, called "control points".

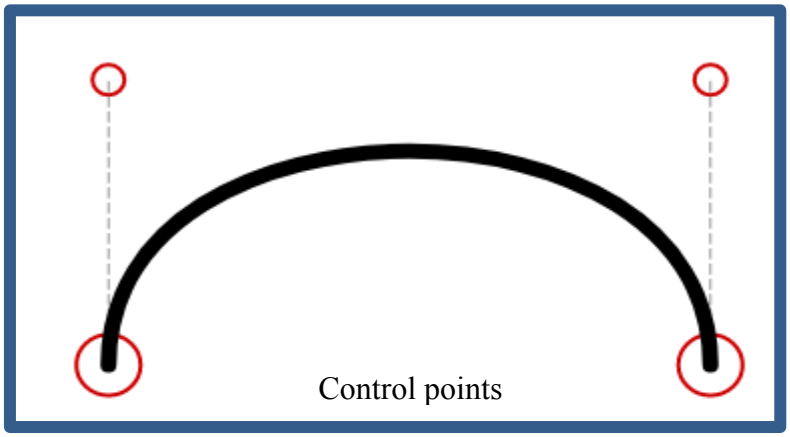

Figure 23

Bèzier Curve

A simple straight path corresponds to a Bézier curve of degree 1 (or linear). Bezier curves of degree greater than 1 are extraordinarily simple to create curved paths between two points. To construct them, one or more points are interpolated between the ends. The more points we interpolate, the more degree (and control possibilities) the curve will be. Simply interpolating one or two control points (Bézier curves of grade 2 and 3) very acceptable results are obtained for a wide variety of situations. The vast majority of vector graphics use cubic Bezier curves.

In vector graphics, Bézier curves are used to model smooth curves that can be scaled indefinitely. "Paths", as they are commonly referred to in image manipulation programs are combinations of linked Bézier curves**[9]**. Bézier curves are also used in the time domain, particularly in animation, user interface design and smoothing cursor trajectory in eye gaze controlled interfaces. Its ease of use has standardized it in graphic design, also extending

to vector animation programs and photo retouching software where it is used to create strokes and closed shapes. This curves find also application in the robotic field where are used to define motion.

CHAPTER III: SOFTWARE DEVELOPMENT

3.1 Triangles filter Algorithm

From an input image, this algorithm creates output data which defines an image described by equilateral triangles; that means triangles define the image (based in the Delaunay Triangulation).

The quantity of triangles designed varies according to the pixels intensity.

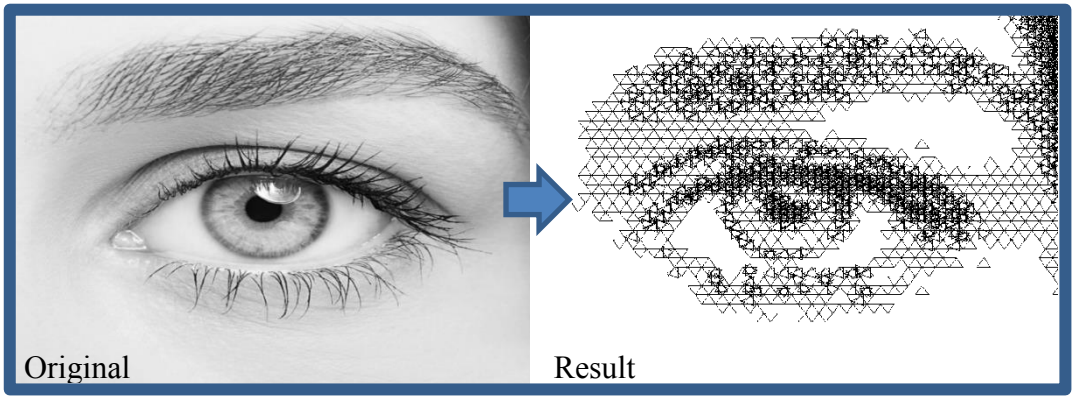

Figure 24

The user chooses the image in which the filter is to be applied. The algorithm receives from the user input parameters. Otherwise, the algorithm may use the default values to initiate the calculation process. The user can modify as desired the number of points used for the Delaunay Triangulation, the contrast and the brightness values.

The algorithm process begins with the image opening, which is transformed into the grey-scale. The algorithm applies the Gaussian Blur filter in order to reduce the image noise, and, in case the user had introduced contrast and brightness values different to the default values then the image will be adapted in accordance to those changes introduced by the user.

The ***generate_equilateral_points(npoints, size)*** function generates the points that will triangulate to equilateral triangles. To get equilateral triangles the grid has to be offset so that every other line is advanced by half the x-spacing. For example, a grid like:

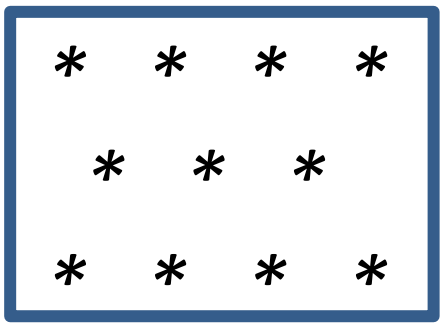

Figure 25

The ***npoints*** is the number of points to generate and size is a 2-tuple describing the maximum x and y values of the field and returns a list of point objects.

Once these points are available ***the Delaunay_Triangulation (points)*** starts; it employs the Bowyer-Watson algorithm explained in the previous chapter and pre-calculates the circumcircles of the triangles. The ***Delaunay_Triangulation*** function produces a list of triangles. These are used by the function:

color_from_image(background_image, triangles)

This function gives a color to each triangle, which in time is defined by the color of the image pixel at its centroid. Since we are dealing with gray-scale images, with "colors" we are referring to pixel's intensity level; a number between 0 and 255.

colors = []

pixels = background_image.load()

size = background_image.size

FOR EACH ***t*** IN ***triangles:***

centroid = tri_centroid(t)

Truncate the coordinates to fit within the boundaries of the image

int_centroid = (int(min(max(centroid[0], 0), size[0] - 1)),
int(min(max(centroid[1], 0), size[1] - 1)))

Get the color of the image at the centroid

p = pixels[int_centroid[0], int_centroid[1]]

colors.append(p)

ENDFOR

RETURN ***colors***

This method returns a list of color objects. This list along with the list of polygons defined by their vertices as the x, y coordinates are used by the function:

draw_polys(colors, polys, background_image)

to create the output file ***.svg*** which in time defines the ultimate image effect. This file is created by evaluating the color assigned to each triangle as shown in the table below:

 # From 0-51 -----> 0 BLACK 4 tris
 # 52-101 ----> 1 3 tris
 # 102-152 --> 2 2 tris
 # 153-203 --> 3 1 tri
 # 204-255 --> 4 WHITE No draw, no triangles

Color values ranging from 0 to 51 are equivalent to the creation of 4 inscribed triangles in the same circumference rotated at 30 degrees from each other using the same rotation center. O value is the blackest value.

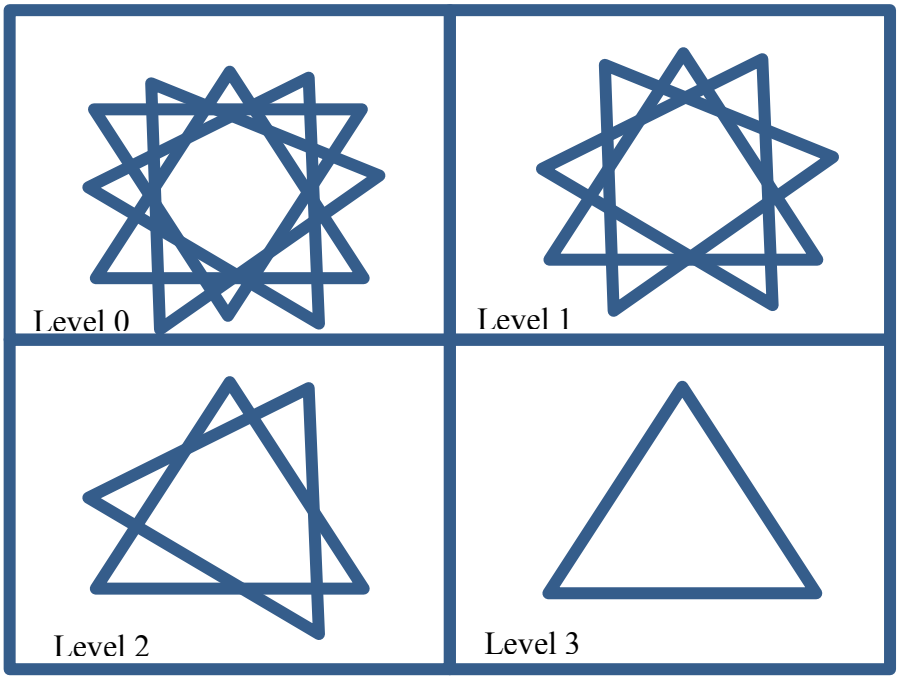

Figure 26

Color values from 52 to 101 are equivalent to the creation of 3 inscribed triangles.

Color values from 102 to 152 are equivalent to the creation of 2 inscribed triangles.

Color values from 153 to 203 are equivalent to the creation of 1 inscribed triangle.

Color values from 204 to 255 are the whiter values, this means there will be no triangle. Color value 255 is pure white.

3.2 Circumferences Filter Algorithm

This filter uses circumferences as graphic patterns to create an innovative effect on the resulting images.

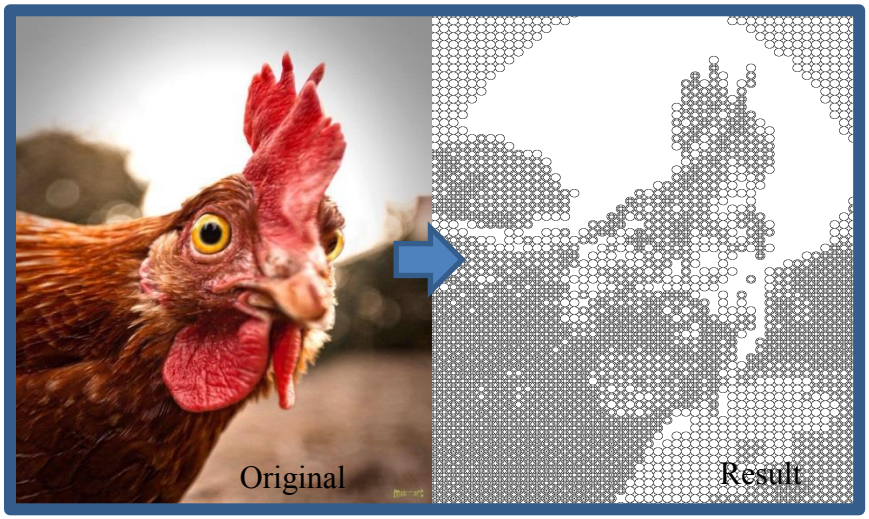

Figure 27

The algorithm includes a part of pre-processing that after the selection of the input image by the user, proceeds to transform the image in grayscale to then filter it by using a “Gaussian Blur” filter, which has the task of eliminating noise. Next, image enhancement tools are applied; contrast and brightness values are used as far as they are different from those of default to improve the quality of the input image and consequently the resulting effect.

From now on the image is treated as a matrix of pixels (raster image), which allows access to features of each pixel through the use of indexes.

Two nested loops allow to run across the entire image for the analysis and creation of the desired effect. The external loop runs across the height of the image, while the internal cycle runs along the width, thus allowing the entire image to be studied.

WHILE **index** IS LESS THAN OR EQUALS TO **Imageheight – radius {**

WHILE **index2** IS LESS THAN OR EQUALS TO **Imagewidth {**

}

}

Within the internal cycle, that is, for each horizontal line and with the step defined by the user at the beginning of the processing, several actions are carried out:

1. A function called ***"Average pixel"*** finds the average pixel color value of the "zone of interest". This zone is a squared sub-matrix in which the size depends of the step defined by the user. For example, if the user decides to use 20 as the step value, the denominated zone will be 20x20. This function returns the average pixel color value in that specific area.

FUNCTION ***avgPixel(width, height, starti, startj, imgPixelMap):***

summ = 0

FOR EACH ***k*** IN ***range(0, height):***

FOR EACH ***l*** IN ***range(0, width):***

summ += imgPixelMap[startj, starti]

avgPixelValue = summ / (width * height)

RETURN ***int(avgPixelValue)***

2. The "Normalize" function uses the value produced by the previous function and places it in one of the 5 levels (from [0-4]).

FUNCTION ***normalize(pixelValue):***

IF ***(pixelValue*** IS GREATER THAN OR EQUALS TO ***0)*** AND ***(pixelValue*** IS LESS THAN OR EQUALS TO ***51)*** THEN***:*** RETURN ***0***

IF ***(pixelValue*** IS GREATER THAN OR EQUALS TO ***52)*** AND ***(pixelValue*** IS LESS THAN OR EQUALS TO ***101)*** THEN***:*** RETURN ***1***
ENDIF
IF ***(pixelValue*** IS GREATER THAN OR EQUALS TO ***102)*** AND ***(pixelValue*** IS LESS THAN OR EQUALS TO ***152)*** THEN***:*** RETURN ***2***
ENDIF
IF ***(pixelValue*** IS GREATER THAN OR EQUALS TO ***153)*** AND ***(pixelValue*** IS LESS THAN OR EQUALS TO ***203)*** THEN***:*** RETURN ***3***
ENDIF
IF ***(pixelValue*** IS GREATER THAN OR EQUALS TO ***204)*** AND ***(pixelValue*** IS LESS THAN OR EQUALS TO ***255)*** THEN***:*** RETURN ***4***
ENDIF

If a color value is between 0 and 51 (included) then it is equal to 0, between 52 and 101 it is equal to 1, and so on for values between 102 and 152. In this case the third level (2) will be assigned. Values between 153 and 203 belong to fourth level (3); while the remaining ones will be assigned the last level.

3. Based on the normalized value, the creation of the circumferences that will define our output image begins. A guard is responsible for separating two large cases.

IF ***avg*** IS LESS THAN ***4*** THEN***:***

If the normalized value is less than 4, that is, it does not belong to the last level of normalization, it means that at least one circle will be drawn and the function ***"Calculate Circumference"*** will be called. Otherwise, when the value belongs to the fifth level no circumference will be drawn and the area of interest will be white; this, in turn means that in the input image was an area that mostly contained pure white values (255) or very close to it (204-255).

If the ***"Calculate Circumference"*** method is invoked, the calculation of the circumferences is executed. Using the starting points of the area of interest the center of the possible circumferences is calculated. These together with the step and the normalized color value serve to start the calculation of the figures. From here three cases are recognized:

- If the normalized value is equal to 0, this means that it belongs to the level with more intensity that in turn translates into the definition of 4 concentric circles inscribed in the sub-matrix / area of interest mentioned above. The largest circumference will have the diameter corresponding to the step, while the others will have smaller diameters dependent on each other and decreasing by 3 pixels each.
- The second case (level 1) contains only 3 circumferences.
- The third case (level 2) will draw 2 and the fourth case corresponding to level 3 will have only 1 inscribed circumference.

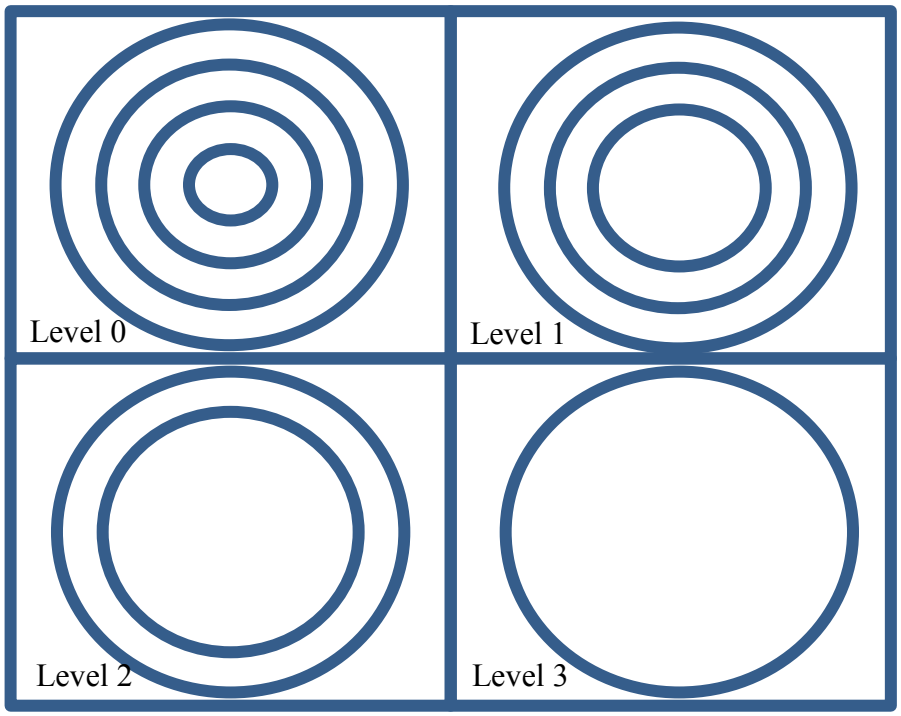

Figure 28

Once each circumference is obtained it is added to a variable which is responsible for storing the SVG path that will later serve the ***"saveSvg"*** function.

polyString += '<circle cx="%d" cy="%d" r="%d" />/n' % (cx, cy, r)

4. ***"saveSvg"*** Put together all the pieces needed to create an svg file. From the header, to the instructions and paths to the appropriate closing of the file. Thus producing the output svg file which will then be interpreted by Scribit and subsequently drawn.

FUNCTION ***saveSVG(pathstring, size):***

textSVG = '<svg xmlns=\"http://www.w3.org/2000/svg\" viewBox=\"0 0 {0} {1} \">\n<g fill="none" stroke="#000000">\n{2}\n</g></svg>'.format(

size[0], size[1], pathstring)

f = open("./circles_results.svg", "w+")

f.write(textSVG)

f.close()

3.3 Lines effect filter Algorithm

This algorithm called lines effect uses lines and their different angles to create innovative graphics outcomes. Two connected points form a line. Depending on the distance that separates them, the lines will tend to grow or decrease their length. The areas of the image that have greater intensity, that is to say darker colors, will be more intense thanks to the superposition of several lines and their different angle. In this way, a contrast will be created between the different intensities and colors of the image that will allow a greater definition.

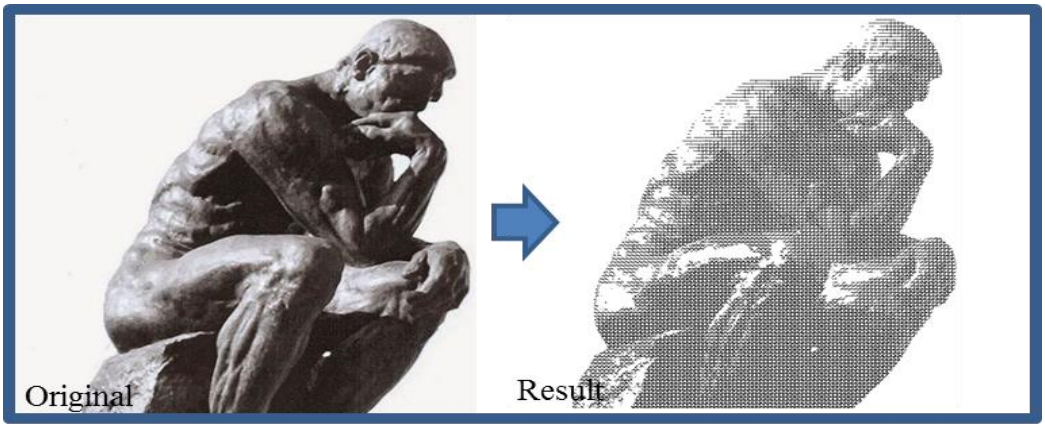

Figure 29

Unlike the other filters developed and explained above, this one, when functions properly require square input images (Example: 512 * 512). The algorithm is based on a logic that makes use square matrices that have a main and a secondary diagonal. The main and secondary diagonal divides our image (seen as a matrix) into two triangular matrices; an upper and a lower matrix. Both are analyzed and studied to create the result effect on the output image.

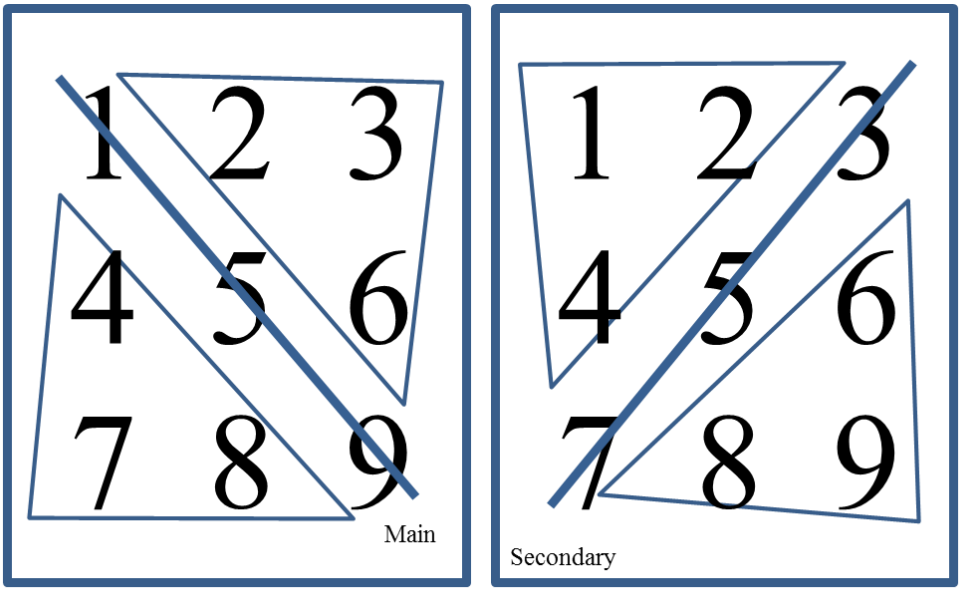

Figure 30

Once the algorithm is executed specifying the input image its dimensions are calculated and then the preprocessing is applied; Gray-scale image transformation, contrast and brightness enhancement and reduction of image noise.

Successively the analysis and calculation of the characteristics of the image seen as a matrix of pixels begins. The user decides the step for this algorithm that means the number of pixels of separation between lines. For example: a step of 6 is equivalent to a narrow separation of 6 pixels that generates an image with an acceptable definition, a step of 2 equals a longer execution time but corresponds to a greater definition, while a step of 10 it requires less execution time but in turn decreases the quality of the final result and so on.

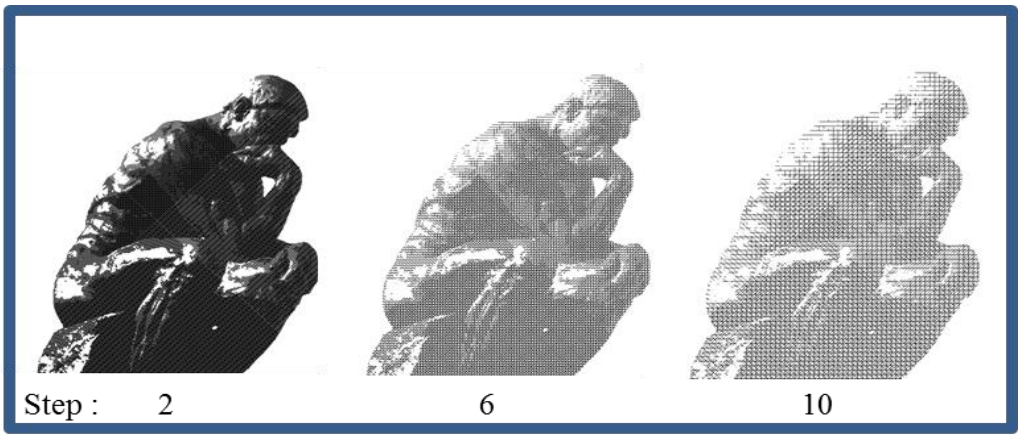

Figure 31

Using the same principle, as the other algorithms developed in this thesis, 5 intensity levels are defined. Being the first level (0) corresponding to the highest intensities, black values and / or close to pure black, while the fifth and last level correspond to the clearest values. The areas belonging to the levels with higher intensities will have a greater number of intersections of lines: horizontal, vertical and diagonals, meanwhile those areas belonging to the highest levels will have few intersections or will lack them.

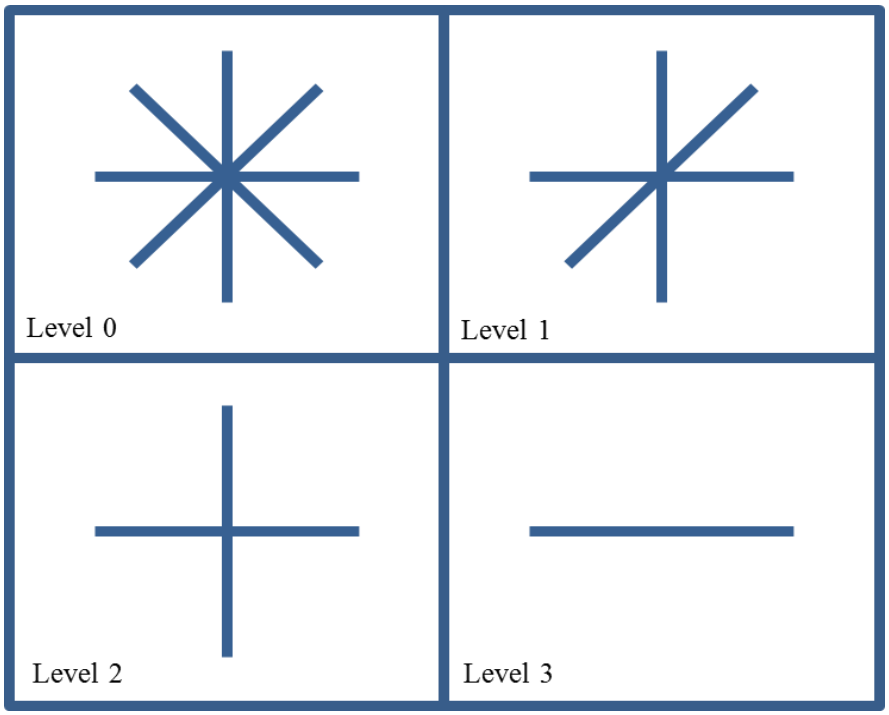

Figure 32

Four important functions have the task of running across the pixel matrix in different directions for analysis and processing. Each of them uses a function called **"checkPixel"** as a support, which is responsible for returning a Boolean value (true or false) according to their membership or not at the required intensity level (passed as an argument to the function). Horizontal lines will be drawn and assigned only to pixel intensity values less than 204. Vertical lines to values lower than 153 and the two types of diagonals less than 102 and 52 correspondingly.

```
FUNCTION checkPixel(starti, startj, imgPixelMap, level):
  IF imgPixelMap[startj, starti] IS LESS THAN level THEN:
    RETURN True
  ELSE THEN:
    RETURN False
```

The lines that will define the final image are made up of two points, an initial and a final point. The approach these functions use to calculate the lines is incremental. The matrix is run across a specific sense (horizontal, vertical or diagonal). First an initial point is sought to define the beginning of our first line, then the image scanning follows up to a point of not-meeting-the-level requirements is found. This indicates the point hereinbefore this one is the end of the line. The process continues when a new point is found that satisfies the necessary conditions for the start of a new line and so on.

These functions use two loops to run the entire image using a ***"newLine"*** boolean variable as support, which has the task of signaling when a new line is started.

newLine = True

Thanks to this variable the code is able to understand when to save the line just found in the variable in charge of storing the svg instructions.

polyString += '<line x1="%d" y1="%d" x2="%d" y2="%d" />/n' % (pt1.x, pt1.y, pt2.x, pt2.y)

- Horizontal: Based on the approach explained above, it runs the matrix horizontally finding in turn the pixels that will be part of the defined horizontal lines. For a point to belong to a horizontal line it must have an intensity value below 204.

FUNCTION ***horizontal(width, height, increment, level):***

newLine = True

check = True

pt1 = Point(0, 0)

pt2 = Point(0, 0)

i = j = 1

```
WHILE i IS LESS THAN OR EQUALS TO height-1:
  WHILE j IS LESS THAN OR EQUALS TO width-1:
    check = checkPixel(i, j, imgPixelMap, level)
    IF check THEN:
      IF newline THEN:
        pt1 = pt2 = Point(j, i)
        newLine = False
      ENDIF
     ELSE THEN:
        pt2 = Point(j, i)
      ENDELSE
    ENDIF
    ELSE THEN:
      polyString += '<line x1="%d" y1="%d" x2="%d" y2="%d" />/n' %
      (pt1.x, pt1.y, pt2.x, pt2.y)
      newLine = True
      pt1 = Point(0, 0)
      pt2 = Point(0, 0)
    ENDELSE
   j = j + 1
 i = i + increment
 j = 0
```

- Vertical: Is similar to the horizontal one, but it transposes the rows and columns. When it calls the function **"checkPixel"** uses a different value because only those pixels with values less than 153 will be part of one of the vertical lines.

- Diagonal Up: This function divides the work into two parts; the upper and the lower part to the diagonal.

To calculate the upper part starts from the upper left corner of the image and continues in the direction of the arrow as indicated in *Figure 32*, always keeping the preset step between diagonal lines.

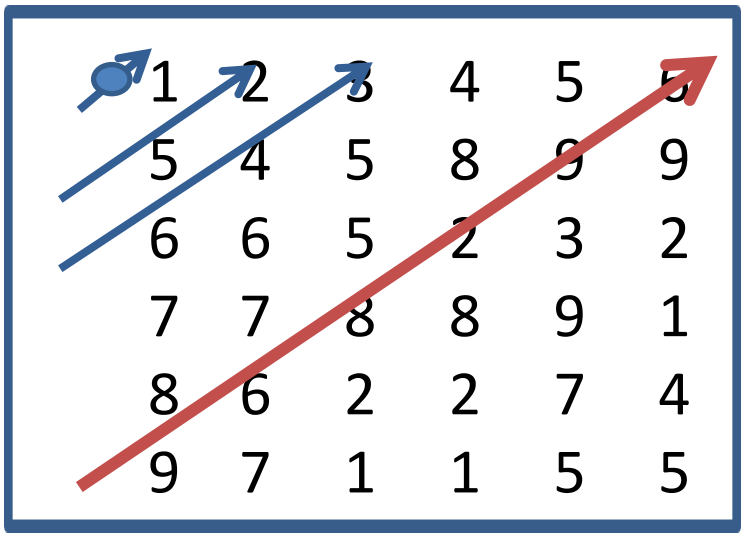

Figure 33

```
WHILE i IS LESS THAN OR EQUALS TO height - 1:
    WHILE j IS LESS THAN OR EQUALS TO width - 1 AND i GREATER THAN OR
EQUALS TO 0:
        check = checkPixel(i, j, imgPixelMap, level)
        IF check THEN:
            IF newline THEN:
                pt1 = pt2 = Point(j, i)
                newLine = False
            ENDIF
            ELSE THEN:
                pt2 = Point(j, i)
                IF i == 0 THEN:
```

```
                polyString += '<line x1="%d" y1="%d" x2="%d" y2="%d" />/n' %
                (pt1.x, pt1.y, pt2.x, pt2.y)
                newLine = True
                pt1 = Point(0, 0)
                pt2 = Point(0, 0)
            ENDIF
        ENDELSE
    ENDIF
ELSE THEN:
    polyString += '<line x1="%d" y1="%d" x2="%d" y2="%d" />/n' %
    (pt1.x, pt1.y, pt2.x, pt2.y)
    newLine = True
    pt1 = Point(0, 0)
    pt2 = Point(0, 0)
ENDELSE
j = j + 1
i = i - 1
i = j + increment
j = 0
```

To calculate the lower part it starts from the lower right corner of the image and continues in the direction of the arrow as indicated in the *Figure 33*, always keeping the preset step between diagonal lines.

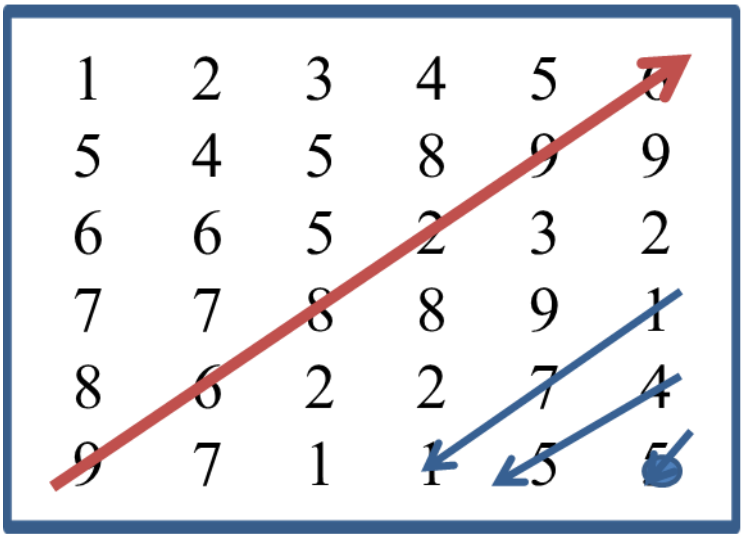

Figure 34

- Diagonal Down: As the diagonal up function it divides the work into two parts; the upper and the lower part.

To calculate the upper part starts from the upper right corner of the image and continues in the direction of the arrow as indicated in *Figure 34*, always keeping the preset step between diagonal lines.

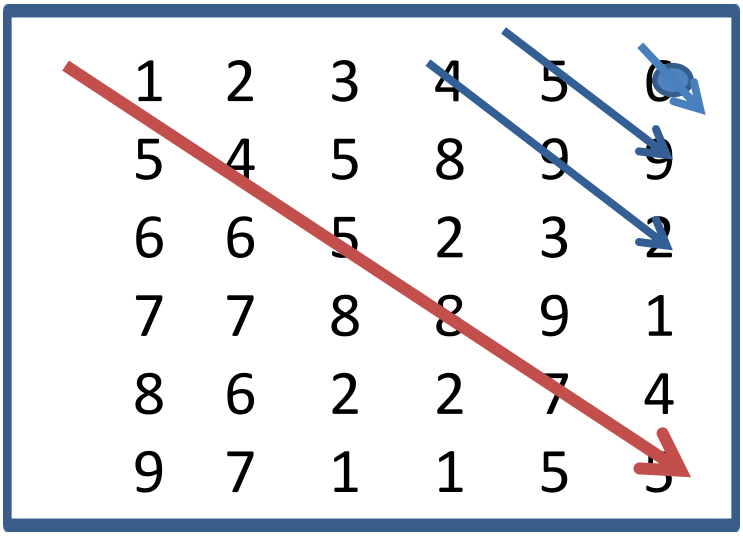

Figure 35

```
WHILE i IS LESS THAN OR EQUALS TO height - 1 AND j IS GREATER THAN
OR EQUALS TO 0:
    WHILE j IS LESS THAN OR EQUALS TO  width - 1:
      check = checkPixel(i, j, imgPixelMap, level)
      IF check THEN:
        IF newLine:
          pt1 = pt2 = Point(j, i)
          newLine = False
        ENDIF
        ELSE THEN:
          pt2 = Point(j, i)
          IF i EQUALS TO 0 THEN:
            polyString += '<line x1="%d" y1="%d" x2="%d" y2="%d"
            />/n' % (pt1.x, pt1.y, pt2.x, pt2.y)
            newLine = True
            pt1 = Point(0, 0)
            pt2 = Point(0, 0)
          ENDIF
        ENDELSE
      ENDIF
      ELSE THEN:
        polyString += '<line x1="%d" y1="%d" x2="%d" y2="%d" />/n' %
        (pt1.x, pt1.y, pt2.x, pt2.y)
        newLine = True
        pt1 = Point(0, 0)
        pt2 = Point(0, 0)
      ENDELSE
    j = j + 1
    i = i + 1
```

j = j - (i + 1) - increment

i = 0

i = height - 2

j = 0

newLine = True

To calculate the lower part it starts from the lower left corner of the image and continues in the direction of the arrow as indicated in the *Figure 35*, always keeping the preset step between diagonal lines.

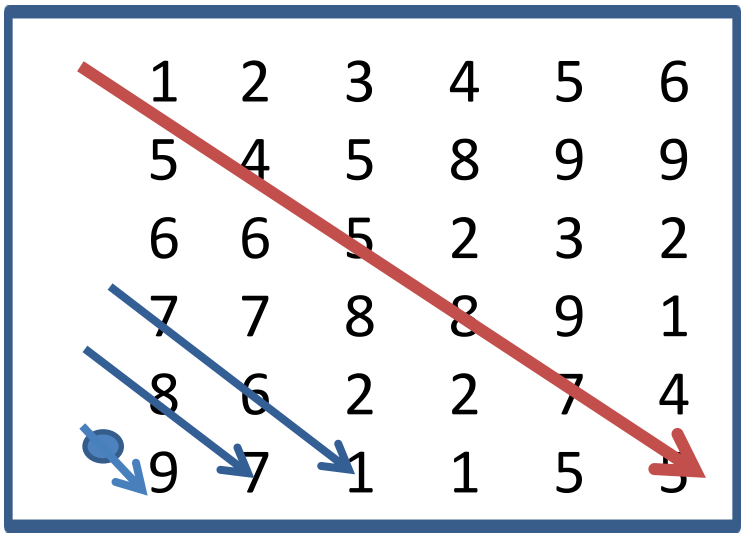

Figure 36

Finally this algorithm is responsible for using ***saveSVG*** to bring together all the pieces needed to create the file. All the instructions are written into an ***.svg*** file which will later be used by the robot.

FUNCTION ***saveSVG(pathstring, size):***

```
    textSVG = '<svg xmlns=\"http://www.w3.org/2000/svg\" viewBox=\"0   0 {0} {1}
\">\n<g fill="none" stroke="#000000">\n{2}\n</g></svg>'.format(
        size[0], size[1], pathstring)
    f = open("./circles_results.svg", "w+")
    f.write(textSVG)
    f.close()
```

3.3 Wiggle draw filter algorithm

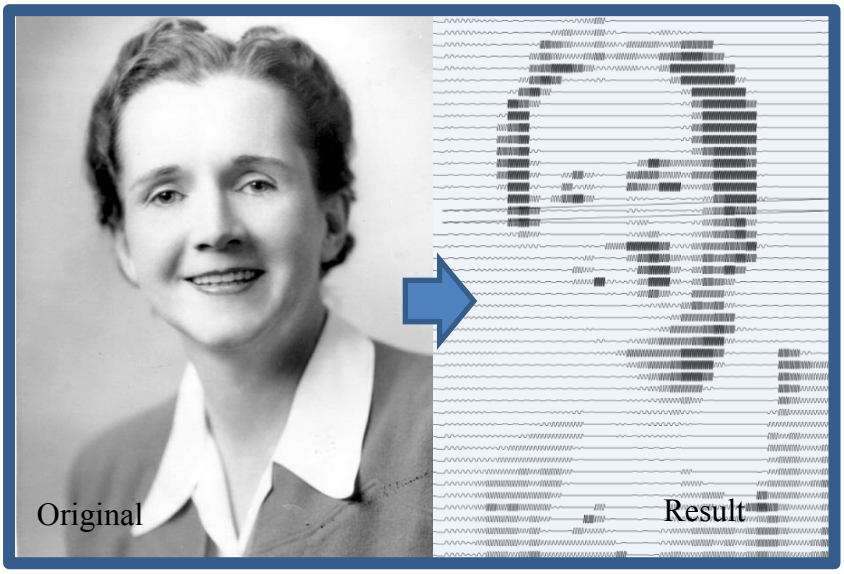

Figure 37

This filter makes use of the Bezier curves as the main design element.

Once the input image is received, two loops are in charge of running across the matrix for analysis. The outer loop iterates horizontally while the inner loop iterates vertically (columns).

The ***leftToRight*** variable is a boolean variable that allows the optimization of the SVG code and therefore decreases the final design time. It is in charge of telling us when the end of the line has been reached. In this way, the successive line will be run in the opposite way. In other words, the first analysis of the first part of the matrix will be done from left to right (***leftToRight*** = true), once the last pixel of the line in question is reached, the ***leftToRight*** flag is assigned to false, which tells us that the next assessment will be done from right to left.

This results in the way Scribit designs, which does not always follow the regular writing parameter, that is, from left to right, but alternates directions, thus improving the completion time of the task.

The ***pixelSum*** variable stores the values of the pixels intensities being processed at the moment.

pixelSum += imgPixelMap[j, i]

Every 30 pixels (value that can be modified) the calculation of the average of the intensities is carried out and a function called ***normalized*** is invoked, which is in charge of the normalization (take values from a scale of 0-255 to a scale of 9 values).

FUNCTION ***normalize(pixelValue):***

IF ***(pixelValue*** IS GREATER THAN OR EQUALS TO ***0)*** AND ***(pixelValue*** IS LESS THAN OF EQUALS TO ***25):***

RETURN ***0***

IF ***(pixelValue*** IS GREATER THAN OR EQUALS TO ***26)*** AND ***(pixelValue*** IS LESS THAN OF EQUALS TO ***51):***

RETURN ***26***

IF ***(pixelValue*** IS GREATER THAN OR EQUALS TO 52***)*** AND ***(pixelValue*** IS LESS THAN OF EQUALS TO ***76):***

RETURN ***52***

IF ***(pixelValue*** IS GREATER THAN OR EQUALS TO 77***)*** AND ***(pixelValue*** IS LESS THAN OF EQUALS TO ***101):***

RETURN ***77***

IF ***(pixelValue*** IS GREATER THAN OR EQUALS TO ***102)*** AND ***(pixelValue*** IS LESS THAN OF EQUALS TO ***127):***

RETURN ***102***

IF ***(pixelValue*** IS GREATER THAN OR EQUALS TO ***128)*** AND ***(pixelValue*** IS LESS THAN OF EQUALS TO ***152):***

RETURN ***128***

IF ***(pixelValue*** IS GREATER THAN OR EQUALS TO ***153)*** AND ***(pixelValue*** IS LESS THAN OF EQUALS TO 179***):***

RETURN ***153***

IF ***(pixelValue*** IS GREATER THAN OR EQUALS TO ***180)*** AND ***(pixelValue*** IS LESS THAN OF EQUALS TO ***203):***

RETURN ***180***

IF ***(pixelValue*** IS GREATER THAN OR EQUALS TO ***204)*** AND ***(pixelValue*** IS LESS THAN OF EQUALS TO ***230):***

RETURN ***204***

IF ***(pixelValue*** IS GREATER THAN OR EQUALS TO ***231)*** AND ***(pixelValue*** IS LESS THAN OF EQUALS TO ***255):***

RETURN ***255***

Unlike the other algorithms presented above, it uses a 10-level pixel intensity level normalization scale. These correspond to ten different types of curves (wiggles), each of them characterized by a particular wavelength and amplitude.

Swich_demo is in charge of storing these correspondences; for each level it selects the respective values that characterize and define the appropriate Bezier curve (wavelength and wave height).

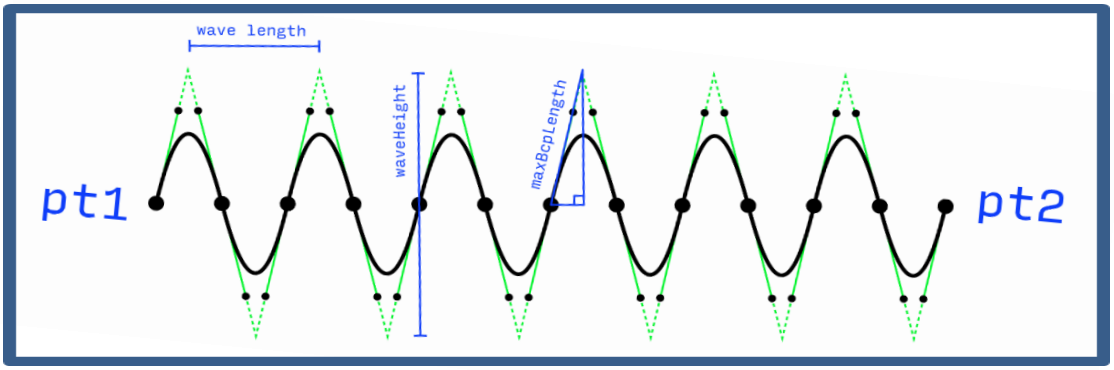

Figure 38

The variable ***factor*** is used to have direct control of the wavelength and wave height values. The areas of the image with greater intensity, that is, colors closer to black, are characterized by having a smaller wavelength (more density of curves, more wiggles in a distance between two points) and a larger wave height (higher wiggles/curves).

```
FUNCTION switch_demo(avg):
  IF avg EQUALS TO 0 THEN:
    RETURN 3, 100*factor
  ENDIF
  ELSEIF avg EQUALS TO 26 THEN:
      RETURN 4, 90*factor
  ENDELSEIF
  ELSEIF avg EQUALS TO 52 THEN:
      RETURN 5, 80*factor
  ENDELSEIF
  ELSEIF avg EQUALS TO 77 THEN:
      RETURN 7, 68*factor
  ENDELSEIF
  ELSEIF avg EQUALS TO 102 THEN:
```

```
        RETURN 10, 53*factor
    ENDELSEIF
    ELSEIF avg EQUALS TO 128 THEN:
        RETURN 13, 35*factor
    ENDELSEIF
      ELSEIF avg EQUALS TO 153 THEN:
        RETURN 15, 20*factor
    ENDELSEIF

    ELSEIF avg EQUALS TO 180 THEN:
        RETURN 20, 13*factor
    ENDELSEIF
      ELSEIF avg EQUALS TO 204 THEN:
        RETURN 25, 5*factor
    ENDELSEIF
    ELSEIF avg EQUALS TO 255 THEN:
        RETURN 30, 0*factor
    ENDELSEIF
   ELSE THEN:
     RETURN -1, -1
    ENDELSE
```

Before starting the computation that involves the calculation of the control points of the curves, it is verified whether it corresponds to a containing area mostly white. If true, you would be dealing with a Bezier curve whose height is zero that clearly corresponds to a straight line. For this type of case, it was decided to use the svg command for lines ***"L% d% d"*** since they are less heavy and easier to draw than a Bezier curve without height.

If it belongs to one of the 9 remaining cases it means that the main function ***calcWiggle(pt1, pt2, wavelenght, waveheight)*** is invoked. This has the important task of finding the elements that describe the Bezier curves that will be subsequently designed. This function takes a start point, an end point and the values of wavelenght and wave height as parameters. First of all, two possible errors have to be detected. The ***curveSquaring*** must be a value between 0 and 1, while ***wavelenght*** must be bigger than zero.

```
assert 0 <= curveSquaring <= 1
assert waveLength > 0
```

The next step is to calculate the distance between the two points; using

$$\sqrt{(\boldsymbol{pt1}.\boldsymbol{x} + \boldsymbol{pt2}.\boldsymbol{x})^2 + (\boldsymbol{pt1}.\boldsymbol{y} + \boldsymbol{pt2}.\boldsymbol{y})^2}$$

```
diagonal = calcDistance(pt1, pt2)
FUNTION calcDistance(pt1, pt2):
    RETURN sqrt((pt1.x - pt2.x)**2 + (pt1.y - pt2.y)**2)
```

and successively the angle of inclination in radians of the wiggle; using

$$\mathbf{tan^{-1}}(\mathbf{pt2}.\mathbf{y} - \mathbf{pt1}.\mathbf{y}, \mathbf{pt2}.\mathbf{x} - \mathbf{pt1}.\mathbf{x})$$

```
angleRad = calcAngle(pt1, pt2)
def calcAngle(pt1, pt2):
  return atan2((pt2.y - pt1.y), (pt2.x - pt1.x))
```

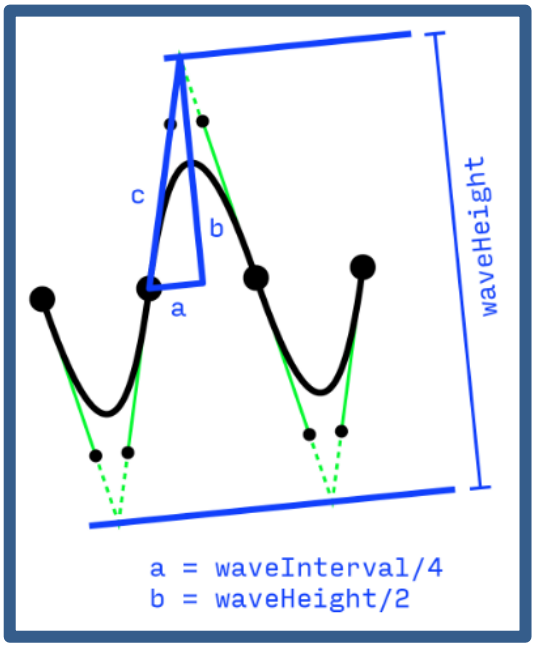

Figure 39

The number of wiggles between the two points is calculated using the above calculated values and then the interval.

wavesNumber = diagonal//int(waveLength)

waveInterval = diagonal/float(wavesNumber)

Right after a series of calculations is carried out to obtain the values of the control points. The segment that ideally connects ***pt1*** and ***pt2*** is divided by a point called the inflection point (when a Bezier curve changes direction).

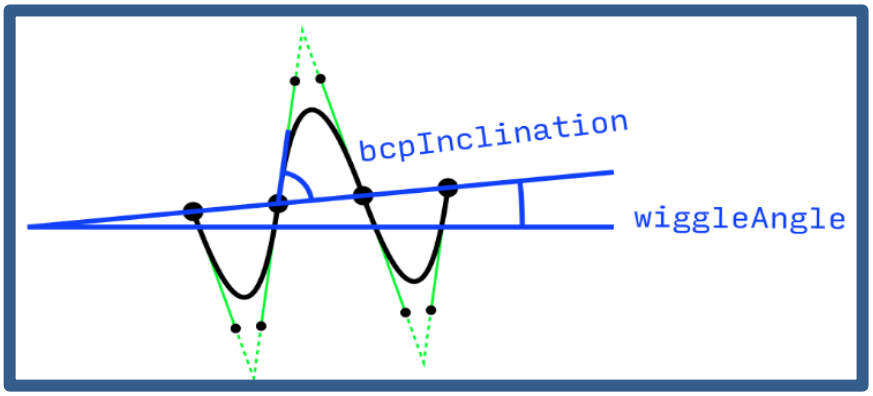

Figure 40

In order to find out the position of the Bezier control points projecting from the flex point, we need to determine the inclination angle (the same formula used above; arc tangent is used) and the Bezier control point length.

maxBcpLength = sqrt((waveInterval/4.)**2+(waveHeight/2.)**2)

bcpLength = maxBcpLength*curveSquaring

bcpInclination = calcAngle(Point(0,0), Point(waveInterval/4., waveHeight/2.))

The **for** loop iteratively computes the wiggle points and stores them in a list. Each iteration calculates the triplet of points needed to define a Bézier curve; ***bcpOut*** (which is linked to the previous points), **bcpIn**, and the end flex points.

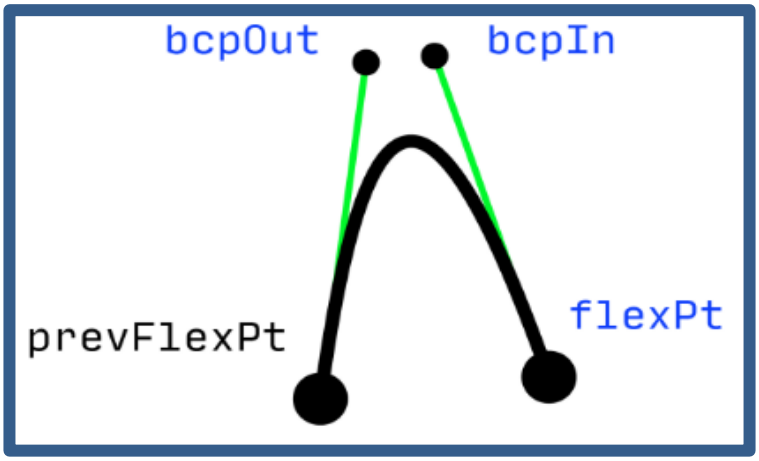

Figure 41

bcpOutAngle = angleRad+bcpInclination*polarity

bcpOut = Point(prevFlexPt.x+cos(bcpOutAngle)*bcpLength,

prevFlexPt.y+sin(bcpOutAngle)*bcpLength)

flexPt = Point(prevFlexPt.x+cos(angleRad)*waveInterval/2.,

prevFlexPt.y+sin(angleRad)*waveInterval/2.)

bcpInAngle = angleRad+(radians(180)-bcpInclination)*polarity

bcpIn = Point(flexPt.x+cos(bcpInAngle)*bcpLength,

flexPt.y+sin(bcpInAngle)*bcpLength)

When the loop finishes its execution it returns the list. This is used so that later the c***heckNewLine*** function writes the svg commands that structure

the Bezier curves just calculated, each time it is invoked it adds the new curves to the previous ***pathstring***.

<path d="M25,220 C75,70 170,25 225,210" ></path>

pathstring = pathstring + "M%d,%d" % (wigglePoints[0].x, wigglePoints[0].y)

ELSE THEN*:*

FOR EACH ***eachBcpOut, eachBcpIn, eachAnchor*** IN ***wigglePoints[1:]:***

pathstring = pathstring +" C%d,%d %d,%d %d,%d" % (eachBcpOut.x, eachBcpOut.y, eachBcpIn.x, eachBcpIn.y, eachAnchor.x, eachAnchor.y)

The last step, as in the other algorithms in this thesis is to use the ***saveSVG*** function to save the data in the file with the ***.svg*** extension.

CHAPTER IV: TESTS, RESULTS AND FURTHER OPTIMIZATION

Tests and results

1. Triangles Filter Algorithm:

The results obtained with this type of filter are very good, the rotated triangles manage to define and outline the details quite accurately. The characteristics of this filter make it suitable for photos or image plenty of peculiarity and elements.

Two examples are shown in Figure 37. The first image displays a chicken standing outdoors; the depth of field can be clearly appreciated. Also a wide range of colors helps distinguish the subject from the background. The second image describes a man in great detail; wrinkles, hair and skin marks are very notable.

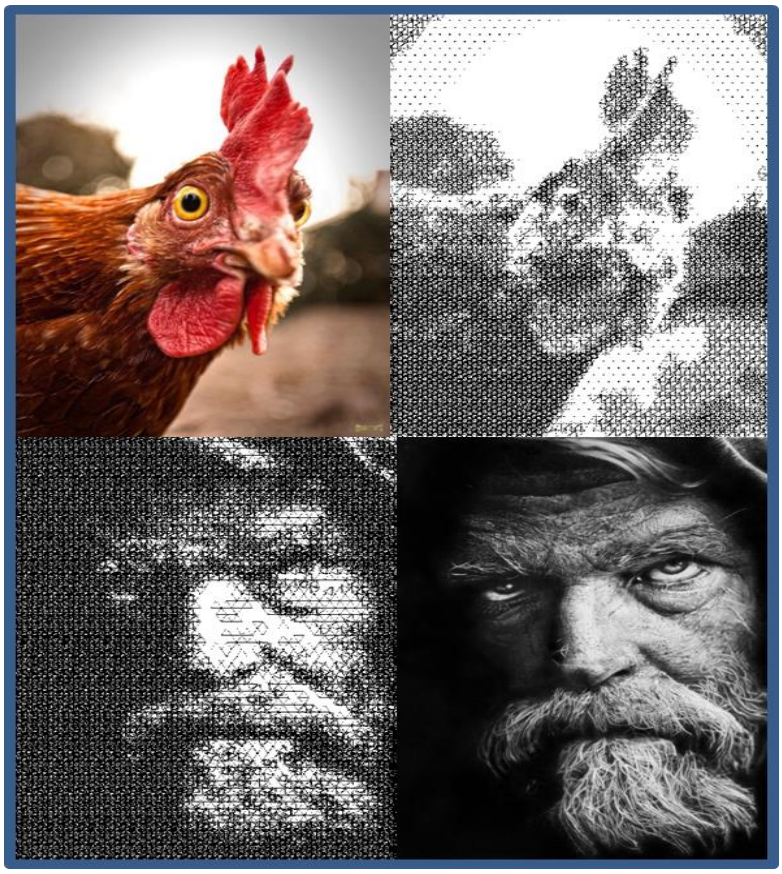

Figure 42

In both cases we can see the resulting effect maintains a quantity of details and definitions that can still be achieved with the use of a banal three-sided geometric figure, giving the original image a different and original touch.

In terms of complexity it is one of the filters that requires more analysis, calculation and processing. On the one hand the use of Delaunay triangulation significantly increases the complexity and design time of the final results, but on the other, it achieves surprising results.

Estimated average design time*: +/- 5 hours.

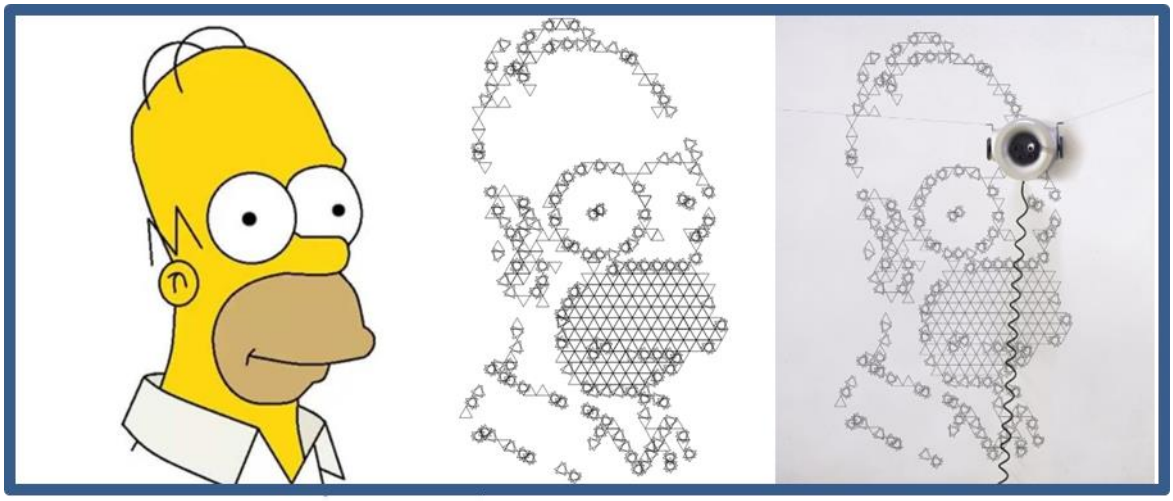

Figure 43

Scribit interprets the triangles with three connected lines, making it possible to design the triangles defined in the output SVG file quite accurately.

*Estimate for images with good count, average amount of detail, approximate resolution of 1024 * 1024 pixels, on a design surface / canvas of 2x2 meters.

2. Circumferences Filter Algorithm:

Figure 39 shows two very different images that demonstrate the capacity of this filter. The representation of angles and straight lines through the use of circumferences challenges human vision, demonstrating that the density of more or less concentric circles can generate a creative, artistic and inventive visual effect.

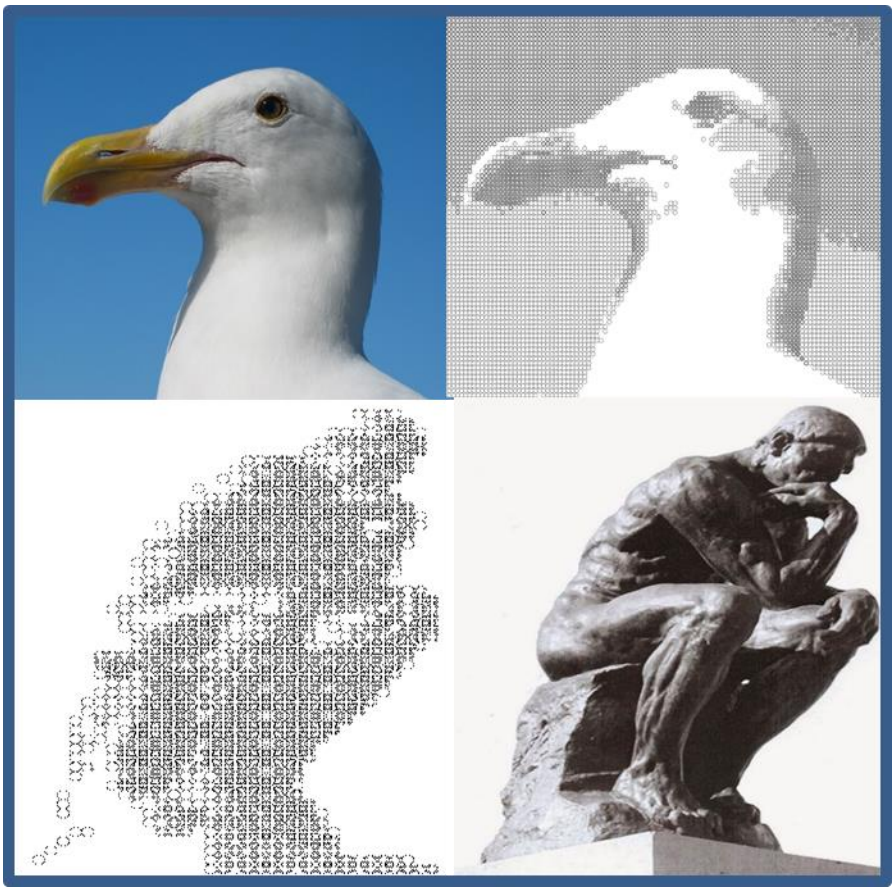

Figure 44

The complexity lies in finding the midpoint between the size of the circle that achieves a definition and understanding of the image obtained as a result and the size of the circumferences that Scribit is able to design.

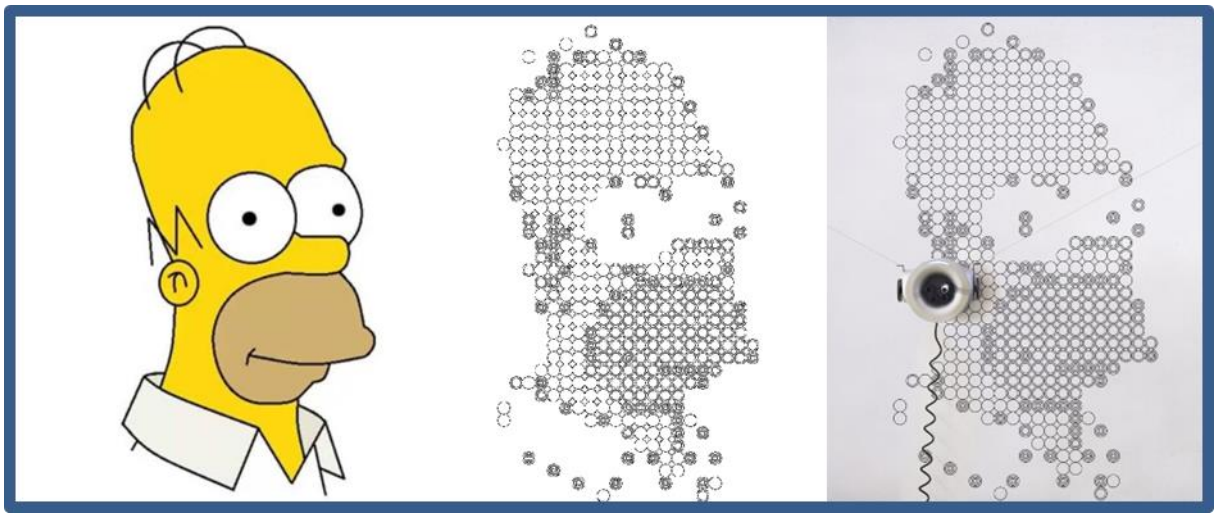

Figure 45

After many calculations and design tests, it was concluded that by the type of markers used by the Scribit robot, the smaller circumference that the robot is able to design has a diameter of 5 pixels. A circumference of this size is physically translated into a point, that is, a circle so small that it resembles a point.

Estimated average design time* : +/- 4 1/2 hours.

3. Lines Filter Algorithm:

Consecutive fine points provide and enrich the outlines of the elements that make up the images. The results obtained with the application of this filter are quite realistic and true to the original image. The straight lines can be defined with great precision and the shades closest to pure black are defined by a cross or interception of

*Estimate for images with good count, average amount of detail, approximate resolution of 1024 * 1024 pixels, on a design surface / canvas of 2x2 meters.

all the lines with which the algorithm works, which allows to recreate the intensity of the original pixels.

As *Figure 41* shows, a good compromise is achieved between the quality of the results obtained and the design time. The robot interprets the commands for line design very easily, since it is based on finding the starting point and running the marker to the end point. This makes it quite fast and effective.

Figure 46

Estimated average design time* : +/- 4 hours.

*Estimate for images with good count, average amount of detail, approximate resolution of 1024 * 1024 pixels, on a design surface / canvas of 2x2 meters.

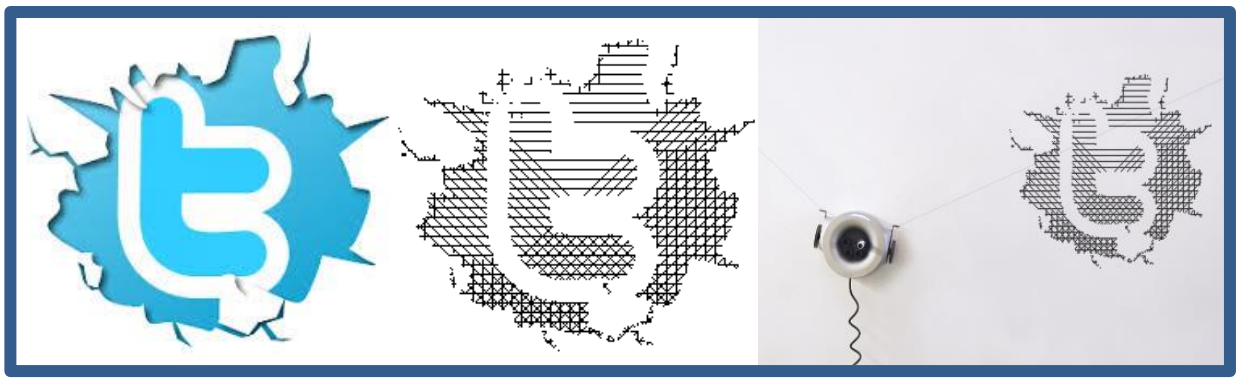

Figure 47

4. Wiggle Filter Algorithm:

This filter creates more remarkable effects that could be classified as a distortion effect, as shown in *Figure 40*. The human eye is used to detect and understand straight lines so the use of curves with different amplitudes and heights makes it one of the most original and innovative filter.

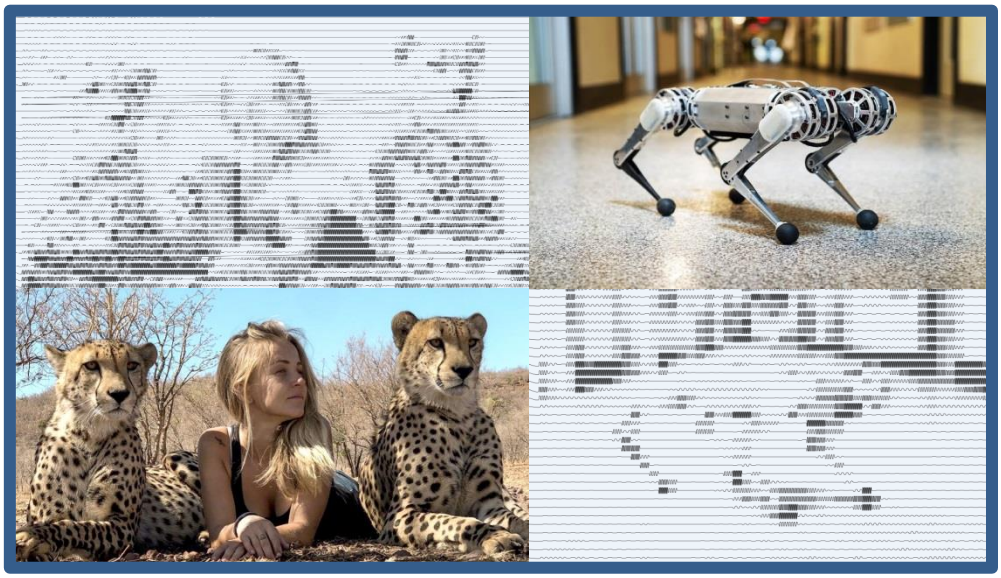

Figure 48

Thanks to the compact representation that the SVG language has of Bezier curves, it is possible to translate an element as complex as these curves into commands readable and assimilable by Scribit. Undoubtedly, these elements need more elaboration, which is transcribed into a longer designing time.

Estimated average design time* : +/- 5 hours.

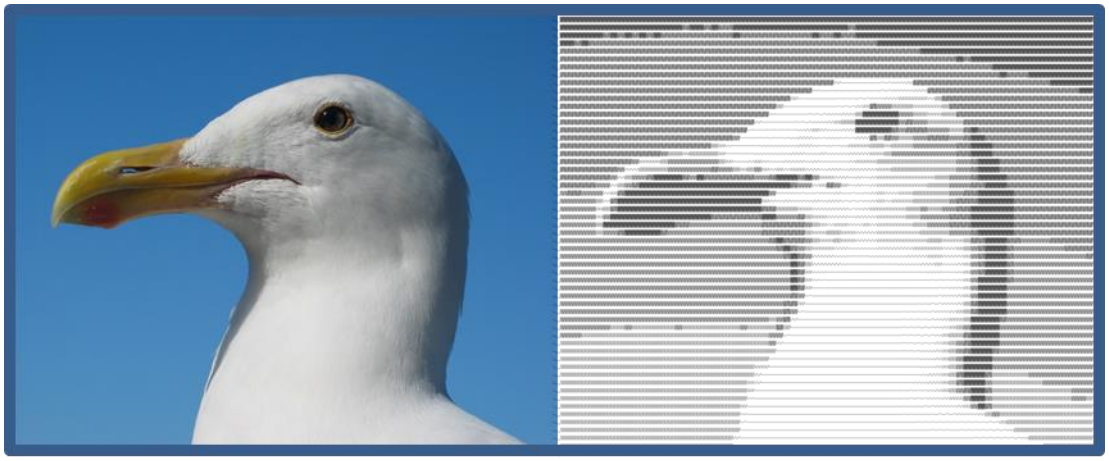

Figure 49

*Estimate for images with good count, average amount of detail, approximate resolution of 1024 * 1024 pixels, on a design surface / canvas of 2x2 meters.

Further optimizations

The field of mathematics together with computer vision opens up endless options for creative development. Starting from the bases with which all these filters were created, they could be extended and modified for the use with other types of figures or geometric elements, for example the use of squares or polygons with a greater number of sides.

1. Triangles Filter Algorithm:

This filter with its established 5 levels of normalization generates awesome results with good definition. These levels could be increased to create a greater visual effect, including modifying the angles of rotation of the equilateral triangles or thinking about the possibility of generating random triangles or different types of triangles (not only of the equilateral type).

2. Circumferences Filter Algorithm:

The use of markers with finer tips by Scribit would allow better performance and definition of the results obtained with this filter. Considering finer points implies that the circles designed in their turn may be smaller, which would undoubtedly increase the ability to better represent the details of the images.

3. Lines Filter Algorithm:

A valid optimization for this filter would be the way in which the input image is analyzed. Optimizing the path of the matrix would imply taking into account the value of the pixel and once it is categorized to know how many lines passed through that

point, avoiding running across the matrix several times would undoubtedly bring more benefits in terms of execution time and calculations.

4. Wiggle Filter Algorithm:

Inserting a random factor to the generation of the Bezier curves could give a different and striking touch to the results produced by this filter. Modifying the control points and the values of the wave length and the wave height would result in a greater amount of different curves that can be used for the representation of the vector images.

CONCLUSION

New technologies are increasingly influencing societies. The impact these generate in our day to day life is bigger every time. When a new product is acquired, there are more functionalities, attributes and characteristics that are evaluated and taken into account to decide if it is the right purchase.

Hence the products offered to the public should provide versatility, connectivity, innovation and integration with other technologies. In this way, customers have the possibility to take advantages from the purchased products and expect future development to increase the application environment and range of use of the devices. Precisely what was sought with this thesis is the creation of new features that allow the user to go beyond, expand the target to different types of customers and increase the use of Scribit. The idea of creating effects shows an alternative, fun and original way of representing many images, designs and artistic creations.

It can be concluded that not all filters are suitable to be used with any type of image. It is clear that the levels of detail that can be defined with lines and points are greater than those that a larger figure such as circles or triangles can generate. The results obtained depend on the quality, contrast, resolution and brightness, among other parameters of the input images. Thanks to the possibility of modifying certain parameters, the user can decide to participate dynamically in the results obtained, that is, to change the quality of the designs made by Scribit.

The use of SVG vector graphics language is a great choice since its scalability properties allow it to be designed in different sizes without loss of definition and resolution. This gives the user more versatility and flexibility in terms of choosing the size of the design surface, facilitating the use of Scribit.

This thesis project opens the door to endless options for the creation and implementation with the Scribit robot; from the modification of the developed algorithms to the development of new effects that may attract the user's attention. The possibility of using the Scribit attributes to the fullest, including the use of additional colors, would provide an entertaining and colorful touch to many designs, so that everyone will want to use Scribit to decorate, improve and beautify their place of preference.

REFERENCES

[1] [2] [3] [4] [5] Digital image processing: Rafael C. Gonzalez and Richard E. Woods. Third Edition, Pearson.

[6] W3C (2019), "SCALABLE VECTOR GRAPHICS (SVG)"
https://www.w3.org/Graphics/SVG/

[7] WIKIPEDIA, "Scalable Vector Graphics"
https://en.wikipedia.org/wiki/Scalable_Vector_Graphics

[8] W3C (2019), "SVG 2"
https://www.w3.org/TR/SVG2/intro.html

[9] W3C (2019) "Paths"
https://www.w3.org/TR/SVG2/paths.html

[10] W3C (2019) "Shapes"
https://www.w3.org/TR/SVG2/shapes.html

[11] W3C (2019) "Text"
https://www.w3.org/TR/SVG2/text.html

[12] Wikipedia, Delaunay triangulation
https://en.wikipedia.org/wiki/Delaunay_triangulation

[13] WIKIPEDIA, "Bézier curve"
https://en.wikipedia.org/wiki/B%C3%A9zier_curve

- https://www.python.org/doc/
- https://pillow.readthedocs.io/en/stable/
- https://scribit.design/
- https://www.kickstarter.com/projects/1864378255/scribit-turn-your-wall-into-an-interactive-canvas
- https://github.com/PeterBeard/delaunay
- https://www.wolframalpha.com/input/?i=rotate+30+degrees
- https://mathworld.wolfram.com/BezierCurve.html
- https://medium.com/@roberto_arista/how-to-draw-a-wiggle-between-two-points-with-python-and-drawbot-788006c18fb0
- http://blogs.sitepointstatic.com/examples/tech/svg-curves/cubic-curve.html