
Degree Project in Technology

First Cycle, 15 credits

# Computational String Art

Ruben Socha

# Computational String Art

Ruben Socha
Supervisor Olof Runborg

2024

# Abstract

String art is an art form that involves winding string between pins to create images. This paper presents a study on the computational generation of string art, focusing on the mathematical modeling, numerical methods and optimization algorithms involved. We develop a mathematical framework to transform grayscale images into string art representations, utilizing a downsampling operator to mimic human visual perception. This gives rise to a minimization problem. Central to our approach is a greedy algorithm that iteratively minimizes the error between the string art and the original image to approximate a solution to the minimization problem. A detailed parameter study investigates the impact of varying parameters such as the number of strings and the downsampling factor on image quality and computational efficiency. The results highlight optimal parameter ranges and the inherent difficulty in rendering complex geometries.

# Contents

# 1. Introduction

Generative art is a field at the intersection of art and programming which involves creating artwork through algorithmic processes and computational techniques. The computer graphics community has developed numerous techniques for transforming input images into various artistic styles. Such as impressionistic images[4], pen and ink illustrations[7] or the general transfer of one style of painting to other images[5].

String art is a form of creative expression where strings are taut between pins to form an image. Specifically, we consider the case of recreating a given image with string art.

A computational model of string art was discussed by *Birsak et al.*[1]. In their study, a mathematical model of string art is constructed which is used in conjunction with a greedy algorithm to create instructions to recreate an image with string art. These instructions are used to create physical string art with a robotic arm. Improvements to the heuristic used in the greedy algorithm were investigated by *Demoussel et al.*[3]. In this study, string opacity was also considered.

In this paper, we explore the computational aspects of string art, aiming to formalize and optimize the process of generating string art of a given image. We begin by defining a precise mathematical model of string art. This model gives rise to a minimization problem. When measuring the error between the given image and the string art we need to consider how we perceive the string art.

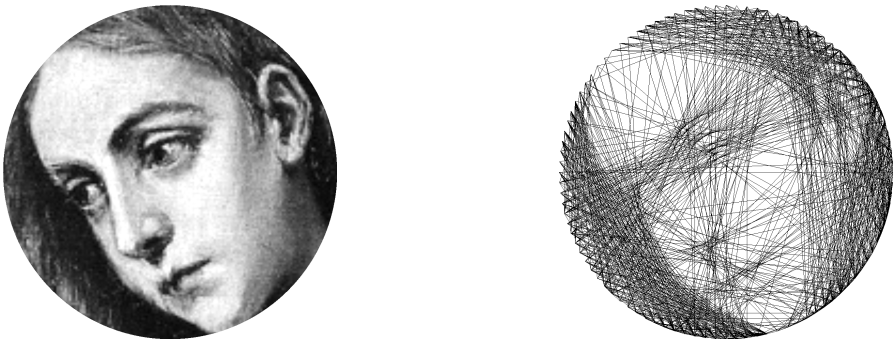
A key component to our approach is the development and implementation of numerical methods, particularly the greedy algorithm. This algorithm iteratively adds strings that minimize the error between the string art and the original image at each step, providing an efficient method of approximating solutions to the minimization problem. This algorithm is further optimized to improve computational efficiency at the cost of slightly decreased image quality.

Our study also includes a detailed numerical parameter analysis, examining how varying parameters such as the number of pins affects image quality and computational efficiency of the resulting string art. Through this analysis we identify optimal ranges of these parameters and highlight the trade-offs involved in their selection.

# 2. Mathematical Formulation

In this chapter, we define a precise mathematical framework for computational string art. Beginning with a gray scale image represented as a matrix of pixel values, we explore how this can be transformed into a vector suitable for simulating string art. We define parameters, further discussed in Chapter 4. Key concepts such as the line drawing algorithm and downsampling are discussed. We also define a metric to compare images. This chapter sets the stage for understanding how computational methods can be employed to systematically approach the creation of string art.

We want to create a non-photorealistic rendering of an image with string art, as seen in Figure 5.1b. A gray scale image can be represented by a matrix of pixel values between 0 and 1, 0 for completely white pixels and 1 for completely black pixels. We take this matrix and stretch it out into a vector. Call this vector $\mathbf{b} \in [0,1]^m$, where $m$ is the total number of pixels. Note that all the input images are circular, meaning that the pixels outside of the circle have a value of 0 for white.

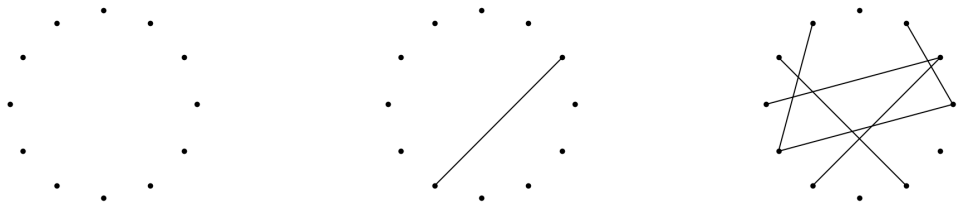


(a) Mary Magdalene

(b) String art of Mary Magdalene

Figure 2.1

We place $N$ pins, evenly spaced around a circle representing the perimeter of the image. Each pair of pins can be connected by a string, as illustrated in Figure 2.2, leading to a total of $n = \binom{N}{2}$ possible strings. In a set of strings, each string can either be included or not. We represent this with a 1 or 0 respectively. A set of strings $\mathbf{x}$ can be represented by an element in $\{0,1\}^n$, which we call $\mathbb{B}^n$. The canonical basis of $\mathbb{B}^n$ is $\{\mathbf{e}_1, \ldots, \mathbf{e}_n\}$, where $\mathbf{e}_i$ is the set only containing string $i$.



(a) 12 pins (b) A string (c) Multiple strings

Figure 2.2

In order to compare the string sets with the image $\mathbf{b}$ we need to represent the strings as images. To do so, lines are drawn between pins, as seen in Figure 2.3, these lines only have pixel values of 0 or 1, there is no anti-aliasing. Let $\boldsymbol{\phi}_i \in \mathbb{B}^m$ be the image of string $i$. Then, we can create the linear map $A = (\boldsymbol{\phi}_1 \dots \boldsymbol{\phi}_n)$, where $A : \mathbb{B}^n \to \mathbb{R}^m$. Note, however, that if two strings in $\mathbf{x}$ overlap, $A\mathbf{x} \notin [0,1]^m$. Therefore, we need to introduce a clamping function $\mathcal{C} : \mathbb{R}^+ \to [0,1]$ such that $\mathcal{C} : x \mapsto \min(1, x)$. Then, $\mathcal{C}(A\mathbf{x}) \in [0,1]^m$ is a valid image.

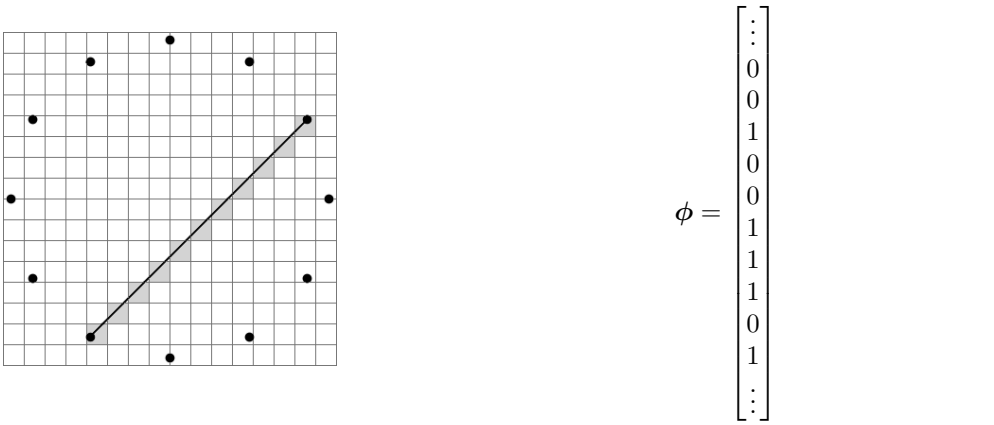
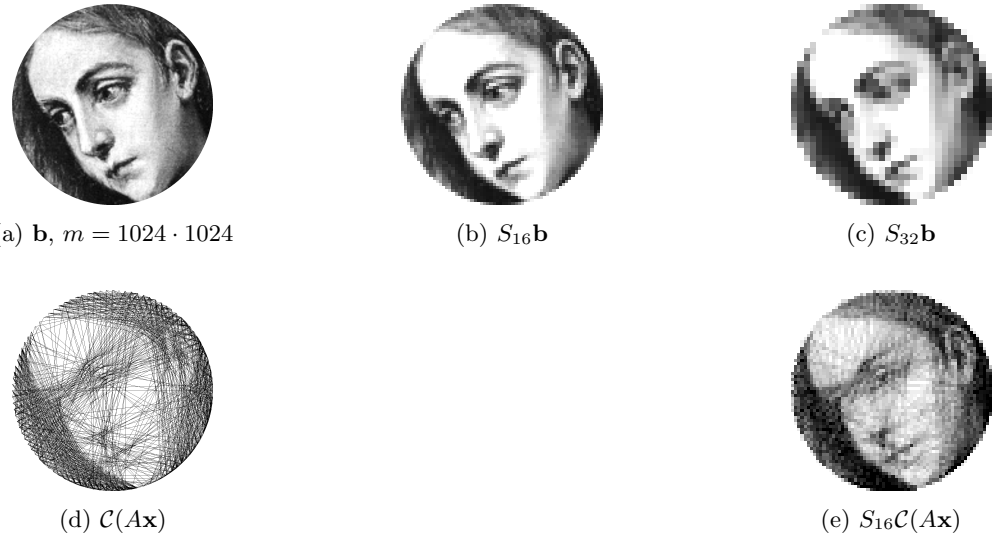



Figure 2.3: String drawn with Bresenham's algorithm and the corresponding basis function $\boldsymbol{\phi}$.

The strings are completely black and opaque, but when viewed from a distance, the eye interprets the density of strings as shades of gray. To model this, we blur the string image with a downsampling operator $S_\sigma : [0,1]^m \to [0,1]^{m/\sigma^2}$. This operator takes an image with $m$ pixels and returns a new image where each pixel is the mean of a $\sigma$ by $\sigma$ grid of pixels of the original image.



(a) $\mathbf{b}$, $m = 1024 \cdot 1024$ (b) $S_{16}\mathbf{b}$ (c) $S_{32}\mathbf{b}$

(d) $\mathcal{C}(A\mathbf{x})$ (e) $S_{16}\mathcal{C}(A\mathbf{x})$

Figure 2.4

To compare images we use the $l^2$ norm, denoted by $\|\cdot\|$, such that $\|\mathbf{b}\|^2 = \sum_{i=1}^{m} b_i^2$. At times, we also use the root mean square. This lets us calculate the error between the string art image and the input image, $\|S_\sigma\mathcal{C}(A\mathbf{x}) - S_\sigma\mathbf{b}\|$. We want to minimize this error, i.e. to find

$$\min_{\mathbf{x}\in\mathbb{B}^n} \|S_\sigma\mathcal{C}(A\mathbf{x}) - S_\sigma\mathbf{b}\|$$

For the line drawing we use Bresenham's algorithm [2], this algorithm draws a line on a grid between two pins, $p_1$ and $p_2$, with coordinates $(x_1, y_1)$ and $(x_2, y_2)$ respectively. First, it calculates $\Delta x$, $\Delta y$ and the direction of the line. Assuming that $x_1 < x_2$, $y_1 < y_2$ and $\Delta y < \Delta x$, the algorithm first draws pixel $(x_1, y_1)$, then determines whether $(x_1 + 1, y_1)$ or $(x_1 + 1, y_1 + 1)$ is closer to the exact line between $p_1$ and $p_2$. This process is repeated until $(x_2, y_2)$ is reached. Bresenham's algorithm provides an efficient way of approximating straight lines on a grid.

# 3. Numerical Methods

In this chapter we provide an overview of the numerical methods employed in finding an approximate solution of the optimization problem, defined in Chapter 2. First, we discuss the image preprocessing needed to convert any image to a structure suitable for our algorithms. Then, we provide a thorough explanation of the greedy algorithm and its variations. Lastly, we give a brief overview of alternative methods.

## 3.1 Image Preprocessing

In computer graphics it is customary to represent black with a 0 and white with a 1. There are two benefits to inverting these. Firstly, this makes adding strings equivalent to vector addition as seen in Chapter 2. Secondly, because strings are thin, the matrix $A$ is mostly zeros, making it appropriate to store in a sparse structure. The number of elements in $A$ is of the order $mN^2$ making $A$ unfeasible to store in a dense structure.

We place the circle of pins inside the image. This means that no strings can be drawn outside of this circle. In order to accurately asses the error between the string art $S_\sigma \mathcal{C}(A\mathbf{x})$ and the input image $S_\sigma \mathbf{b}$ we set the pixels outside of the circle of pins to white.

## 3.2 Greedy Algorithm

Greedy algorithms[6] are a class of algorithms that at each step pick the option that offers the most benefit based on some heuristic. In the case of string art, we choose to add the string that minimizes the error the most at any given point. Conventionally, the choices made by a greedy algorithm are irreversible, however, the number of drawn strings is typically low compared to the total number of possible strings. This means that in addition to adding the string that minimizes the error we can consider removing drawn strings in our comparison without any considerable overhead.

Note that the solution that the greedy algorithm produces is a local minimum as any string we add or remove will increase the error, but this local minimum is generally not the global minimum. We can see this by finding a lower local minimum after the greedy algorithm terminates. This can be done by selecting a random subset of strings and inverting their state. Then, we continue the algorithm from that point until termination. This leads to a new local minimum that could be lower than the original local minimum.

### 3.2.1 Approximating $S_\sigma \mathcal{C}(A\mathbf{x})$

In the algorithm we need to compute $S_\sigma \mathcal{C}(A\mathbf{x})$ often. This has time complexity $O(m \cdot \sigma^2)$ and is the most computationally expensive operation per iteration. We can approximate $S_\sigma \mathcal{C}(A\mathbf{x})$ by pre-computing $S_\sigma A$ and clamping afterwards. This greatly reduces the time complexity at the cost of an increase in error.

If no strings cross the same pixel, then

$$S_\sigma \mathcal{C}(A\mathbf{x}) = \mathcal{C}(S_\sigma A\mathbf{x}).$$

Each pixel of the output image is the average of $\sigma^2$ pixels, an intersection in one of these blocks adds at most $1/\sigma^2$ to the value of the output pixel. With $k$ intersections, the total difference is at most $k/\sigma^2$. The number of pixels of the output image is $m/\sigma^2$. Thus, the difference per pixel between $S_\sigma \mathcal{C}(A\mathbf{x})$ and $\mathcal{C}(S_\sigma A\mathbf{x})$ is at most

$$\frac{k}{\sigma^2}\frac{1}{m/\sigma^2} = \frac{k}{m}.$$

In practice, $k \ll m$, making pre-computing $S_\sigma A$ a good approximation.

### 3.2.2 Pseudo Code

The greedy algorithm, with string removal and approximated $S_\sigma \mathcal{C}(A\mathbf{x})$, can be implemented like this.

1: pre-compute $S_\sigma A$
2: **while** *true* **do**
3: $\quad i \leftarrow \arg\min_i \|\mathcal{C}(S_\sigma A(\mathbf{x} \pm \mathbf{e}_i)) - S_\sigma \mathbf{b}\|$
4: $\quad$ **if** $\|\mathcal{C}(S_\sigma A(\mathbf{x} \pm \mathbf{e}_i)) - S_\sigma \mathbf{b}\| < \|\mathcal{C}(S_\sigma A\mathbf{x}) - S_\sigma \mathbf{b}\|$ **then**
5: $\quad\quad \mathbf{x} \leftarrow \mathbf{x} \pm \mathbf{e}_i$
6: $\quad$ **else**
7: $\quad\quad$ *break*
8: $\quad$ **end if**
9: **end while**

The matrix $A$ can be constructed in $O(\sqrt{m} \cdot N^2)$. Computing $S_\sigma A$ on line 1 is $O(N^2 \cdot m \cdot \sigma^2)$. The time complexity per comparison on line 3 is $O(m/\sigma^2)$, with $\binom{N}{2}$ strings, this is $O(N^2 \cdot m/\sigma^2)$ per iteration.

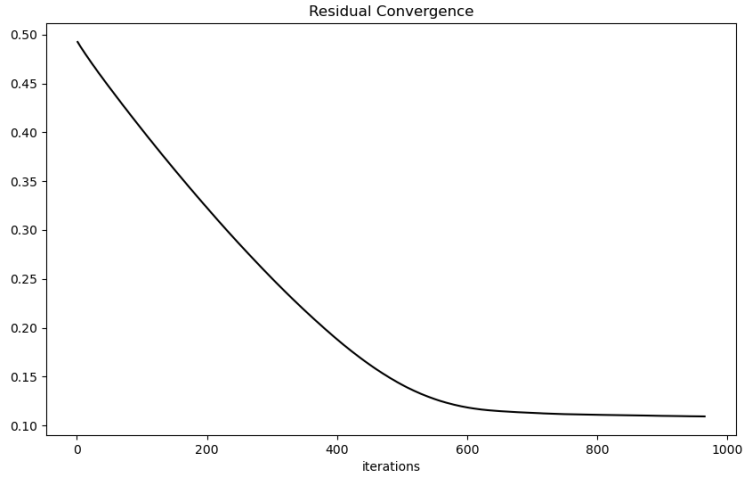



Figure 3.1: Convergence

The convergence plot of Figure 2.4d is shown in Figure 3.1. The error sharply decreases until a plateau at around 600 strings. After that, the improvements are minimal until a local minima

is achieved. Note that the error is always decreasing since the algorithm terminates as soon as this is unachievable.

## 3.3 Other Methods

Here we give a brief overview of alternative methods that can be used to produce string art from an input image.

### 3.3.1 Altered Method of Least Squares

The minimization problem described in Chapter 2 can be solved with the method of least squares over $\mathbb{R}^n$, if we exclude the clamping function $\mathcal{C}$ this would be both faster and simpler. However, string art is represented by a vector $\mathbf{x} \in \mathbb{B}^n$, meaning that these solutions cannot represent string art. Instead, we can solve

$$\min_{x \in \mathbb{R}^n} \|S_\sigma A \mathbf{f}(\mathbf{x}) - S_\sigma \mathbf{b}\|, \text{where } f : \mathbb{R} \to [0, 1].$$

With an appropriate choice of $f$, $\mathbf{f}(\mathbf{x})$ should be close to an element in $\mathbb{B}^n$. Another approach is to implement a penalty method. Then, the minimization problem can be expressed like this

$$\min_{x \in \mathbb{R}^n} \|S_\sigma A\mathbf{x} - S_\sigma \mathbf{b}\| + \lambda \|\mathbf{g}(\mathbf{x})\|, \text{where } \mathbf{g} = \begin{bmatrix} g(x_1) \\ g(x_2 \\ \vdots \\ g(x_n) \end{bmatrix}, \; g(y) = y^2(1-y)^2 \text{ and } \lambda \gg 1.$$

The function $g(y) = y^2(1-y)^2$ penalizes values of $y$ that are far from 0 or 1. Thus, a large $\lambda$ ensures valid string art.

### 3.3.2 Radon Transform

The Radon Transform $\mathcal{R}$ of a function $f(x, y)$ representing an image, maps $f$ onto a set of its line integrals along various directions. It is defined as follows

$$\mathcal{R}(f)(\alpha, s) = \int_{-\infty}^{\infty} f((z \sin\alpha + s\cos\alpha), (-z\cos\alpha + s\sin\alpha))dz,$$

where $\alpha$ is the angle of the line relative to the $x$-axis and $s$ is the perpendicular distance from the line to the origin. These line integrals across the image could potentially be used to determine which strings should be drawn.

# 4. Numerical Parameter Study

In this chapter we conduct a detailed parameter study, analyzing how different variables affect the residual of the optimization problem. We use mathematical reasoning to find boundaries of these parameters. Then, we preform numerical experiments in which we vary one parameter at a time, keeping the others constant. We use the results of these experiments to develop heuristics of how these parameters should be chosen.

## 4.1 Error Dependence

This section is a discussion of how the various parameters of the algorithm and aspects of the input image affect the residual. The residual is the root mean square of the difference between the downsampled string art and input image after applying the greedy algorithm with string removal and pre-computed $S_\sigma A$ to termination. We discuss the fundamental difference between convex and concave images, establish the bounds of values $\sigma$ can assume and the color fidelity of the string art.

### 4.1.1 Convex and Concave Images

Consider a white circle on a black background, if all strings that do not intersect the circle are drawn we get a perfect representation of the circle. This argument can be extended to any convex shape. In contrary, a concave shape, like the lune shown in Figure 4.3d, can never be represented exactly. If we draw a line through the concave area we have to intersect the white area increasing the error. Therefore, some geometries are fundamentally more difficult to recreate with string art.

### 4.1.2 Lower Bound of $\sigma$

When increasing the resolution of the output image without adjusting the number of pins $N$, the white areas increase faster than the black areas. This causes visual artifacts like the Moiré pattern seen in Figure 4.1 to emerge as we have too few strings to cover the image. To counteract this, $N$ needs to grow proportionally to the width, in pixels, of the downsampled output image. After applying the downsampling operator $S_\sigma$, the image consists of $m/\sigma^2$ pixels, so the width is proportional to $\sqrt{m}/\sigma$. Therefore, we need

$$N > C_1 \cdot \frac{\sqrt{m}}{\sigma} \implies \sigma > C_1 \cdot \frac{\sqrt{m}}{N}, \text{ for some constant } C_1. \tag{4.1}$$

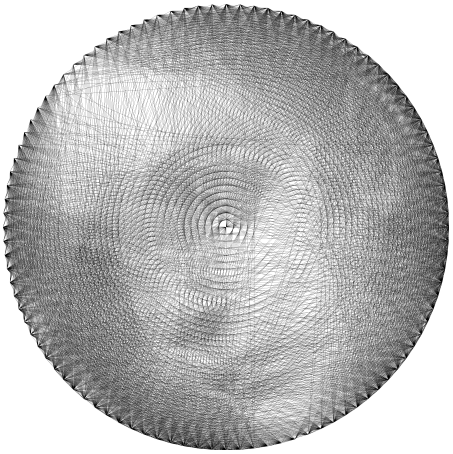


Figure 4.1: Moiré pattern

### 4.1.3 Requirements for a Completely Black Image

In order to make a pixel of the output image black we need to be able to draw enough strings through the $\sigma$ by $\sigma$ grid corresponding to that pixel. Each string passing through the grid can add up to $1/\sigma$ to the average of the grid. Therefore, at least $\sigma$ strings passing through each $\sigma$ by $\sigma$ grid are needed to create a black image. The number strings able to pass through one of these grids is proportional to the number of pins $N$. Thus, we need

$$\sigma < C_2 \cdot N \text{ for some constant } C_2. \tag{4.2}$$

### 4.1.4 Black Resolution in Relation to $\sigma$

Each pixel of the downsampled string art output image is the average of $\sigma^2$ pixels of the string art, where the pixels have either a value of 0 or 1. Therefore, each pixel of the downsampled string art has the value $\mathcal{C}(k/\sigma^2)$, for some $k \in \mathbb{N}$. This effectively means that the output image can have $\sigma^2 + 1$ shades of gray. Hence, increasing $\sigma$ leads to better color fidelity, which in turn leads to more detail being captured and a smaller error.

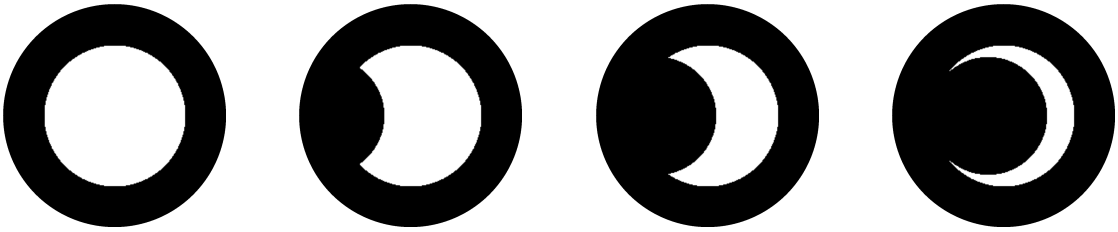
## 4.2 Setup of Study

We chose a set of three images to test our method. The constant image 4.2a serves as the simplest case, in which all pixels have the same value. The lune 4.2b has a concave inner edge making an exact solution impossible. We can vary the concavity of the lune, this gives us an insight of how the algorithm handles geometries of varying complexity. Lastly, we use a portrait 4.2c to examine the algorithms performance in a typical practical application.

(a) Constant image, $\alpha = 0.5$ (b) Lune (c) Mary

Figure 4.2

For all images, we vary the number of pins $N$ and the downsampling factor $\sigma$. Note that changing $\sigma$ alters the resolution of the output image, we want the output image resolution to be constant to make comparisons fair, therefore, we keep $m/\sigma^2$ constant. Because of the fact that strings are always one pixel wide, this results in $\sigma$ being inversely proportional to the string thickness. For the constant image we will vary it's gray scale value, which we denote as $\alpha$. Here, $\alpha = 0$ is a completely black image and $\alpha = 1$ is a completely white image, as opposed to the inverse in our model.



(a) Lune, $\eta = 0$ (b) Lune, $\eta = 25$ (c) Lune, $\eta = 45$ (d) Lune, $\eta = 65$

Figure 4.3

A lune is the set difference of two circles, see examples in Figure 4.3, when the distance of the two circles decreases the inner edge of the lune becomes more concave. We call the intersection distance, in pixels, of the two circles $\eta$ and use it as a proxy for concavity.

## 4.3 Varying $N$

This section examines the impact of varying the number of pins, $N$, on the quality of string art images. For all plots multiple values of $\sigma$ are plotted to see the relation between $N$ and $\sigma$. The output resolution $m/\sigma^2$ is always 32 by 32 pixels. This analysis helps in understanding the optimal number of pins needed for a balance between computational efficiency and image quality.

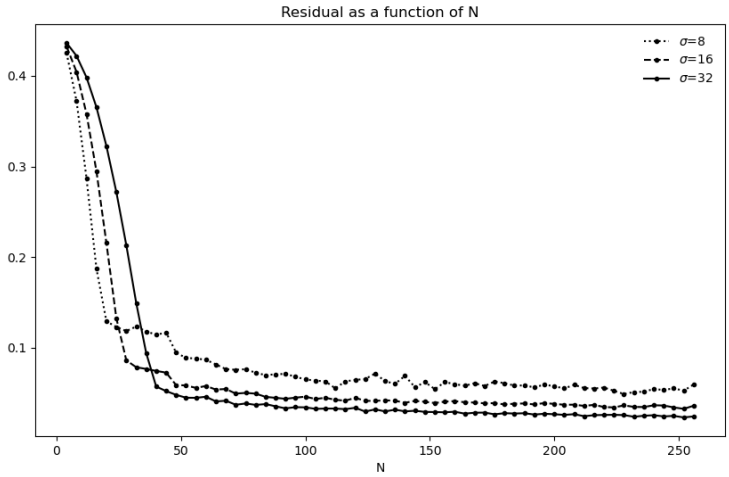



Figure 4.4: Constant image with $\alpha = 0.5$

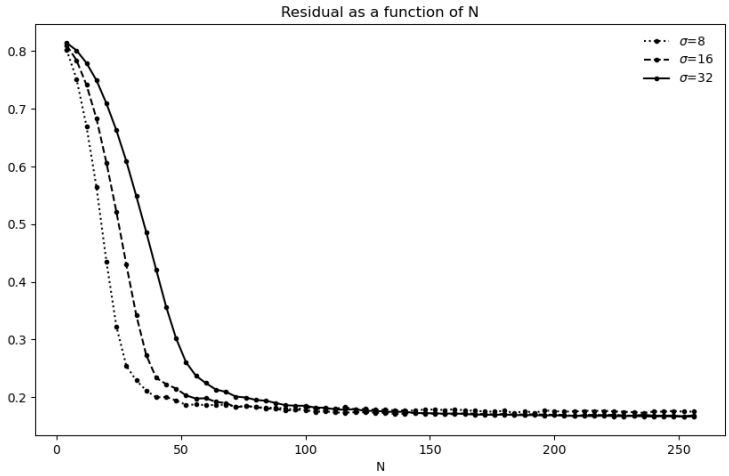



Figure 4.5: Lune, $\eta = 65$

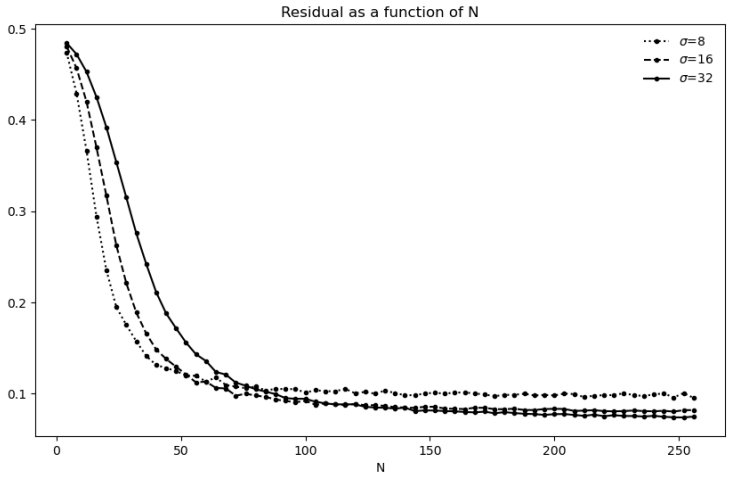



Figure 4.6: Mary

In Figure 4.4, the residual sharply decreases as $N$ increases, reaching a plateau thereafter, indicating marginal or negligible improvement. Additionally, increasing the value of $\sigma$ generally leads to a reduction in error, albeit with diminishing returns. It is important to note that higher values of $\sigma$ necessitate more pins before reaching a plateau. Furthermore, maintaining the output resolution constant while varying $\sigma$ is equivalent to altering string thickness, where higher values of $\sigma$ correspond to thinner strings. Consequently, a higher $\sigma$ requires more strings to sufficiently darken the image, in agreement with (4.2). Similar trends are observed in Figure 4.6, where the disparity between $\sigma = 16$ and $\sigma = 32$ is even more marginal.

In Figure 4.5, the discrepancies among different $\sigma$ values are practically negligible. This may be attributed to the residual being unable to decrease below approximately 0.2. This limitation could stem from the concavity of the lune, where strings must cross the white areas. Thinner strings result in smaller errors when passing through the white areas, but this also necessitates a greater number of strings to achieve a sufficiently dark image. Consequently, these opposing effects offset each other, maintaining a consistent residual value.

## 4.4 Varying $\sigma$

In this section we study how varying the downsampling factor $\sigma$ affects the error. In each plot multiple values of $N$ are shown. The output resolution is 32 by 32 pixels. As discussed in Section 4.1, there is an optimal $\sigma$, outside of which the error increases.

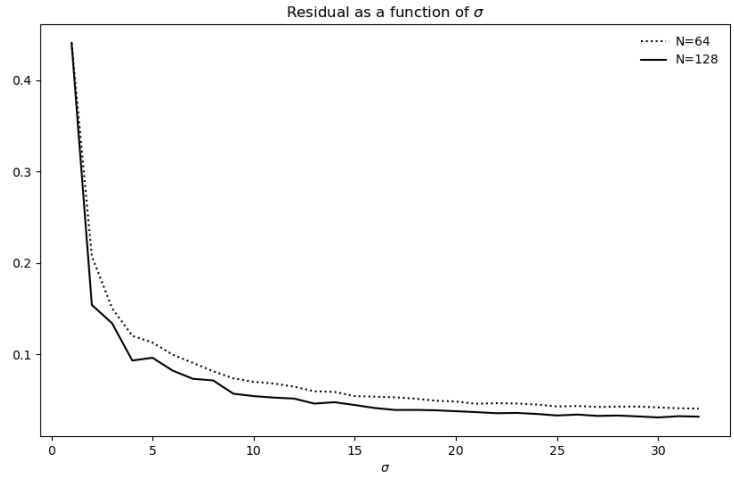



Figure 4.7: Constant image

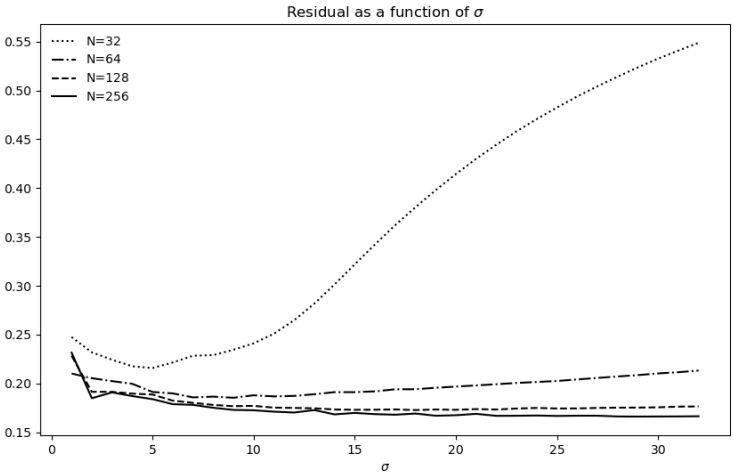



Figure 4.8: Lune, $\eta = 65$ including N=32

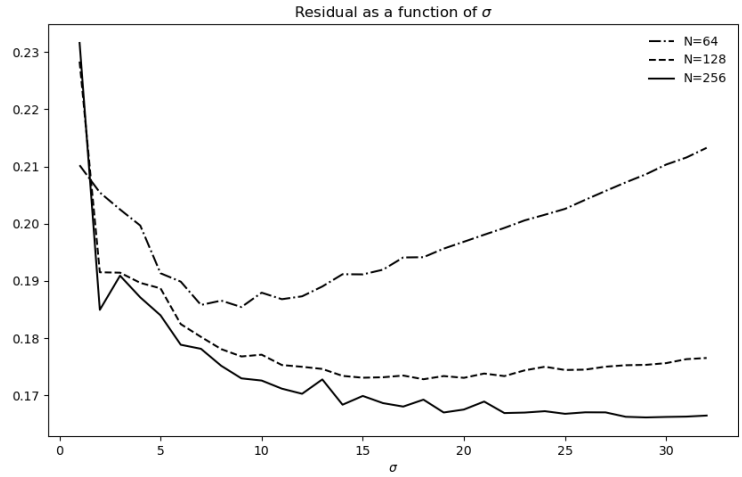



Figure 4.9: Lune, $\eta = 65$

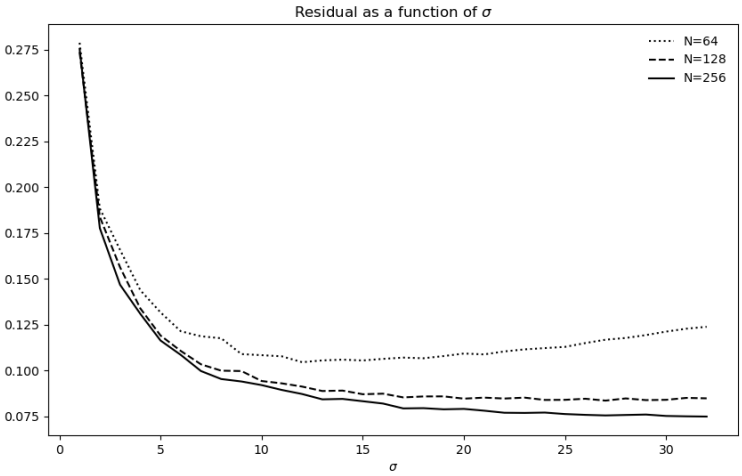



Figure 4.10: Mary

In Figure 4.7, we see that increasing $\sigma$ leads to a reduction in the residual, as discussed in Section 4.1.4, with higher values of $N$ corresponding to lower residual values. However, Figure 4.8 shows that for a specific value of $N$, there exists a minimum $\sigma$ beyond which the residual increases. This trend is further illustrated in Figure 4.9, where similar behavior is observed across various $N$ values. It can be inferred that the error associated with $N = 256$ also begins to increase beyond a certain $\sigma > 32$. Additionally, Figure 4.10 displays a similar minimum for $N = 64$.

For small $\sigma$, the residual drops later when $N$ is large, in agreement with 4.1. From 4.2 we expect that the residual should indeed increase eventually when $\sigma$ gets large and $N$ is fixed.

## 4.5 Varying $\alpha$

Varying $\alpha$ provides insight into how the algorithm handles shades of gray. As mentioned in Section 4.1.4, the range of shades we can produce is tied to the downsampling factor $\sigma$. In Figure 4.11, $\sigma = 8$. In Figure 4.12, $N = 128$. The output resolution is 32 by 32 pixels.

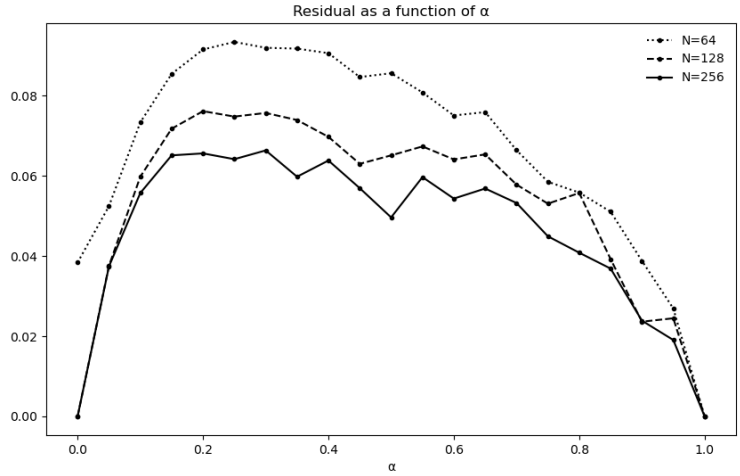



Figure 4.11: Constant image, $\sigma = 8$

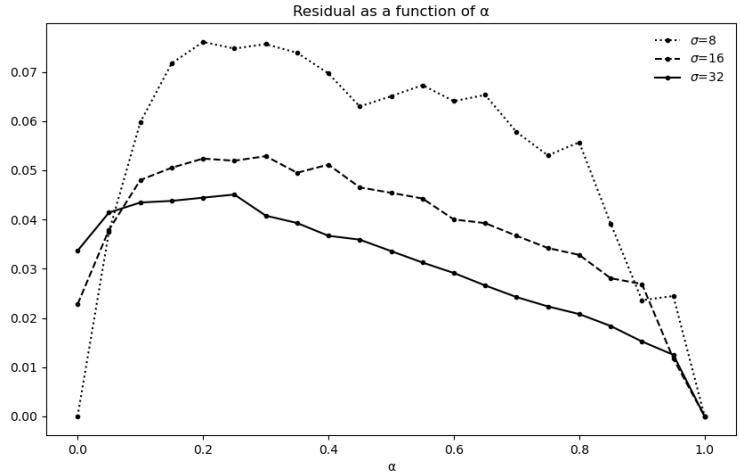



Figure 4.12: Constant image, $N = 128$

In Figure 4.11,we see that increasing $N$ results in a reduction of the residual, although with diminishing returns. Notably, this reduction becomes less pronounced for images with extreme darkness or brightness. Note that for $N = 64$, the image cannot achieve complete darkness as it

has too few possible strings, as predicted by (4.2). Figure 4.12 displays a similar effect, where the image fails to attain complete darkness due to excessively thin strings. This issue could be addressed by adding more pins.

## 4.6 Varying $\eta$

Here, we vary $\eta$, the proxy for concavity. With $N = 128$, $\sigma = 8$ and a 32 by 32 pixel output image.

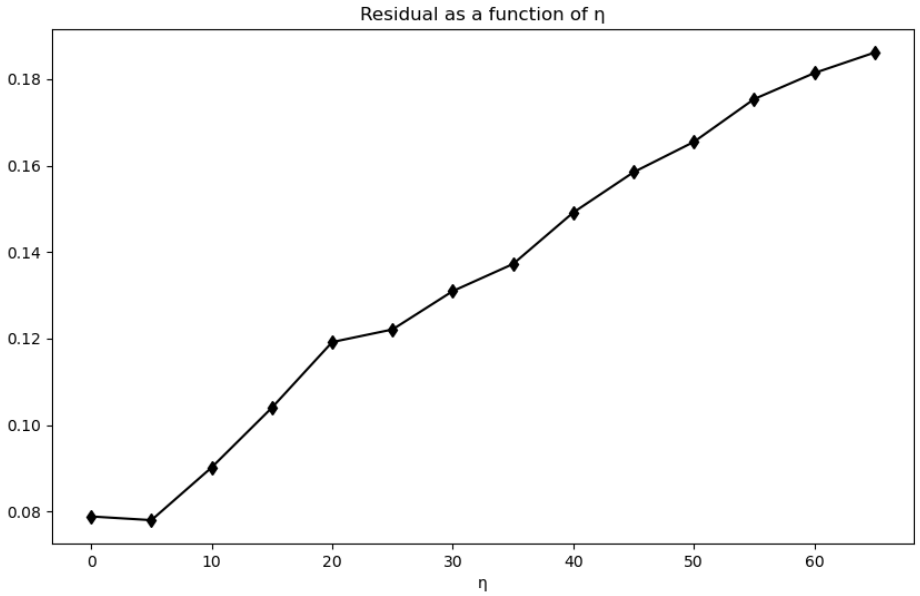



Figure 4.13: Lune, varying $\eta$

In Figure 4.13 we see an unexpected result. Firstly, for $\eta = 0$ the input image is completely convex so we expect an error of 0, Secondly, $\eta = 5$ has a lower error than $\eta = 0$. Both of these anomalies can be explained by the downsampling. When we downsample the perfect circle $\eta = 0$, the edge becomes a gradient instead of a sharp line, our algorithm cannot replicate this gradient exactly, thus we get an error. For the second case, $\eta = 5$, the concave area is so slight that after downsampling that edge becomes a straight line. Presumably, the algorithm handles the gradient of this line better than the edge of the circle, which could be why the error is lower than for $\eta = 0$.

For the remaining points we see an increase in error as the concavity increases, as expected. Concave images are more difficult for our algorithm, as discussed in Section 4.1.1.

## 4.7 Conclusion

From our parameter study, we can draw a number of conclusions. Firstly, varying the number of pins $N$ shows that increasing $N$ reduces the residual error, with diminishing returns. This suggests that beyond a certain point, adding more pins does not significantly improve image quality. Increasing $N$ also greatly increases the computational cost. This indicates an optimal

range for $N$ that balances computational efficiency and error minimization.

Secondly, varying the downsampling factor $\sigma$ revealed that for each $N$ an optimal $\sigma$ exists, as discussed in Section 4.1. With a large $N$ we will realistically not choose a $\sigma$ large enough to increase the error. A good rule of thumb when selecting $\sigma$ is to look at $S_\sigma \mathbf{b}$ and choose the largest $\sigma$ such that $S_\sigma \mathbf{b}$ is still discernible, then choosing $N$ such that the image can be black when covered by all the strings. Note that here $m$ is fixed, unlike in the tests where $m$ is dependant on $\sigma$.

These findings coincide with the discussion in Section 4.1 and provide a framework for choosing parameters that lead to a decreased error. However, even with well chosen parameters image quality is not guaranteed. Some images are fundamentally more difficult to recreate and lead to poor string art. A largely convex image with a light center, dark edges and a simple geometry tends to produce good string art with the greedy algorithm. Portraits usually have theses qualities and thus tend to lead to good string art representations.

# 5. Examples

Here, we showcase a collection of string art created by the greedy algorithm with string removal and pre-computed $S_\sigma A$.

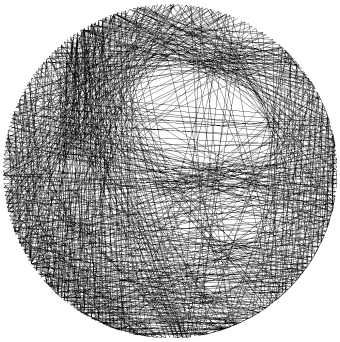
(a) (b)

Figure 5.1: Mary Magdalene

(a) (b)

Figure 5.2: Dog

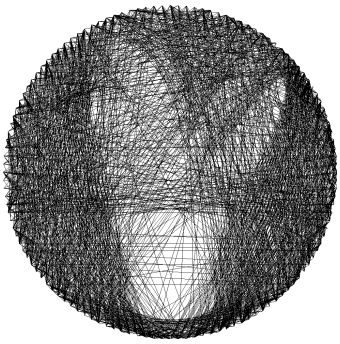
(a)



(b)

Figure 5.3: Albert Camus

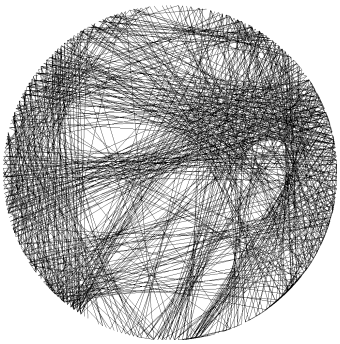
(a)



(b)

Figure 5.4: Plant

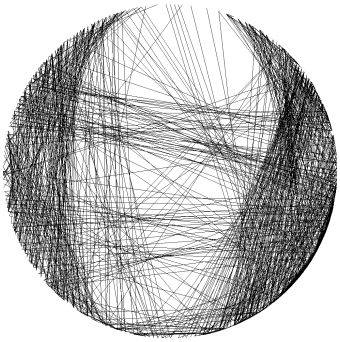
(a)



(b)

Figure 5.5: Bobby Fischer

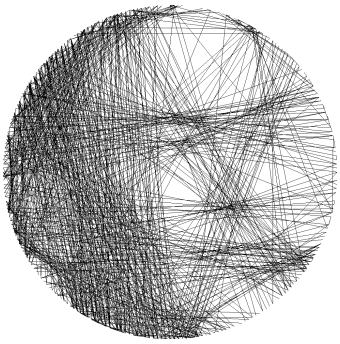
(a)



(b)

Figure 5.6: Ada Lovelace

(a)



(b)

Figure 5.7: Frida Kahlo

| Image | $N$ | $\sigma$ | $m$ | Number of Iterations | Time to Compute[s] |
|---|---|---|---|---|---|
| Mary | 128 | 16 | 1024 by 1024 | 965 | 260 |
| Dog | 256 | 16 | 1024 by 1024 | 1131 | 1166 |
| Camus | 256 | 16 | 1024 by 1024 | 867 | 937 |
| Plant | 128 | 16 | 1024 by 1024 | 1650 | 396 |
| Fischer | 256 | 16 | 1024 by 1024 | 881 | 945 |
| Lovelace | 256 | 16 | 1024 by 1024 | 1088 | 1183 |
| Kahlo | 256 | 16 | 1024 by 1024 | 620 | 782 |

# 6. Conclusion

The discrete problem of making string art that best represents a given image can be effectively modeled as a minimization problem over $\mathbb{B}^n$. Solutions of this minimization problem can be approximated with a greedy algorithm. This algorithms finds a local minimum, not a global minimum. This local minimum often produces discernible string art.

The parameter study underscores the importance of selecting appropriate values for the number of pins $N$ and the downsampling factor $\sigma$. However, it also shows that even with proper parameter selection high quality string art is not guaranteed. Some images are simply more difficult for the greedy algorithm.

# Bibliography


[1] Michael Birsak, Florian Rist, Peter Wonka, and Przemyslaw Musialski. String art: towards computational fabrication of string images. In *Computer Graphics Forum*, volume 37, pages 263–274. Wiley Online Library, 2018.

[2] Jack E Bresenham. Algorithm for computer control of a digital plotter. In *Seminal graphics: pioneering efforts that shaped the field*, pages 1–6. 1998.

[3] Baptiste Demoussel, Caroline Larboulette, and Ravi Dattatreya. A Greedy Algorithm for Generative String Art. In *Bridges 2022 ( MathematicsArt MusicArchitectureCulture )*, Aalto, Finland, August 2022.

[4] Paul Haeberli. Paint by numbers: abstract image representations. *SIGGRAPH Comput. Graph.*, 24(4):207–214, sep 1990.

[5] Aaron Hertzmann, Charles Jacobs, Nuria Oliver, Brian Curless, and David Salesin. Image analogies. *Proceedings of ACM SIGGRAPH*, 2001, 06 2001.

[6] Tardos Kleinberg. *Algorithm Design*, chapter 4. Pearson, 2014.

[7] Michael Salisbury, Sean E. Anderson, Ronen Barzel, and D. Salesin. Interactive pen-and-ink illustration. *Seminal Graphics Papers: Pushing the Boundaries, Volume 2*, 1994.

TRITA – XXX-XXX 20XX:XX
Stockholm, Sweden 2024

www.kth.se