EUROGRAPHICS 2018 / D. Gutierrez and A. Sheffer
(Guest Editors)

Volume 37 (2018), Number 2

String Art: Towards Computational Fabrication of String Images

Michael Birsak[1] Florian Rist[1] Peter Wonka[2] Przemyslaw Musialski[1]

[1]GCD / TU WIEN [2]VCC / KAUST

Figure 1: *Closeup photography of a string art image generated with our computational fabrication pipeline.*

Abstract

In this paper we propose a novel method for the automatic computation and digital fabrication of artistic string images. String art is a technique used by artists for the creation of abstracted images which are composed of straight lines of strings tensioned between pins distributed on a frame. Together the strings fuse to a perceptible image. Traditionally, artists craft such images manually in a highly sophisticated and tedious design process. To achieve this goal fully automatically we propose a computational setup driven by a discrete optimization algorithm which takes an ordinary picture as input and converts it into a connected graph of strings that tries to reassemble the input image best possibly. Furthermore, we propose a hardware setup for automatic digital fabrication of these images using an industrial robot that spans the strings. Finally, we demonstrate the applicability of our approach by generating and fabricating a set of real string art images.

1 Introduction

String art is a technique for the creation of visual artwork where images emerge from a set of strings that are spanned between pins. There are several artists that have experimented with the creation of string art using a proprietary algorithmic solutions in combination with manual crafting [Vre16] or automatic fabrication [Laa18]. In this paper, we propose a formal treatment of the problem and an automatic computational fabrication pipeline to generate string images that resemble a gray scale image given as input.

Our goal is to compute the string path around the given pins and to fabricate the string images automatically using a robot. While the pins can be placed in any pattern, we focus on a setup where pins are distributed along a circular frame which encloses a canvas. In addition, we would like the image to be generated by a single string without cutting it. Please refer to Figure 2 which depicts this process.

The main challenge of computational string art is the proper formulation of an optimization problem that computes the path of the string. There are many parameters and design choices leading to very different optimization problems. For example, the selection of parameters, like the number of pins and the image resolution used in the computation, drive the scale of the problem. The image formation model influences the complexity of optimization.

Considering all possible factors leads to an overly ambitious modeling of the problem that is discrete, very-large scale, and nonlinear. Therefore, the best known algorithm for generating string art proposes to relax the problem into a least squares formulation and employ thresholding [Var17]. This leads to a reasonable baseline solution, but there are still too many visual artifacts.

Based on a large set of experiments, we confirmed that it is reasonable to split the problem into two parts. The complete string path is broken down into individual string segments from one pin to another. The first part of the optimization is concerned with selecting a subset of individual string segments and the second part of the

optimization computes a sequence of string segments. However, we also observed that the relaxation of the string segment selection leads to large errors and that it is better to formulate the problem as a non-linear binary least squares problem directly. While this problem is NP-hard and cannot be solved exactly in feasible time, we propose an iterative greedy algorithm that clearly outperforms the relaxed version of the problem in practice.

Contributions. We propose a framework for the computation and automatic fabrication of string images. We present an analysis of the important parameters in the fabrication process in Section 3, a formulation of the problem as an optimization task with various objectives in Section 4, an iterative greedy approximation solver for binary problems in Section 5, an improved Eulerian path extraction algorithm in Section 6, and an automatic fabrication setup using an industrial robotic arm in Section 7.

2 Related Work

The generation of artistically looking images has a long tradition in the computer graphics community. There have been many approaches for the abstraction and creation of images both for digital as well as for physical purposes. All these techniques have one in common: there is some input example (often just an image), and there is a computational pipeline that transforms the input into a specific artistic style, appearance, or a physical phenomenon.

Abstractions and Conversions. There are couple of pioneering works that aim at abstraction of input photographs such that they mimic some artistic techniques, like impressionistic images [Hae90], pen and ink illustrations [SABS94], or general transfer of one style of painting to other images [HJO*01]. Elber [Elb10] proposed a method for creation of abstract 3d ortho-pictures from sets of various 2d inputs.

Also other types of abstractions have been developed, as for instance generation of halftone QR codes [CCLM13] or creating personalized jigsaw puzzles from input images [LSS*14]. Also methods for the conversion of low-resolution pixel images to vector graphics have been proposed [KL11].

Shadows and Perspective. Another branch of works aims at the generation of images using controlled shadow casting. One impressive work proposes a system for optimization of 3d objects in order to cast varying shadow images [MP09] if lit from different directions. Alexa and Matusik proposed a method which generates images by self-occlusion of a large set of small tubes of varying length [AM11]. The idea of images cast by shadowing has been further explored by Bermano et al. [BBAM12], who proposed a method to create multiple self-shadowing pictures which emerge depending on the direction the surface is lit from, and by Baran et al. [BKB*12], who proposed a set of layers which cast a desired shadow if lit appropriately.

Recently, Zhao et al. [ZLW*16] proposed an approach for the computation of perforated lampshades for continuous projective images which emerge as shadows on the walls if a lamp is lit. Another very interesting method proposed by Schüller et al. [SPSH14] creates so-called appearance-mimicking surfaces, where an image is distributed across different geometric objects and emerges as a relief only if viewed from a very specific position.

Light and Optics. Yet another set of methods aim at the generation of images from reflections or other light effects, especially by fabricating surfaces of objects with controlled appearance and reflectance properties. Weyrich et al. [WPMR09] proposed a method for the fabrication of small geometric patches whose surfaces exhibit desired appearance, such that e.g., their reflection casts a specific pattern. These ideas were further extended, e.g., by Levin et al. [LGX*13]. Other interesting image generation methods utilize effects caused by light transport through (semi) transparent media [PJJ*11, YIC*12, STTP14]. These works introduce solutions how to create controlled caustics using inverse light transport.

Reliefs and Fabrication. Reliefs are yet another technique for creation of abstracted physical images. They were used by artists since ancient times, and also in the computer graphics community, their computation from 2d [AM10] and 3d objects [WDB*07] with further digital fabrication using CNC milling has been proposed. Reliefs have been also used to make existing artworks more accessible by blind and visually impaired people [RMP11].

More recently, techniques for the texturing of physical reliefs or 3d objects [PDP*15, ZYZZ15] as well as for the generation of spatial physical shapes from flat sheets using thermoforming have been proposed [SPG*16]. In both cases the problem is to map the image appropriately to the deformed surface.

Computer and Robot Aided Arts. Utilization of robots for image generation has also been already approached. For instance, Lindemeier et al. [LPD13] intorduced a feedback loop robot-setup which repaints given target images in with a certain abstraction. Another setup uses drones that are automatically controlled in order to paint stipples on the canvas in order to reproduce an input image [GKAK16].

Computationally aided-painting has been also introduced. For instance, Prevost et al. [PJJSH16] introduced a spray-painting system based on motion-tracking and computer-controlled release of the spray-valve, and Shilkrot et al. [SMPZ15] an augmented air-brush system.

Finally, also in the HCI community, questions of the utilization of robots for arts [SY17] have been raised.

Our method, however, does not claim to generate arts. Indeed, our method is an automation of a known technique which generates artistically looking results from existing input samples.

3 Fabrication Considerations

The physical setup we propose is the following: we use a (circular) canvas of a certain diameter and distribute pins along its frame. Then we span strings between the pins following an Eulerian path using an industrial robot. In Section 7 we explain the details of the fabrication process. However, in order to model the problem formally as a computational task and at the same time to meet the requirements for digital fabrication, we cast a set of assumptions and requirements:

(R1): We assume that a physical string is entirely opaque. In practice, the polyester overlock thread we have used meets this assumption very well. This implies that drawing the same string multiple times results in the same outcome, hence, the question if a particular string between two pins is drawn or not is *binary* (i.e., $\in \mathbb{B}$).

(R2): Further, we discretize the canvas with a regular grid and assume that the physical string thickness t in combination with the

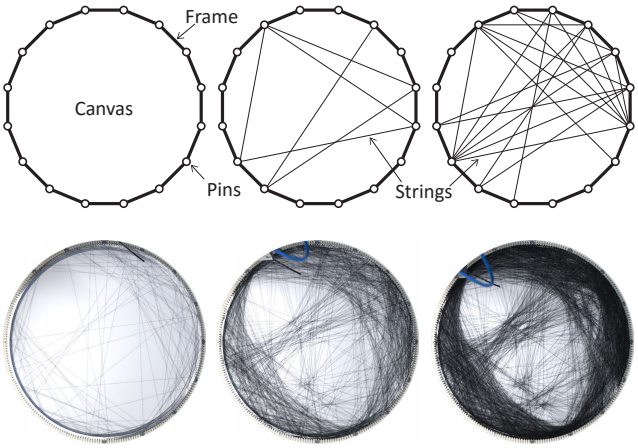


Figure 2: *Top: three schematic steps of the string image generation process. The setup is a frame that surrounds a canvas. The pins are distributed along the frame. Then strings are spanned between the pins until they fuse to a perceptible image. Bottom: three steps of a real string image generation process with 256 pins.*

diameter of the canvas d induce its resolution Z, in particular:

$$Z = \frac{1}{t} d\,.$$

This means that if a string is drawn horizontally or vertically and it passes through the middle of a pixel, it covers it entirely. The rationale behind this assumption is that it gives us a unique assignment of the physical domain to the computational model, such that, in general, our model is independent of the used string thickness or frame size. However, please note that it now influences the computational complexity of the model.

In practice we work with a canvas of $d = 630$ mm and a thread with a thickness $t = 0.15$ mm and thus with a full grid resolution of $Z = 4{,}096 \times 4{,}096$ pixels.

(R3): We account for the pin diameter (or width), which implies that a connection between two pins in practice can be drawn in four different ways, as shown in Figure 6. In our setup, a pin has a rectangular cross section with a width of 2 mm, which corresponds to approx. 13 pixels on the canvas. Working with single connections would thus introduce a significant error and moreover, also disturbing Moire patterns.

(R4): Eventually, we also take into account that the output needs to be fabricable using one long thread, which means that the finally generated path needs to be an Eulerian path.

In order to meet all these requirements we have developed a computational fabrication model which converts an input target image into a string artwork that is automatically fabricated by a robot.

4 Problem Formulation

The input to our framework is an ordinary grayscale image. We convert it into a real number representation in the range of $[0,1]$ and denote it further as the column vector

$$y \in [0,1]^m \subset \mathbb{R}^m\,,$$

where m is the number of all pixels, concatenated row-wise. The output of the optimization algorithm is a binary array, where each entry corresponds to an edge which could be drawn on the canvas. Activated bins reflect edges which need to be drawn. We denote the output as the column vector

$$x \in \mathbb{B}^n\,,$$

where n is the number of all possible edges. It has the dimensionality of

$$n = 4\binom{p}{2}\,,$$

where p is the number of pins on the frame of the canvas. Please note that $\mathbb{B}^n$ is an extended space of edges which includes all four possibilities to connect two pins as required by **(R3)** as depicted in Figure 6.

In the following, the goal of our efforts is to determine the best way to define a mapping F from the space of edges to the space of pixels, i.e.,

$$F\colon \mathbb{B}^n \to [0,1]^m \quad \text{with} \quad x \mapsto F(x)\,,$$

and to determine the values of the elements of the vector x such that it delivers the *best approximation* of the input image. Best means here to reach the highest possible visual quality of the fabricated result which needs to be—of course—judged qualitatively. However, in order to quantify this problem to treat it computationally, we cast it as an optimization task of the form:

$$\min_x \|F(x) - y\| \quad \text{s.t.} \quad x \in \mathbb{B}^n$$

under a certain norm $\|\cdot\|$.

4.1 Linear Least Squares

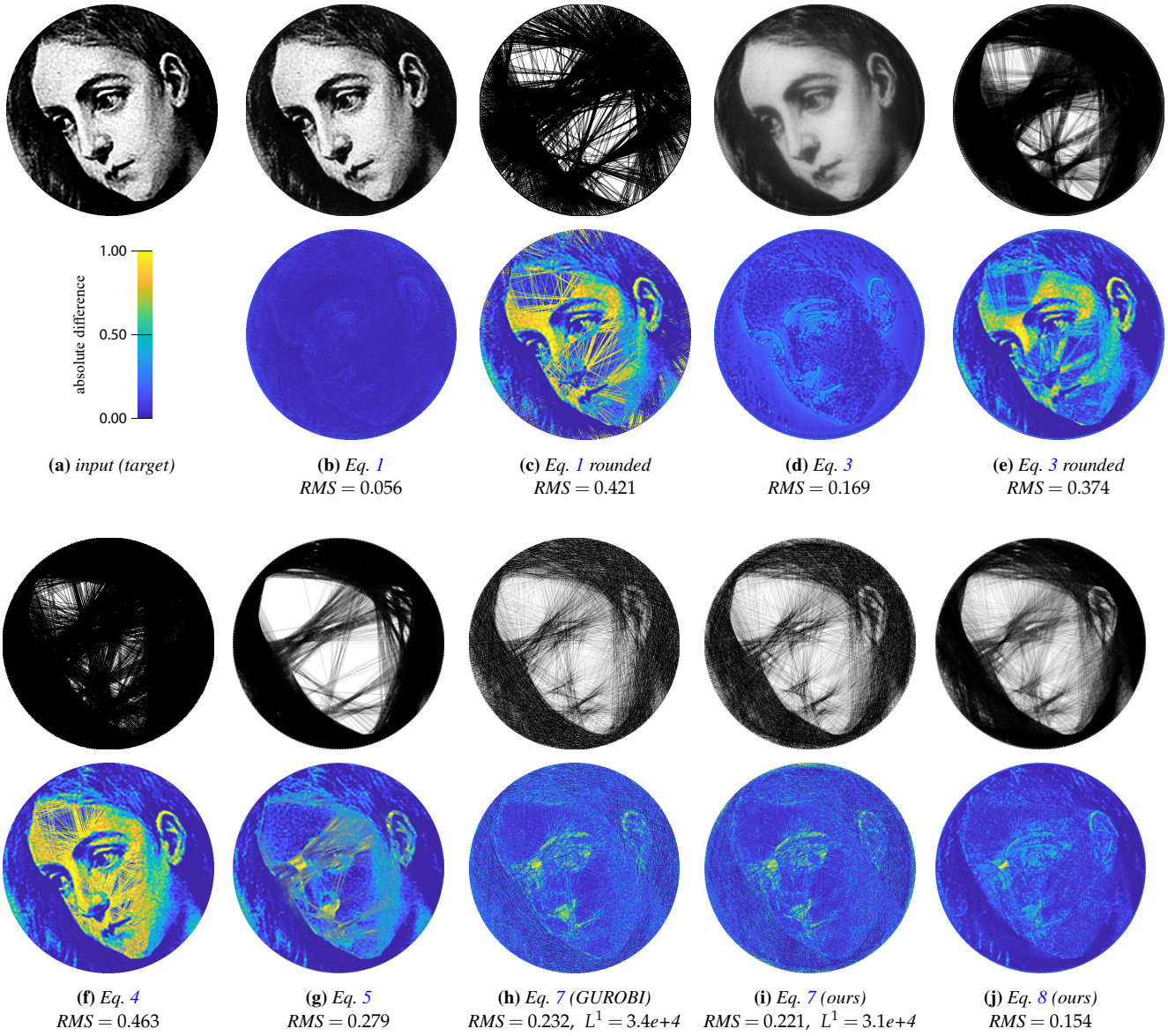
Using least squares is the simplest baseline approach [Var17]. The goal is to determine the values of the vector x such that the squared l^2-norm (further denoted as $\|\cdot\|_2^2$) between the input image and the approximated image is minimized in the least squares sense. In the simplest case, we could cast the problem as an ordinary linear least squares problem of the form:

$$\min_x \|Ax - y\|_2^2 \quad \text{with} \quad A\colon \mathbb{R}^n \to \mathbb{R}^m\,, \tag{1}$$

where A is a linear mapping from the space of edges to the space of pixels. The rows of the matrix A correspond to pixels and columns to edges and in the simplest case it contains ones in all entries A_{ij} if a pixel i is (partially) covered by an edge j and zero elsewhere. The coverage of the pixels can be determined by drawing the edge lines on the canvas separately using a discrete line drawing algorithm (e.g., Bresenham [Bre65]). An output image can be then created as a linear combination of the strings as $z = Ax$.

In practice, in order to increase the accuracy, we construct the matrix by rendering each edge with an anti-aliasing algorithm (e.g., Wu et al. [WXWX91]) and use the resulting grayscale values as each pixel's contribution, such that $A_{ij} \in [0,1]$.

If $n < m$, the problem is overdetermined and it results in a positive definite quadratic function with a unique solution, which can be found very efficiently. We used a sparse iterative conjugate gradient based solver (e.g., LSQR [PS82]]) which needs less then 5 seconds for $m = 262{,}144$ and $n = 130{,}560$ on a standard PC. Figure 3b



(a) *input (target)*

(b) *Eq. 1* $RMS = 0.056$

(c) *Eq. 1 rounded* $RMS = 0.421$

(d) *Eq. 3* $RMS = 0.169$

(e) *Eq. 3 rounded* $RMS = 0.374$

(f) *Eq. 4* $RMS = 0.463$

(g) *Eq. 5* $RMS = 0.279$

(h) *Eq. 7 (GUROBI)* $RMS = 0.232,\ L^1 = 3.4e{+}4$

(i) *Eq. 7 (ours)* $RMS = 0.221,\ L^1 = 3.1e{+}4$

(j) *Eq. 8 (ours)* $RMS = 0.154$

Figure 3: *Comparison of the output images given by different objective functions we have tested in this paper. The first image is the target. Each result is given as its reconstructed image and the difference image to the target. First row of results are the linear least squares approaches (3b-3e). Second row are binary approaches. Please note that even if results 3b and 3d provide very good reconstructions, they violate fabrication constraints and cannot be physically produced.*

depicts the result and an error image. Unfortunately, the minimizer x is real-valued and can contain large (> 1) as well as negative values, which would mean that an edge can be (partially) subtracted from the canvas. Indeed, the decision if an edge should be drawn or not is binary, thus, this solution violates requirement **(R1)**.

A simple remedy would be rounding to the nearest integer with an operator like

$$R\colon \mathbb{R}^n \to \mathbb{B}^n \quad \text{with} \quad u \mapsto R(u) = \text{round}\left(\frac{u - \min(u)}{\max(u - \min(u))}\right),$$

such that the final image is given by

$$z = C(AR(x)),$$

with the clamping function

$$C\colon {\mathbb{R}_+}^m \to [0,1]^m \quad \text{with} \quad u \mapsto C(u) = \min(1,u). \qquad (2)$$

Please note that since all entries of A are nonnegative, the result $AR(x)$ is also nonnegative, thus we need only to clamp at the upper bound of the range. This procedure is, however, very inaccurate and does not provide satisfactory results, as evident in Figure 3c. Also the baseline approach of [Var17] proposed a similar operation,

under the assumption of a non-opaque string and thus quantizing the result to a (small) integer value. However, their results have not been fabricated.

More importantly, this solution does not take the resolution constraints **(R2)** into account, which states that the output image resolution is induced by the physical thickness of the thread.

We could address it by upsampling the input y to the necessary resolution using an upscaling (w.l.o.g. a linear) operator $U\colon \mathbb{R}^m \rightarrow \mathbb{R}^{sm}$, and obtain

$$\min_x \left\| \tilde{A}x - Uy \right\|_2^2 \quad \text{with} \quad \tilde{A}\colon \mathbb{R}^n \rightarrow \mathbb{R}^{sm}, \tag{3}$$

where the matrix $\tilde{A}$ maps the output array x to a supersampled image space of real values $\mathbb{R}^{sm}$. The scaling factor s equals the number of used superpixels per input pixel. The resolution is chosen in such a way that a superpixel fulfills the fabrication requirement **(R2)**. Please refer to Figure 3d for the resulting image.

However, also in this case the output is not binary and hence still violates **(R1)**. Applying the rounding operator R again delivers very inaccurate results, as depicted in Figure 3e.

Obviously, a crucial disadvantage of the least squares approaches is their non-binary output. One solution which could help would be constraining the problem to $x \in \mathbb{R}_{0,+}$ or even better to $x \in [0,1]$, however, our efforts to solve it with MATLAB failed due to memory limitations.

For this reason, we further focused on a solution with a hard constraint that enforces x to strictly be $\in \mathbb{B}^n$, which in fact is an NP-hard problem. While there exist approaches to relax it and address it more efficiently [QS08, CZ07], they are currently still limited in the size of the problem they can deal with. In particular, in our current setup, we require a number of $130{,}560$ edges and 4096^2 pixels, which is a too large number for MILES [CZ07] that only works with dense matrices.

4.2 Binary Non-Linear Least Squares

At this point we conclude that in order to meet the requirement **(R1)**, we need to solve a binary optimization problem and to meet requirements **(R2)** and **(R3)**, the solution needs to scale to a very large number of variables. Hence, we propose to solve with an iterative greedy solver described in detail in Section 5. Using this solver we can cast the problem as

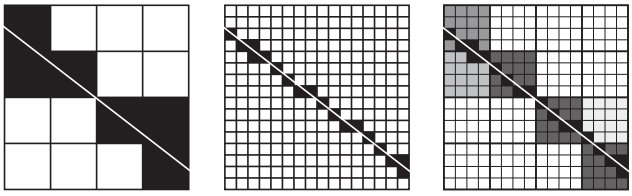
$$\min_x \left\| C(Ax) - y \right\|_2^2 \quad \text{s.t.} \quad x \in \mathbb{B}^n, \tag{4}$$

in order to solve it at the lower resolution or, to meet criterion **(R2)** as

$$\min_x \left\| C\left(\tilde{A}x\right) - Uy \right\|_2^2 \quad \text{s.t.} \quad x \in \mathbb{B}^n. \tag{5}$$

In both cases C is a clamping operator as defined in Eq. 2 with appropriately chosen dimensionality m or sm. It brings pixel values back to the range $[0,1]$ and is the actual non-linear component of the objective. Please note that clamping at this stage does not falsify our computation because, since the string is opaque, putting one or more layers over one other always results in a value ≥ 1. Thus, clamping to 1 does not change the result.

Figures 3f and 3g show the results of optimization using the objectives in Equations 4 and 5 respectively.



(a) (b) (c)

Figure 4: *Lines sampled directly on the grid lack in range. Whether sampled in lower resolution (a) or in high resolution (b), each pixel is either black or white. Our solution (c) is to sample lines in superresolution and to filter them down to introduce more range per pixel in the lower resolution (cf. Sec. 4.4).*

4.3 Binary Integer Programming

Finally, a third possible way would be the usage of binary integer programming, to account for the fabrication requirement **(R1)**-**(R3)**. In particular, we could solve the problem:

$$\min_x \left\| \tilde{A}x - Uy \right\|_1 \quad \text{s.t.} \quad x \in \mathbb{B}^n,$$

with $\|\cdot\|_1$ being the sum of absolute differences between pixel values (L^1-norm). Although it is also an NP-hard problem, there exist mature algorithms which can approximate a solution within both reasonable time and error bound. We have formulated this task and tried to solve it using a state-of-the-art integer programming library (GUROBI [Gur16]), however, it turns out that it is too large to be solved using commodity hardware. Nonetheless, we still managed to solve a relaxed version of the problem as further explained in Section 4.5

4.4 Range Problem

The investigation of the results so far allows to conclude, that if the problem is solved for a binary minimizer x, we obtain results which are characterized by very strong contrast and either dark or white regions (Fig. 3c, 3e, 3f, 3g). This is not the case with the real-valued least squares results in Fig 3b and 3d.

The reason for that is that the output image lacks an expressive range per pixel. In other words, since each pixel is a linear combination of the edge contributions (which are in the case of the Bresenham algorithm also binary), and it is crossed by many edges, it reaches almost always values ≥ 1.

Whether we compute the norm in low (Eq. 4) or in high (Eq. 5) resolution, we encounter a variant of this problem. This is depicted schematically in Figure 4a and 4b for each resolution respectively.

The effect is also visible in Figure 3, where in low resolution (Fig.3f) the edges are too thick and too many of them are drawn. While in high resolution the edges are thinner, they are drawn in order to account for darker or lighter regions but gray areas in between cannot be expressed due to the lack of range (Fig.3g). Additionally, we should notice that the computation in high resolution is very expensive and the generation of the result in Figure 3g took 14.7 hours.

Our solution to this issue is to render the resulting strings in such a way that they exhibit enough range in order to simulate the resulting

string images. Thus, we cast is as the following problem:

$$\min_x \left\| DC(\tilde{A}x) - y \right\|_2^2 \quad \text{s.t.} \quad x \in \mathbb{B}^n, \tag{6}$$

where the function

$$D\colon [0,1]^{sm} \to [0,1]^m \quad \text{with} \quad u \mapsto D(u) = Bu$$

is a downsampling operator that filters the image back to the target resolution. In our method, we use a box-filter which can be represented as a linear map B, however, higher order filters could also be used.

The rationale is to draw the edges in the high resolution (and hence also to meet requirement **(R2)**), but to compute the norm at the lower resolution and at the same time to increase the range of each pixel by the number of super pixels used (in our case $s = 8 \times 8$, cf. Figure 4c). Using this method we can extend the expressiveness of the output image during the optimization and improve the visual results significantly as shown in Figure 3j. Additionally, since we perform the evaluation of the objective in the low-resolution space, we reduce the computational burden significantly if compared to Eq. 5, where the result in Figure 3j took 2.3 hours.

4.5 Comparison to L^1-norm

Using the extended range introduced above, we were able to solve the problem with the L^1 norm using an adapted binary integer program for the following objective function:

$$\min_x \left\| D\tilde{A}x - y \right\|_1 \quad \text{s.t.} \quad x \in \mathbb{B}^n. \tag{7}$$

We could reach a solution since now the codomain is the lower resolution with a significantly smaller dimensionality.

In order to solve it with GUROBI [Gur16], we have developed a hierarchical multiresolution approach that works only with a subset of possible edges at each resolution. In detail, we compute an image pyramid of our 512×512 input image down to a resolution of 32×32 pixels. Then we solve Equation 7 repeatedly, starting at the lowest resolution (32×32) with all 130,560 edges in the matrix $\tilde{A}$. In each iteration we increase the resolution of the image by taking the image on the next upper level of the pyramid, while at the same time we reduce the number of available edges (= number of columns of $\tilde{A}$). To obtain a coherence across the levels of the pyramid, we use the outcome of each iteration to determine the columns of $\tilde{A}$ for the next iteration.

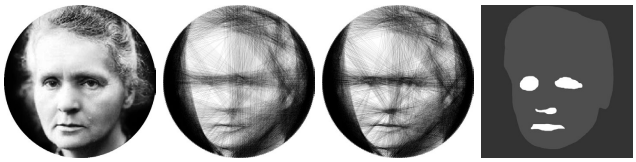
We argue that the non-zero entries of x of each iteration provide a good guess of the optimum, so we always use the corresponding columns in $\tilde{A}$ for the next iteration. Since a restriction to just the edges corresponding to x would heavily limit the variety of possible outcomes of the solver, we additionally add edges to $\tilde{A}$ up to a maximum quota. We empirically identified a decrease of this quota as a scaled and shifted e^{-x} function to allow meaningful runtimes with noticeable quality increase of the result in each iteration. The result is depicted in Figure 3h.

In order to compare it, we have also solved the same objective using our greedy optimization solver and depict the results in Figure 3i.

4.6 Importance Mapping

In order to introduce additional control over the results we use a real-valued importance map of the form

$$W\colon [0,1]^m \to [0,1]^m,$$



(a) *input* **(b)** *without W* **(c)** *with W* **(d)** *W*

Figure 5: *An example of the usage of the importance map W as defined in Eq. 8 in Section 4.7.*

which allows to enhance the importance of particular regions of the target image manually. It allows to downgrade the importance of particular pixels, for instance the background noise or other less expressive features. This map can be controlled by the user and its effects can be observed in Figure 5.

4.7 Summary

Now, we encapsulate the mapping from edges to pixels as

$$F(x) = DC(\tilde{A}x)$$

and state our final binary non-linear least squares optimization task:

$$\min_x \left\| WF(x) - Wy \right\|_2^2 \quad \text{s.t.} \quad x \in \mathbb{B}^n. \tag{8}$$

5 Optimization Algorithm

In the following we describe our greedy optimization algorithm used for solving of Eq. 6 and 8.

while *true* **do**
  $j = \arg\min_i \left\| WF(x \pm e_i) - Wy \right\|^2$
  $\tilde{f} = \left\| WF\left(x \pm e_j\right) - Wy \right\|^2$
  if $\tilde{f} < f$ **then**
    $x = x \pm e_j$
    $f = \tilde{f}$
  else
    break
  end
end

Algorithm 1: Our algorithm computes a subset of all possible strings which are used together to reassemble the input image. Please note that this code summarizes the edge addition $(x + e_i)$ and edge removal $(x - e_i)$ operations of the algorithm as described in Section 5. Note that the vector e_i refers to the i-th column of the identity matrix.

We start the process with an empty canvas and therefore initialize the output vector $x = 0$ and the residual $f = \|Wy\|_2^2$. Then we successively add edges to the canvas, one at a time, and update the vector x accordingly. To keep track of the progress, after each update of the vector x we compute the current reconstruction result and the corresponding norm according to Equation 8. In each iteration we pick the edge that allows the biggest norm reduction, and we stop when further addition would cause an increase of the error. We outline our approach in Algorithm 1.

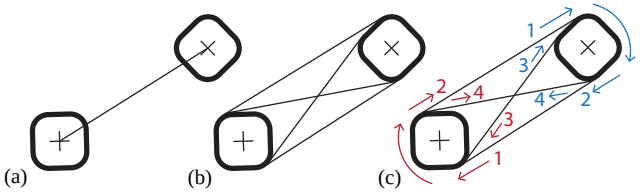


Figure 6: *Different connections of two pins: (a) pin center to pin center, (b) four different strings, and (c) enumeration of strings used by our algorithm (cf. Section 6).*

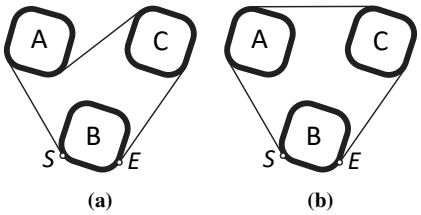


Figure 7: *An even valence at every pin is not a sufficient condition to make a result fabricable. (a) While pin A has even valence, the thread can not be easily mounted to follow the given edges. (b) The result can easily be produced by moving the thread around the pins.*

The big disadvantage of a greedy approach as we use it is that a decision at an early stage can later turn out to be a bad choice. In our case, an addition of one edge might bring the biggest benefit when it is chosen, but can then prevent a better solution later on. Thus, we try to further improve the results by an iterative removal of edges. In particular, if the initial addition stage terminates, we sequentially remove those edges that allow to improve the norm. If it is not possible anymore, we start to add edges again, pick one edge at a time, and continue as long as the norm can be improved.

We alternate those two stages until it is not possible to further improve the norm, neither by removal nor by addition, in which case the algorithm terminates.

6 Eulerian Graph Extraction

As stated in Section 3, requirement **(R4)**, the string representation of the input image needs to be fabricated from a single piece of a thread. This is only the case if the corresponding graph consisting of the computed edges is Eulerian or at least semi-Eulerian.

6.1 Eulerian Graph

In the first case, each pin would have an even valence and would therefore be connected to an even number of edges. The graph would then, by definition, contain an Eulerian cycle—the first and last pin would coincide and could be chosen arbitrarily. In the second case, there would be two dedicated pins with odd valence acting as the start and end node. These properties, however, would only work according to our expectation when we consider connections between the pin centers (cf. Figure 6a).

Since we aim at an automated fabrication in which we can attach the thread to the pins by simply moving it clockwise or counter-clockwise around the pins (and not sticking it to the pin centers), we distinguish between 4 different possibilities to connect two pins and therefore consider 4 edges for each pair of pins in our algorithm (cf. Figure 6b). One implication of this is that an even valence of the pins is not anymore a sufficient condition to make it fabricable. The problem is that not only the valence, but also the type of the connections is of importance. This issue is shown as a toy example in Figure 7.

Pin **B** acts as the start and end node of an Euler cycle, where we denote the connection points of the first and last edge at the pin **B** with *S* and *E* respectively. Although pin **A** exhibits an even number of connected edges, it is not possible to fabricate the toy example 7a because there is no simple way to mount the thread at pin **A** such that it is connected to the pins **B** and **C** in the demonstrated manner. In contrast, in the example 7b, pin **A** also has an even valence but the thread can easily be moved clockwise around **A** to produce the desired result.

6.2 Edge Enumeration

To get a better track of different connection types between two pins, we enumerate them as shown in Figure 6c. The enumeration numbers are defined w.r.t. a reference pin. In the figure, red numbers refer to the lower pin and the blue numbers to the upper pin. The following explanation refers to the lower pin. Further, we assume a clockwise movement of the string around the pin. We enumerate the inbound outer tangent with 1, the outbound outer tangent with 2, the diagonal inbound tangent with 3 and the diagonal outbound tangent with 4. Note that w.r.t. the upper pin the enumerations of the edges 1 and 2 are swapped while the numbers 3 and 4 stay unchanged. Due to their similar role as inbound edges in case of a clockwise movement, we group the edges 1 and 3 to one group (denoted as 1-3) and due to their role as outbound edges we group the edges 2 and 4 to another group (denoted as 2-4). In case of a counter-clockwise movement of the thread, only the directions but not the groups would change. The edges in group 1-3 would then act as outbound edges while the edges in group 2-4 would act as inbound edges.

Finally, we deduce the key criteria of a fabricable graph: At each pin the sum of edges of group 1-3 must equal the sum of group 2-4, thereby allowing an easy mounting of the thread by entering the pin on an edge of one group and leaving it on an edge of the other group. Compared to the traditional definition of an Eulerian graph, we consider our graph only Eulerian if all pins have an equality of the two mentioned edge groups (1-3, 2-4), which allows an arbitrary choice of the start node. The thread would then reach and leave each pin equally often and the start node also represents the end node.

A valid semi-Eulerian graph in our notion implies equality of the edge groups at all but two pins where the absolute difference between the sums equals 1. This means that one of these two pins is left once more by the thread than it is reached (start node) and one pin is reached once more by the thread than it is left (end node).

6.3 Auxiliary Edges

The first step towards a fabricable string art image is therefore an analysis of the graph given by the computed edges. Since the result will generally not meet the mentioned requirements, we add auxiliary edges to get a graph that meets the mentioned requirement of equilibrium of numbers of edges from each group at each pin. We

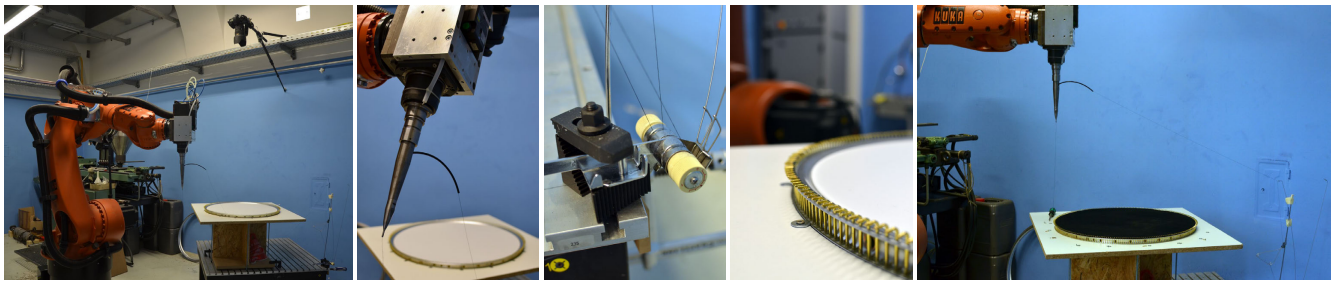


Figure 8: *The hardware setup for digital fabrication using a* KUKA *industrial robot and custom tools for winding (cf. Section 7.1).*

compute these auxiliary edges by ordering the set of pins which are not in equilibrium in descending order w.r.t. to the number of edges which are needed to reach an equilibrium stage. Then we identify pairs that need the same number of edges and connect them with suitable auxiliary edges. Note that two pins with an equal number of missing edges can always be brought into equilibrium, no matter on which side the edges are missing. This is possible through the use of all four connection types. Note that we can let two pins with offset of 1 untouched. These pins would then be the start and end of an Eulerian path.

To avoid visible artifacts, the auxiliary edges which were added are not drawn across the image but outside of the domain. The two corresponding pins of each auxiliary edge are therefore not connected by a straight line but by an arch around the frame which does not influence the image quality at all. Note that this strategy can also be managed during the fabrication with the robot by moving the tool tip on an arch outside the frame accordingly (cf. Sec. 7).

6.4 Path Identification

We use Hierholzer's algorithm [HW73] to compute a path in the graph that we augment with the auxiliary edges. The algorithm of Hierholzer finds an Eulerian cycle (or an Eulerian path in a semi-Eulerian graph) by the computation of sub-cycles. Since we require a path through the graph on which we enter each pin using an edge of one group (1-3, or 2-4) and leaving it using an edge of the other group, we have to adapt Hierholzer's algorithm to meet our requirements. In detail, we have to change the edge choice when a pin is reached. While computing the sub-cycles in the algorithm, we have to constrain that when a pin is entered on an edge of one group, we are only allowed to reach a neighbor pin using an edge of the other group. With this small adaption of the algorithm we receive a path through the graph that we can easily fabricate by a simple movement of the thread around the pins.

7 Fabrication

7.1 Hardware Setup

Industrial Robot. We used a high precision 6-axis articulated arm type industrial robot with a reach of 2033 mm and a payload of 60 kg, a KUKA R60HA, to automatically fabricate our samples. The robot was equipped with some standard tools holders, drills and end mills, a custom made winding tool, custom fixtures to hold the winding frame, a mechanism to adjust the string tension and a digital IP extension to trigger a digital camera (cf. Fig 8).

Frame Preparation. The fabrication process starts with the preparation of the frame and pins. For convenience we used aluminum bicycle rims as the base frame to hold the pins in place. The rim was mounted on a precisely defined piston on the machine table in a custom fixture, for each pin a hole was drilled at the center at an angle of about 10 degrees, the spikes (2x2x35mm) were then pressed into these holes.

Number of Pins. In an experimental setup, we have determined that a circular canvas of the diameter of $d = 630$ mm (approx. 26") the number of pins $p = 256$ is adequate and increasing this number improves the result only negligibly. This results in total in $n = 4 \times 32{,}640 = 130{,}560$ possible edges. Please refer to supplemental material for the details on the experiment.

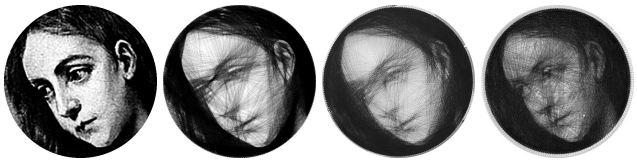
String Winding. To wind the string the drilling tool was replaced by a custom made tool mounted in a tool holder. The string exits this tool at the center point of the 2.5 mm nozzle at the center of the tool axis (therefore the main spindle did not have to be locked but was allowed to rotate freely during the fabrication process). The string tension was adjusted by running the string trough the modified thread tension regulator of a knitting machine. The thread used is thin polyester thread fabricated by Amann, type Seracor.

Fabrication Process. The large number of relatively short motions and necessary path accuracy between the pins slows down the fabrication process. To fully utilize the maximum speed available we only used linear path interpolation where absolutely necessary and point-to-point moves wherever possible. We also adjusted the allowed path deviation during different phases of the winding process and we carefully set the end effector orientation to make sure that the slow axis (A5 and A6) are not limiting the speed of the faster axis (A1, A2, A3). With these optimizations and given a prepared frame, the automated fabrication process is capable of producing a typical sample (5000 windings, 2500 m of string) in about 2 hours.

7.2 Robot Programming

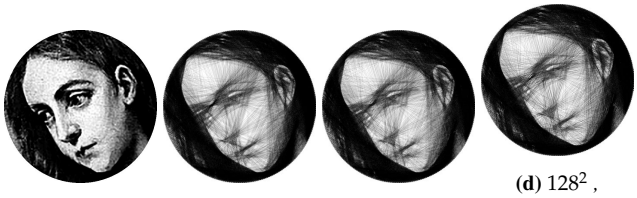
A KUKA industrial robot as we use it for the fabrication of our results provides a programming interface via the proprietary KUKA Robot Language (KRL), which contains standard commands `PTP`, `LIN`, and `CIRC` to control a point-to-point, a linear, and a circular movement of the robot tool tip respectively.

While a linear movement is guaranteed to describe a straight line, a point-to-point command moves the tool tip to the required position as fast as possible, thereby describing a (for the programmer) undefined curve. Since we could not identify any problems when using a



(a) *original* **(b)** *our result* **(c)** *our make* **(d)** *from [Vre16]*

Figure 9: *Comparison of our result generated with Eq. 8, our make, and a result taken from the web-page of Petros Vrellis.*



(a) 512^2, *original* **(b)** 512^2, *t=2.8h* *RMS*= 0.154 **(c)** 256^2, *t=9.5h* *RMS*= 0.136 **(d)** 128^2, *t=12.6h* *RMS*= 0.129

Figure 10: *Comparison of the results computed with our solver and Eq. 8 with varying input resolution.*

PTP command to move the tool tip from one pin to another, we use it throughout our source code for performance reasons. To move the tool tip around the pins, we use CIRC commands depending on the output of our algorithm in clockwise or counter-clockwise direction. We also use CIRC commands for the auxiliary edges to move the thread around the frame.

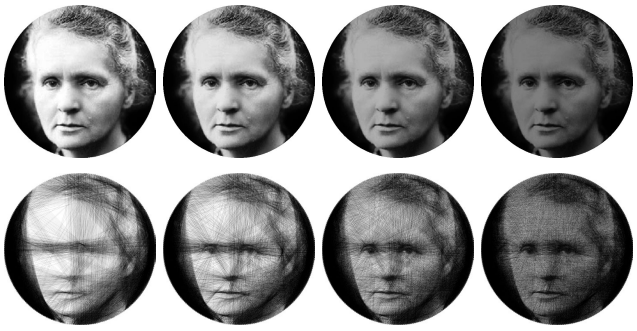
8 Implementation

Our whole pipeline is implemented in MATLAB. Due to the heavy size of the matrix $\tilde{A}$ ($4096^2 \times 130{,}560$ with $615{,}320{,}477$ non-zeros, therefore 4.58 GB in double precision), we precompute it once as a sparse matrix and store the non-empty indexes and values given by the `find` function into binary files on the hard disc. For efficiency reasons, we avoid the use of usually slow `for` loops and instead make use of large index vectors to reference the pixels of all edges at once. This gives us a good performance in the update of the vectors in which we store the forecast of the l^2 norms for the addition and removal of each edge. These vectors have to be updated after each iteration of our greedy algorithm. To further improve the performance, during each update we only refer to the pixels of each edge which are influenced. To sum up the changes for all edges at once we make use of the MATLAB function `accumarray`.

9 Results and Discussion

Outputs. Using our optimization method and Eq. 8 we have computed a number of string art images, which are shown in Figures 12 and 13. Table 1 shows the number of edges and the total running times for the computation of the particular outputs. Table 2 shows the timings of the comparisons shown in Figure 3.

Additionally, we have fabricated the results shown in Fig. 13 using our fabrication setup as described in Section 7. The fabrication time was about 2.4 hours for Ada Lovelace, 3.4 hours for Einstein, and 4.3 hours for the cat. For the Ada we used 2,679 meters of thread, for Einstein 2,980 meters, and for the cat, which is most complex, we used 5,985 meters.



(a) *original* **(b)** $c = 0.9$, *RMS*= 0.137 **(c)** $c = 0.7$, *RMS*= 0.114 **(d)** $c = 0.5$, *RMS*= 0.097,

Figure 11: *Comparison of the results computed with our solver and Eq. 8 with scaling of the input image range by the factor c. Top: target, bottom: the resulting output.*

Comparisons. For our benchmarks and comparisons we have used a cutout of the image of the picture *Penitent Magdalene* by El Greco. This motive has also been used by the artist Petros Verlis [Vre16] for his outputs, and we show a comparison in Figure 9.

Moreover, we compared the results of our solver to the GUROBI multiresolution solver using the same objective function (Eq. 7). Figures 3i and 3h show the visual outcomes respectively. Interestingly, regarding the running times (cf. Table 2), it turns out that our solver outperforms the multiresolution GUROBI solver by an order of magnitude and provides even better results in terms of both the minimized L^1 norm and the l^2 norm.

Input Resolution. In Figure 10 we compare the results of our algorithm with varying resolution of the input image. Since we work with a constant output resolution in order to suffice fabrication requirement **(R2)**, reducing the input resolution results in increasing the subsampling window, which again results in increasing of the range of each pixel that is evaluated. In practice, as it can be envisioned in Figure 10, reduction of the input resolution does not influence the quality of the results considerably, however, it leads to longer computation times. The rationale is that a bigger range leads to a bigger design space per pixel, or in other words to more possibilities, which boils down in longer execution times of the solver, which tries to explore this space.

In practice we have determined empirically that a input resolution of 512^2 pixels, which results in 8^2 subsampling windows and thus in a range of 64 gray values, delivers best results in most cases.

Image Preprocessing. An additional question is about an appropriate preprocessing of the input images. From the experiments we have performed we concluded that results are better if inputs do not exhibit too big variations in the range and if the images are in general darker. Thus, an essential issue of the string imagery is the actual range of the images. In Section 4.4 we introduce a method to extend the range during optimization. On the other side, we could also compress the range and lower the intensity of the input image in order to shift the optimization problem into a more appropriate design space region.

We implemented it in a rather straightforward manner just by scaling the input image down by a constant factor $c \in (0,1)$, i.e., $\bar{y} = cy$

name	*W*	*n*	*t* **[s]**
Ada Lovelace	yes	6575	6998.289
Alan Turing	no	20559	14922.560
Albert Einstein	no	13026	9432.940
Albert Einstein (with tongue)	no	8324	6721.494
Cat	no	14850	15812.312
Gaussian Blob	no	507	2178.496
Magdalene	no	10493	9992.371
Marie Curie	yes	6785	6354.935

Table 1: *Timings of some results presented in Figures 12 and 13.*

name	RMS	*t* **[s]**
Fig. 3(b) Eq. 1	0.056	3.82
Fig. 3(d) Eq. 3	0.169	92.81
Fig. 3(f) Eq. 4	0.463	4,084.88
Fig. 3(g) Eq. 5	0.279	51,271.36
Fig. 3(h) Eq. 7 (GUROBI)	0.232	7765.43
Fig. 3(i) Eq. 7 (ours)	0.221	539.47
Fig. 3(j) Eq. 8 (ours)	0.154	9,992.37

Table 2: *Timings and errors of the samples shown in Figure 3.*

and could obtain better results for images with large variations in the range, as shown in Figure 11.

Discussion. From the experiments we have performed and documented in Section 4, we extract the following insights:

First of all, we need enough range per pixel in order to express the grayscale nuances which make the string images look much more realistic. Thus, we trade resolution for range by utilizing supersampling and downfiltering. In fact, the number of superpixels gives us approximately the number of achievable graylevels.

Second, in general, we do not need a very high resolution input, since the degree of detail that can be approximated with a string image is limited. Using 512^2 sized input

Third, the input should not exhibit too large range variations and especially very light regions. Darker and smoother images can be better approximated by string images.

Finally, we need to say that non-binary least squares do not suffer the range problem since the minimizer can take real values and account for enough grayscale range by adding fractional parts of the edges to particular pixels. This is, however, not possible to fabricate with our setup.

Limitations. A major limitation comes from the current setup with pins only placed on the boundary of a circular frame. This limits the space of images that can be faithfully represented. Placing pins in the interior of the artwork or jointly optimizing for pins and strings should provide additional degrees of freedom and improve the quality of the output. Our current computation should be able to adapt to different fixed pin placement setups without major changes, but the joint optimization of pin placement and string selection would require new algorithmic components.

One final limitation to mention is the fact that our solver does not necessarily converges to a global minimum. In fact, it can still get stuck in a local minimum; our efforts were however, to find a minimum which still results in a string image of good visual quality.

10 Conclusions

In this paper, we proposed a novel method for the automatic computation and digital fabrication of artistic string images. We analyze the problem and derive a problem formulation as a non-linear binary least squares problem. We propose a hardware setup for the automatic digital fabrication of these images using an industrial robot that spans strings between pins. We also propose a greedy algorithm to compute an approximate solution to the optimization problem and demonstrate that the quality of our solution significantly outperforms other approaches. Finally, we demonstrate the applicability of our methods by generating and fabricating a set of real string art results.

In future work, we would like to extend our setup to experiment with more general pin placement, to generate 2.5d string art. Further, we would like to experiment with strings of different colors and more transparent strings. We are also interested to investigate the fabrication of customized shading systems using a similar process.

11 Acknowledgments

We thank the reviewers for their helpful suggestions. This research was funded by the Austrian Science Fund (FWF P27972-N31) and the Vienna Science and Technology Fund (WWTF ICT15-082).

References

[AM10] ALEXA M., MATUSIK W.: Reliefs as images. *ACM Transactions on Graphics 29*, 4 (jul 2010), 1. 2

[AM11] ALEXA M., MATUSIK W.: Images from self-occlusion. In *Proceedings of the International Symposium on Computational Aesthetics in Graphics, Visualization, and Imaging - CAe '11* (New York, New York, USA, 2011), ACM Press, p. 17. 2

[BBAM12] BERMANO A., BARAN I., ALEXA M., MATUSK W.: ShadowPix: Multiple Images from Self Shadowing. *Computer Graphics Forum 31*, 2pt3 (may 2012), 593–602. 2

[BKB*12] BARAN I., KELLER P., BRADLEY D., COROS S., JAROSZ W., NOWROUZEZAHRAI D., GROSS M.: Manufacturing Layered Attenuators for Multiple Prescribed Shadow Images. *Computer Graphics Forum 31*, 2pt3 (may 2012), 603–610. 2

[Bre65] BRESENHAM J. E.: Algorithm for computer control of a digital plotter. *IBM Systems Journal 4*, 1 (1965), 25–30. 3

[CCLM13] CHU H.-K., CHANG C.-S., LEE R.-R., MITRA N. J.: Halftone QR codes. *ACM Transactions on Graphics 32*, 6 (nov 2013), 1–8. 2

[CZ07] CHANG X.-W., ZHOU T.: MILES: MATLAB package for solving Mixed Integer LEast Squares problems. *GPS Solutions 11*, 4 (nov 2007), 289–294. 5

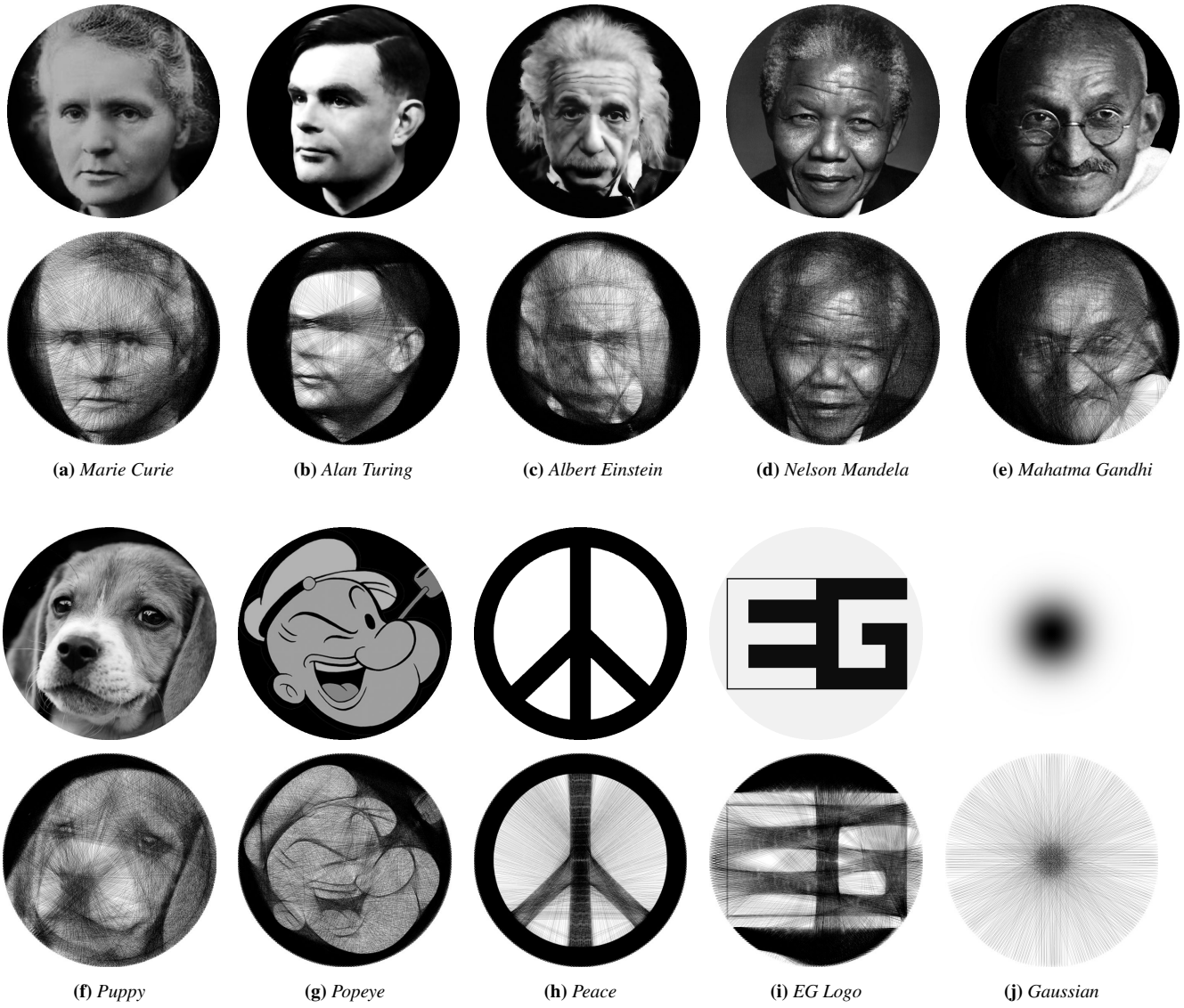
[Elb10] ELBER G.: Ortho-Pictures: 3D Objects from Independent 2D Data Sets. In *Advances in Architectural Geometry 2010*. Springer Vienna, Vienna, 2010, pp. 175–192. 2

[GKAK16] GALEA B., KIA E., AIRD N., KRY P. G.: Stippling with aerial robots. In *Proceedings of the Joint Symposium on Computational Aesthetics and Sketch Based Interfaces and Modeling and Non-Photorealistic Animation and Rendering* (2016), Eurographics Association, pp. 125–134. 2

[Gur16] GUROBI OPTIMIZATION I.: Gurobi Optimizer Reference Manual. http://www.gurobi.com, 2016. Accessed: 2018-02-06. 5, 6

[Hae90] HAEBERLI P.: Paint by numbers: abstract image representations. *ACM SIGGRAPH Computer Graphics 24*, 4 (sep 1990), 207–214. 2

[HJO*01] HERTZMANN A., JACOBS C. E., OLIVER N., CURLESS B.,



(a) *Marie Curie* **(b)** *Alan Turing* **(c)** *Albert Einstein* **(d)** *Nelson Mandela* **(e)** *Mahatma Gandhi*

(f) *Puppy* **(g)** *Popeye* **(h)** *Peace* **(i)** *EG Logo* **(j)** *Gaussian*

Figure 12: *Results we have computed with our automated pipeline. Top: the input image, bottom: the results of our optimization.*

SALESIN D. H.: Image analogies. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques - SIGGRAPH '01* (New York, New York, USA, 2001), ACM Press, pp. 327–340. 2

[HW73] HIERHOLZER C., WIENER C.: Ueber die Möglichkeit, einen Linienzug ohne Wiederholung und ohne Unterbrechung zu umfahren. *Mathematische Annalen 6*, 1 (mar 1873), 30–32. 8

[KL11] KOPF J., LISCHINSKI D.: Depixelizing pixel art. *ACM Transactions on Graphics 30*, 4 (jul 2011), 1. 2

[Laa18] LAARCO: Laarco Studio | String Art, Algorithmic Art, Customized Portraits, 2018. 1

[LGX*13] LEVIN A., GLASNER D., XIONG Y., DURAND F., FREEMAN W., MATUSIK W., ZICKLER T.: Fabricating BRDFs at high spatial resolution using wave optics. *ACM Transactions on Graphics 32*, 4 (jul 2013), 1. 2

[LPD13] LINDEMEIER T., PIRK S., DEUSSEN O.: Image stylization with a painting machine using semantic hints. *Computers & Graphics 37*, 5 (aug 2013), 293–301. 2

[LSS*14] LAU C., SCHWARTZBURG Y., SHAJI A., SADEGHIPOOR Z., SÜSSTRUNK S.: Creating personalized jigsaw puzzles. In *Proceedings of the Workshop on Non-Photorealistic Animation and Rendering - NPAR '14* (New York, New York, USA, 2014), ACM Press, pp. 31–39. 2

[MP09] MITRA N. J., PAULY M.: Shadow art. *ACM Transactions on Graphics 28*, 5 (dec 2009), 1. 2

[PDP*15] PANOZZO D., DIAMANTI O., PARIS S., TARINI M., SORKINE E., SORKINE-HORNUNG O.: Texture Mapping Real-World Objects with Hydrographics. *Computer Graphics Forum 34*, 5 (aug 2015), 65–75. 2

[PJJ*11] PAPAS M., JAROSZ W., JAKOB W., RUSINKIEWICZ S., MATUSIK W., WEYRICH T.: Goal-based Caustics. *Computer Graphics Forum 30*, 2 (apr 2011), 503–511. 2

[PJJSH16] PRÉVOST R., JACOBSON A., JAROSZ W., SORKINE-HORNUNG O.: Large-scale painting of photographs by interactive optimization. *Computers & Graphics 55*, C (apr 2016), 108–117. 2

[PS82] PAIGE C. C., SAUNDERS M. A.: LSQR: An Algorithm for

(a) *Albert Einstein* **(b)** *Ada Lovelace* **(c)** *Black Cat*

Figure 13: *Three results we have fabricated with our automated pipeline. Top: the input image, bottom: photography of the physically fabricated image.*

Sparse Linear Equations and Sparse Least Squares. *ACM Transactions on Mathematical Software 8*, 1 (mar 1982), 43–71. 3

[QS08] QIAO S., SANZHENG: Integer least squares. In *Proceedings of the 2008 C3S2E conference on - C3S2E '08* (New York, New York, USA, 2008), ACM Press, p. 23. 5

[RMP11] REICHINGER A., MAIERHOFER S., PURGATHOFER W.: High-quality tactile paintings. *Journal on Computing and Cultural Heritage 4*, 2 (nov 2011), 1–13. 2

[SABS94] SALISBURY M. P., ANDERSON S. E., BARZEL R., SALESIN D. H.: Interactive pen-and-ink illustration. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques - SIGGRAPH '94* (New York, New York, USA, 1994), ACM Press, pp. 101–108. 2

[SMPZ15] SHILKROT R., MAES P., PARADISO J. A., ZORAN A.: Augmented Airbrush for Computer Aided Painting (CAP). *ACM Transactions on Graphics 34*, 2 (mar 2015), 1–11. 2

[SPG*16] SCHÜLLER C., PANOZZO D., GRUNDHÖFER A., ZIMMER H., SORKINE E., SORKINE-HORNUNG O.: Computational thermoforming. *ACM Transactions on Graphics 35*, 4 (jul 2016), 1–9. 2

[SPSH14] SCHÜLLER C., PANOZZO D., SORKINE-HORNUNG O.: Appearance-mimicking surfaces. *ACM Transactions on Graphics 33*, 6 (nov 2014), 1–10. 2

[STTP14] SCHWARTZBURG Y., TESTUZ R., TAGLIASACCHI A., PAULY M.: High-contrast computational caustic design. *ACM Transactions on Graphics 33*, 4 (jul 2014), 1–11. 2

[SY17] SEO S. H., YOUNG J. E.: Picassnake. In *Proceedings of the Companion of the 2017 ACM/IEEE International Conference on Human-Robot Interaction - HRI '17* (New York, New York, USA, 2017), ACM Press, pp. 418–418. 2

[Var17] VARGA D.: String art. https://github.com/danielvarga/string-art, 2017. Accessed: 2018-02-06. 1, 3, 4

[Vre16] VRELLIS P.: A new way to knit. http://artof01.com/vrellis/works/knit.html, 2016. Accessed: 2018-02-06. 1, 9

[WDB*07] WEYRICH T., DENG J., BARNES C., RUSINKIEWICZ S., FINKELSTEIN A.: Digital bas-relief from 3D scenes. *ACM Transactions on Graphics 26*, 3 (jul 2007), 32. 2

[WPMR09] WEYRICH T., PEERS P., MATUSIK W., RUSINKIEWICZ S.: Fabricating microgeometry for custom surface reflectance. *ACM Transactions on Graphics 28*, 3 (jul 2009), 1. 2

[WXWX91] WU X., XIAOLIN, WU, XIAOLIN: An efficient antialiasing technique. In *Proceedings of the 18th annual conference on Computer graphics and interactive techniques - SIGGRAPH '91* (New York, New York, USA, 1991), vol. 25, ACM Press, pp. 143–152. 3

[YIC*12] YUE Y., IWASAKI K., CHEN B.-Y., DOBASHI Y., NISHITA T.: Pixel Art with Refracted Light by Rearrangeable Sticks. *Computer Graphics Forum 31*, 2pt3 (may 2012), 575–582. 2

[ZLW*16] ZHAO H., LU L., WEI Y., LISCHINSKI D., SHARF A., COHEN-OR D., CHEN B.: Printed Perforated Lampshades for Continuous Projective Images. *ACM Transactions on Graphics 35*, 5 (jun 2016), 1–11. 2

[ZYZZ15] ZHANG Y., YIN C., ZHENG C., ZHOU K.: Computational hydrographic printing. *ACM Transactions on Graphics 34*, 4 (jul 2015), 131:1–131:11. 2