Received: 25 April 2019 | Accepted: 29 April 2019

DOI: 10.1002/cav.1904

SPECIAL ISSUE PAPER

WILEY

A computational approach for spider web-inspired fabrication of string art

Seungwoo Je[1] | **Yekaterina Abileva**[2] | **Andrea Bianchi**[1] | **Jean-Charles Bazin**[3,4]

[1]Department of Industrial Design, KAIST (Korea Advanced Institute of Science and Technology), Daejeon, South Korea

[2]School of Computing, KAIST (Korea Advanced Institute of Science and Technology), Daejeon, South Korea

[3]Graduate School of Culture Technology, KAIST (Korea Advanced Institute of Science and Technology), Daejeon, South Korea

[4]School of Electrical Engineering, KAIST (Korea Advanced Institute of Science and Technology), Daejeon, South Korea

Correspondence
Jean-Charles Bazin, Graduate School of Culture Technology, KAIST (Korea Advanced Institute of Science and Technology), Daejeon 34141, South Korea; or School of Electrical Engineering, KAIST (Korea Advanced Institute of Science and Technology), Daejeon 34141, South Korea.
Email: bazinjc@kaist.ac.kr

Funding information
KAIST Interdisciplinary Research on the 4th Industrial Revolution; Adobe; National Research Foundation of Korea (NRF) funded by the Korean government (Ministry of Science and ICT), Grant/Award Number: 2018R1A5A7025409

Abstract

Creating objects with threads is commonly referred to as string art. It is typically a manual, tedious work reserved for skilled artists. In this paper, we investigate how to automatically fabricate string art pieces from one single continuous thread in such a way that it looks like an input image. The proposed system consists of a thread connection optimization algorithm and a custom-made fabrication machine. It allows casual users to create their own personalized string art pieces in a fully automatic manner. Quantitative and qualitative evaluations demonstrated our system can create visually appealing results.

KEYWORDS

fabrication, image halftoning, string art

1 | INTRODUCTION

Some species of spiders can build intriguing and sophisticated web structures using just a single continuous thread. For example, Miagrammopes spiders in the family of Uloboridae can build webs consisting of as few as one sticky thread.[1] Inspired by this spider's ability, we investigate how to automatically fabricate planar objects from one single continuous thread (no cut, no glue) in such a way that it looks like an input image. For example, given a photo of a face, the goal is to create a web-like object made of a single thread and resembling the input face (see Figure 1). Creating objects with threads is commonly referred as *string art*. Creating string art pieces is typically a manual, tedious work. While casual people can manually design and fabricate simple basic shapes, complex patterns can only be crafted by skilled artists. Designing the thread pattern is a particularly challenging task because it requires to decide how to connect the thread, that is, connect which pin to which pin and which order. Moreover, the manual fabrication itself can take up to numerous hours or even days depending on the pattern complexity and the number of connections.

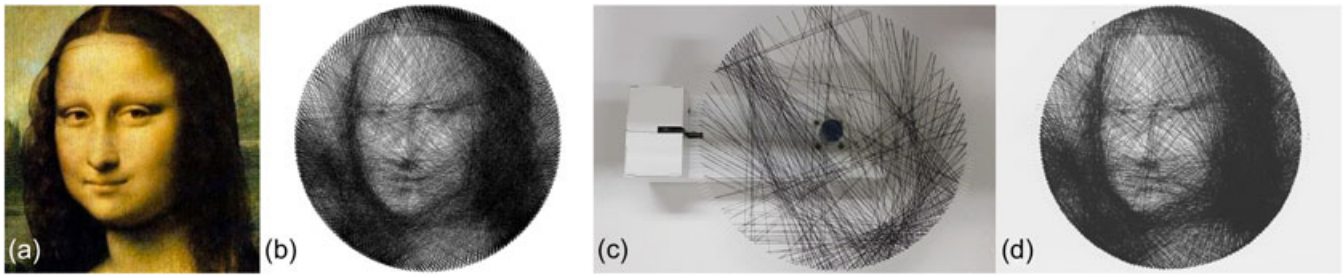

FIGURE 1 (a) Input photo. (b) Simulation result obtained by the proposed thread connection optimization. (c) Our custom-made fabrication machine to connect the thread according to this optimized set connections. (d) Photo of the fabricated object. Both the thread connection optimization and the actual fabrication run in a fully automatic manner. The fabricated object has a diameter of 36 cm and is made of one continuous thread (no cut, no glue), and the total thread length is around 200 meters

In this paper, we focus on planar thread object fabrication, just like spider webs. The fabrication is performed by arranging a single thread between pins located on the boundary of a planar support, such as a circle. The input is an image provided by the user, such as a photo, drawing, or painting (Figure 1a). We then run the proposed thread connection optimization algorithm to find in which order the thread must be connected to the pins so that the simulation result of the optimized thread connection looks like the input image (see the rendered image in Figure 1b). We then fabricate the thread object by physically connecting the thread to the pins using our custom-made machine (see Figure 1c). The output is the fabricated version of the input image (see the photo of the fabricated object in Figure 1d).

Overall, our two main contributions are as follows.

- The full description of an automatic fabrication system using one single continuous thread (no cut, no glue). We describe both the software (e.g., thread optimization) and hardware (e.g., our custom-made fabrication machine) components in detail.
- We introduce and evaluate different thread optimization methods, including physical constraint-aware algorithms to deal with the mechanical constraints of the machine.

Quantitative and qualitative evaluations will show that our system can create visually appealing results. In summary, our system allows casual users for creating their personalized recreational string art from their own photos in a fully automatic manner.

2 | RELATED WORK

Fabrication In computer graphics and related fields, several exciting works have recently investigated unconventional, inspiring computational fabrication of objects and art creation, such as knitting,[2] thermoforming,[3] stippled prints with drone,[4] perforated lampshades,[5] spray painting,[6] airbrush system,[7] balloons,[8] inflatable structures,[9] tile decors,[10] and kites.[11] Our work belongs to this line of works, and we present a fully automatic system to fabricate spider web-inspired string art pieces.

In the context of string art, the artist Petros Vrellis* designed an algorithm to compute the pin connection. However, his exact approach is not described, and the pins are connected by hand, which might take numerous hours or even days. In contrast, we provide a full description of our thread optimization algorithm and present a custom-made fabrication machine that can automatically connect the thread. Recently and independently of our work, Birsak et al.[12] have presented an automatic approach for the thread optimization and fabrication. However, their optimization is expressed as a nonlinear binary least squares problem, which requires around 2–4 hr of computation. In contrast, our approach takes just a few minutes. Moreover, they use an expensive, large-size industrial robot (more than 20,000 USD), whereas our proposed DIY custom-made machine costs only around 120 USD. In addition, we introduce several extensions and applications, such as multicolor string art pieces, multiview 3D objects, and stencils.

Computational art creation and image generation Computational methods have also been proposed for the design and generation of recreational arts and images in various contexts such as pixel art,[13] emerging images,[14]

*http://artof01.com/vrellis/

camouflage images,[15] shadow art,[16] collage art,[17] text art,[18] spray painting,[6] and airbrush,[7] among many others. The main difference is that our work considers a single continuous thread, which requires a dedicated optimization algorithm and a fabrication machine.

Dithering and halftoning Generation of images with specific patterns has been studied in the contexts of dithering, halftoning, or stippling. For example, halftoning is the process of generating a pattern of binary pixels that creates the illusion of a continuous-tone image.[19] Closely related is stippling where dots are drawn, both digitally[20,21] or physically,[4] in such a way that it looks like a target photo. Another interesting work is the generation of a QR code resembling an input image.[22] These works consider a discrete set of dots or pixels that can be individually optimized and drawn. In contrast, we consider one single continuous physical thread, which requires dedicated software and hardware components.

3 | PROPOSED APPROACH

In our setup, we consider a set of K pins (typically 200) uniformly distributed on the boundary of a circular support, and one continuous thread of a single color. An example is shown in Figure 1c. In the following, we present our hardware and software components.

3.1 | Hardware

SpiderPrinter Our fabrication machine, called *SpiderPrinter*, is a custom-made turn-table machine that dispenses a single thread around a circular transparent acrylic plate (Figure 2a). The plate contains pins protruding from its circumference, with every two adjacent pins forming a V cut (Figure 2b). SpiderPrinter operates through the synchronized motions of two motors: first, a servomotor guides the thread to the top or bottom side of the plate and, second, a stepper motor rotates the plate, effectively pulling the thread from one point of the circumference to another, forming a line.

Our proposed fabrication is composed of a sequence of thread lines placed alternatively on the top or bottom side of the plate. Because the plate is transparent, the lines on both layers equally contribute to the final visual result. Compared for example to winding the thread around a pin,[12] the main advantage of our two-layer approach is its simplicity and efficiency. For example, winding the thread around a pin would require more complex motion paths for the motors, more complex mechanisms (higher degrees of freedom), more expensive components, and more challenging controls.[12] In contrast, our approach simply operates a sequence of in-plane rotations and up/down motions. An additional advantage compared to winding is that it ensures that two consecutive lines always share exactly the same point on the circumference (i.e, the bottom of the V cut), hence providing a "sharper" look of the object. In our design, the plate is directly attached to the stepper motor's shaft with an adapter (Figure 2a), so that a thread on the bottom layer cannot possibly pass through the plate's center. We will show that this physical constraint can simply be enforced in the thread connection optimization.

Hardware specifications SpiderPrinter consists of a 280×110×130 mm acrylic box containing several electromechanical components (see Figure 3): a stepper motor, a microstepper driver, an analog servomotor, two cooling fans, a thread dispenser, and an Arduino UNO controlling board connected through USB to a PC. The acrylic plate is screwed on a

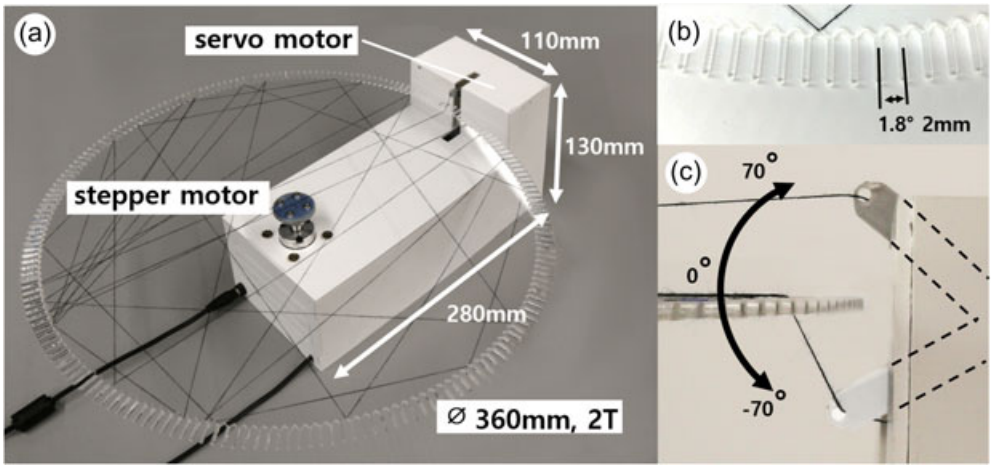

FIGURE 2 (a) Overview of our SpiderPrinter. (b) Top view of the acrylic plate's pins. (c) Side view of the servo-arm motion from the top to the bottom layer of the acrylic plate

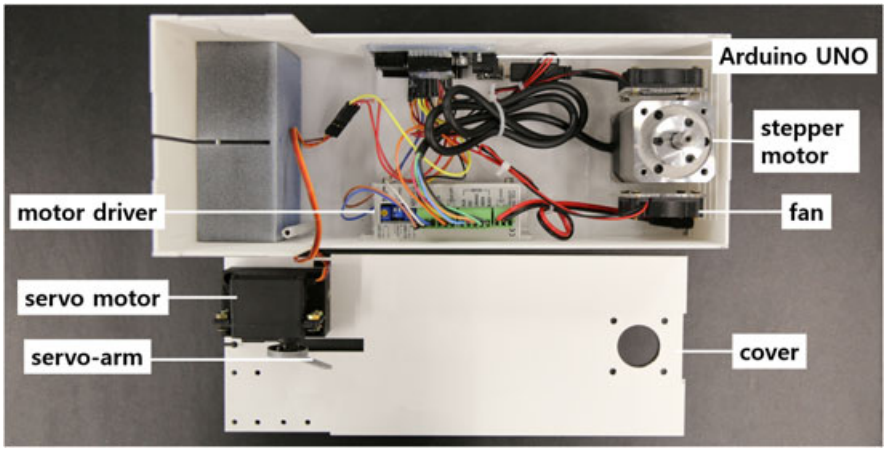

FIGURE 3 Inside view of our SpiderPrinter

3D-printed 30-mm radius adapter made of PolyLactic Acid (PLA), which is fixed on the top of the shaft of a five-phase stepping motor (A15K-S545-G10 with 0.75A/Phase) capable of 15 kgf-cm torque. The microstepper controller (MD5-ND14) drives the stepper motor with a 0.072° accuracy per step. Two small fans (SZH-GNP511, 5V at 0.14A, 6,000 RPM) are attached next to the base of the stepper motor to provide cooling and to ensure an operational temperature between 30°C and 40°C. A 25-mm-long PLA arm is attached to the shaft of a servomotor (FS5103B, 4.8V, 3 kg·cm) and is used to guide the thread through the V cuts of the acrylic plate. The thread spool is placed on the back of the device, and two small bearings ensure its fluid motion.

SpiderPrinter uses a circular 2T acrylic plate with a diameter of 36 cm that contains $K = 200$ equidistant pins protruding from its circumference (length: 12 mm, thickness: 1.5 mm), with every two adjacent pins forming a V cut (Figure 2b). The distance between two pins is 2 mm (1.8°), which is large enough to comfortably allow the passage of the 3D-printed arm (thickness: 1 mm) attached to the servomotor or the thread. The thread we used is a spun polyester thread 60S/3 (approximately 0.14 mm thick).

3.2 | Software

In this section, we will present our thread connection optimization. Before going into the details of the algorithm, let us first clarify the terminology. In the following, we use the term pins as a general term to refer to the elements where the thread is attached, in our case, the bottom of the V cuts in the plate (Figure 2b). We call the line a part of the thread connected between two pins (see Figure 4d). We interchangeably use the terms thread combination, arrangement, and connection. The user-provided image is called input image or target image.

We now present how to compute the thread arrangement, that is, connect the thread from which pin to which pin and in which order. We will introduce two different strategies for the thread optimization and compare them. We consider a black thread and a white background. Therefore, the absence of thread in a part of the image is seen as white. An important and interesting observation is that the perceived color tone depends on the arrangement of the threads, that is, if many threads pass trough a certain part, then this part will be perceived as darker. This means that, even if the thread has a single black color, humans can perceive variations of gray depending on the thread arrangement. Therefore, by optimizing the thread arrangement, it is possible to control the perceived color tone.

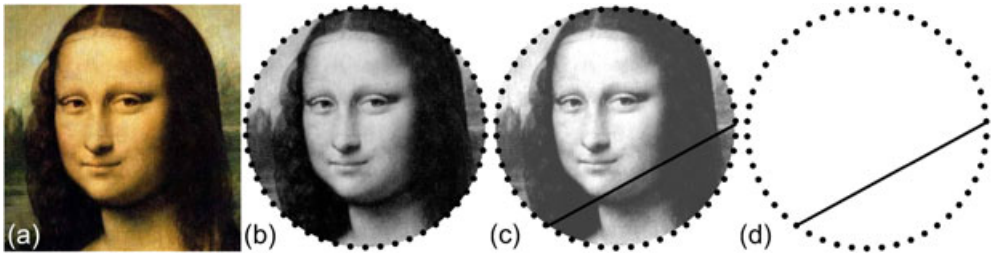

FIGURE 4 Given an input image (a), we convert it to a canonical virtual circular layout (b). A selected line on this virtual circle (c) is used to render the simulation result and corresponds to one line on the physical circle (d)

3.2.1 | Preprocessing

In a preprocessing step, we convert the input image to a canonical form with a virtual circular layout (see Figure 4). This virtual circular layout represents the physical circular support and contains the same number K of virtual pins as on the physical circle. First, we convert the input color image to grayscale, crop it to a square, and resize it to $2,000 \times 2,000$ pixels. Then, we set the virtual circle at the center of the image. Now, the goal is to optimize the thread arrangement.

3.2.2 | General algorithm

We introduce our general bottom-up optimization algorithm that iteratively adds lines (i.e., thread connections) one by one in such a way that the "resemblance" to the target image increases (the resemblance will be formally defined later). It is general in the sense that it is not tuned to a specific resemblance cost definition, and thus, different definitions can be used. It is bottom-up because we start with an empty result (i.e., without any thread) and iteratively add thread connections. While the proposed algorithm takes local decisions, experiments will show that it can provide visually satisfying results in a short amount of time (in a couple of minutes, depending on the number of lines).

The proposed thread optimization algorithm is shown in Algorithm 1. The input is the user-provided target image I_t. We first set the maximal number N of lines and the number K of pins. Typical values are $N = 3,000$ lines and $K = 200$ pins, and all the results shown in this paper have been created with these values. The pins are uniformly distributed on the virtual circle boundary and can be represented by an index from 1 to K. The output is the list of connections $\mathcal{L}$ containing the optimized pin-to-pin connections. For initialization (Line 1), $\mathcal{L}$ is set to the pin of Index 1, without lack of generality.

At each iteration (the outer for loop at Line 4), we search for the best line locally, that is, the ith iteration optimizes the ith line of the object. At each of these iterations, we test the thread connections between the previous pin PrevPin to all the K pins (the inner for loop at Line 8), evaluate their cost (Line 10), and finally select the pin leading to the best cost (Line 11). The selected best next pin is then added to the list of pin connections (Line 21). The cost is obtained at Line 10 and can be computed in different ways, as will be discussed in Section 3.2.3. The function `CheckAllowedConnection()` at Line 9 tests if a pin connection is allowed. For example, if a connection between two pins is already in the list of selected lines, then this connection is not allowed. More examples of use cases will be provided below. Finally, the procedure stops when a stopping criterion is reached. Algorithm 1 in general stops when the maximum number of lines has been reached or when no allowed combinations are found.

Algorithm 1 Thread optimization

```
1:  Input: target image I_t, maximum number of lines N, number of pins K
2:  Initialization: the combination list L = [1], i.e. the first pin (without lack of generality)
3:  //Find the next best pin iteratively
4:  for i=1:N do
5:      BestCost=∞, BestNextPin=∅
6:      PrevPin=L(i), i.e. the latest added pin
7:      //Test all the pin candidates
8:      for NextPin=1:K do
9:          if CheckAllowedConnection(PrevPin,NextPin)==True then
10:             cost=ComputeCost(L,PrevPin,NextPin)
11:             if cost<BestCost then
12:                 BestCost=cost
13:                 BestNextPin=NextPin
14:             end if
15:         end if
16:     end for
17:     if BestNextPin==∅ //i.e. no combination allowed then
18:         stop
19:     end if
20:     //Update
21:     L = [L , BestNextPin] //add the best pin to the combination list
22: end for
23: Return: L (i.e., the computed combination list)
```

3.2.3 | Cost

The role of the function `ComputeCost()` (see Line 10 of Algorithm 1) is to compute the resemblance between the target (input) image I_t and the resulting rendered image I_r composed of the set of optimized lines $\mathcal{L}$. In the following, we present two kinds of cost we experimented with local scale and global scale.

Global scale Given a set of lines $\mathcal{L}$, we render an image composed of these lines. Considering a black thread and a white background, we draw the lines by line rasterization[23] in black over a white background image. We obtain the rendered image I_r. We define the global scale cost as the tone cost between the rendered image I_r and the target (input) image I_t, where the tone cost is computed as the pixelwise mean squared error (MSE) between the Gaussian-blurred versions of the two images:

$$C(\mathrm{I}_r, \mathrm{I}_t) = \frac{\sum_{x=1}^{W} \sum_{y=1}^{H} \left(g(\mathrm{I}_r)_{x,y} - g(\mathrm{I}_t)_{x,y} \right)^2}{W \times H}, \tag{1}$$

where $g(\mathrm{I})$ is the Gaussian-blurred version of the image I, $g(\mathrm{I})_{x,y}$ is the pixel value of the blurred image $g(\mathrm{I})$ at the position (x, y), and H, W refer to the height and width of the image. In our implementation, we consider a grayscale version of the images with values scaled between 0 and 1, and employ a Gaussian kernel of size 15×15. The range of the tone cost is $[0, 1]$, where a lower value indicates higher tone similarity. We use the difference of Gaussian-filtered images instead of the difference of images to model the human visual perception along distance.[22,24–26] We call this cost the global scale cost because it considers information of the whole image; that is, when we test a pin connection, we render the whole image composed of all the lines selected so far plus the currently tested pin, and we compare the whole rendered image to the whole input image by the tone cost of Equation (1). We select the pin providing the lowest cost; then, we add it to the list of lines, and we go for the next iteration/line.

In practice, while some implementation tricks can speed up the execution,[†] the global scale defined at Equation (1) is computationally expensive. Indeed, at each iteration of the for loop at Line 8 (pin evaluation), it requires to create the rendered image I_r (i.e., draw all the lines selected so far, plus the current pin), apply a Gaussian filter on the rendered image I_r, and compute the MSE of the tone cost over the whole image. Overall, it is a computationally expensive process, around 8 s per line for a typical number of $K = 200$ pins, which is around 7 hr in total for typically 3,000 line connections. In the following, we will propose an alternative cost that runs around 70 times faster (around 6 min), with similar visual quality.

Local scale For the local scale cost, we use local information of the image. Contrary to the global scale approach, we start by rendering I_r using only the lines $\mathcal{L}$ selected so far, that is, without the current line (`PrevPin`,`NextPin`), and its Gaussian-filtered version $g(\mathrm{I}_r)$. Therefore, this can be done *outside* the for loop of the pin evaluation (Line 8) and, thus, is faster. Next, instead of computing the total MSE cost over the whole image (Equation 1), we compute the local signed pixelwise tone difference at the pixel (x, y):

$$D(\mathrm{I}_r, \mathrm{I}_t)_{x,y} = g(\mathrm{I}_r)_{x,y} - g(\mathrm{I}_t)_{x,y}. \tag{2}$$

The range of each signed tone difference is $[-1, 1]$. A positive value means that $g(\mathrm{I}_r)_{x,y} > g(\mathrm{I}_t)_{x,y}$, that is, $g(\mathrm{I}_r)_{x,y}$ is brighter than $g(\mathrm{I}_t)_{x,y}$, and so should be darker to resemble the target image more. Reciprocally, a negative value means that $g(\mathrm{I}_r)_{x,y} < g(\mathrm{I}_t)_{x,y}$, that is, $g(\mathrm{I}_r)_{x,y}$ is darker than $g(\mathrm{I}_t)_{x,y}$, and so should be brighter to resemble the target image more. Because our method can only add lines, we can only make an image location darker, not brighter.

Based on the above explanation, our aim is to find a line that will make the part darker. For this, we test all the pin combinations at Line 8 and select the one whose brightness is too bright compared to the target image so that we can make it darker. Concretely, to compute the brightness of a line, we obtain the coordinates of the pixels along this line by line rasterization[23] and compute the cost based on the values of these pixels by Equation (2). We compute the line cost as the average values (divided by the number of rasterized line pixels) to not indirectly favoring shorter segments. Finally we select the line with the highest cost, that is, the most "too bright" line.

[†]In practice, the image I_r does not have to be rendered from scratch when testing a new combination, that is, at each iteration of the for loop at Line 8. Indeed, it is possible to just draw the current line (`PrevPin`, `NextPin`) on top of the image composed of the lines $\mathcal{L}$ obtained so far at the previous iteration i (Line 4). Alternative ways are also possible, such as "undrawing" the previously tested line and drawing the current line, which avoids creating a duplicate of the image.

In terms of implementation, as discussed above, the rendered image I_r and its Gaussian-filtered version $g(I_r)$ are composed of the lines obtained so far, at the previous iteration. Therefore, the tone difference at Equation (2) can be computed once per iteration i, that is, outside the for loop of the pin tests (Line 8) and, thus, runs much faster than the global scale computation. The cost of a line is thus reduced to a sum of values along the line, which is computationally cheap. We call this cost the local scale cost because we use the local information along the line.

3.2.4 | Physical constraints-aware optimization

The thread optimization described so far allows any pins to be virtually connected to any pins. In practice, our fabrication machine has physical constraints that we need to take into account during the thread optimization.

First, as discussed in Section 3.1, our machine connects the thread by alternating top and bottom connections, that is, the bottom layer contains the even connections and the top layer contains the odd connections. Because our plate is transparent, the bottom and top layers are seamlessly combined and equally contribute to the final perceived result. An important practical aspect is that the threads on the bottom layer cannot pass through the plate's center because the plate's center is attached to the stepper motor's shaft (see Figure 2a). Therefore, the thread optimization must be adapted to deal with this physical constraint. It can be implemented by a general function that checks if a connection is allowed: The function `CheckAllowedConnection(PrevPin,NextPin)` at Line 9 returns True if the combination between the pins `PrevPin` and `NextPin` is allowed, and returns False otherwise. For example, if the line defined by the pins `PrevPin` and `NextPin` passes within 3 cm of the plate's center (approximately the motor's shaft radius) and is on the bottom layer, then the function returns False and this connection will be skipped.

A second physical constraint is that two pins too close to each other should not be connected. In order to ensure that our machine can pull a thread between any two pins and hold it firmly in place, we experimentally found that the minimum distance between two consecutive pins should be no smaller than 5°. The function `CheckAllowedConnection()` can simply deal with this case by returning False when the distance between two pins is shorter than a threshold set to 5°.

4 | EXPERIMENTS

4.1 | Implementation and hardware details

Our algorithm was implemented in Python (nonoptimized code) and runs on a laptop equipped with an Intel i7-4790K, a 4.0-GHz CPU, and 32 GB of RAM. Given the input photo provided by the user, the program automatically computes and returns the optimized pin combination as well as a visualization of this pin combination (i.e., the rendered image I_r). The duration of the pin optimization varies greatly with the choice of the cost function. It typically takes around 5–8 hr for the global cost and just 4–6 min for the local cost, depending on the complexity of the input photo and the number of pins. The fabrication takes between 1 and 3 hr, depending on the number of lines, with our current machine. If fabrication time is an issue, the fabrication can be drastically sped up with faster motors and encoders. The total cost of the machine is around 120 USD: Arduino (15 USD), motor driver (30 USD), motor (50 USD), servo (20 USD), plastic cables, cover, and paint (5 USD). The fabrication cost of one thread object is composed of the supporting acrylic plate and the thread, which is less than 10 USD in total.

4.2 | Results

Figure 5 shows several representative results obtained by our approach (local cost) for various kinds of face appearance, in terms of input materials (photos vs. paintings with different artistic styles, for example, expressionism for Van Gogh and mannerism for El Greco), head orientations (frontal vs. three quarter views), facial hair (none vs. beard), facial expressions (neutral vs. extreme with tongue stuck out), age (baby vs. adults), genders (male vs. female), and accessories (with/without glasses and hat). Overall, the photos of the fabricated objects are visually satisfying, resemble the input images, and contain recognizable features of the input faces.

4.2.1 | Quantitative evaluation

We quantitatively evaluate and compare the results obtained by the global and local costs. While the local cost is used to select the line, we can still compute the target global tone cost (after the line is selected). Therefore, we compare the two costs in terms of global tone cost.

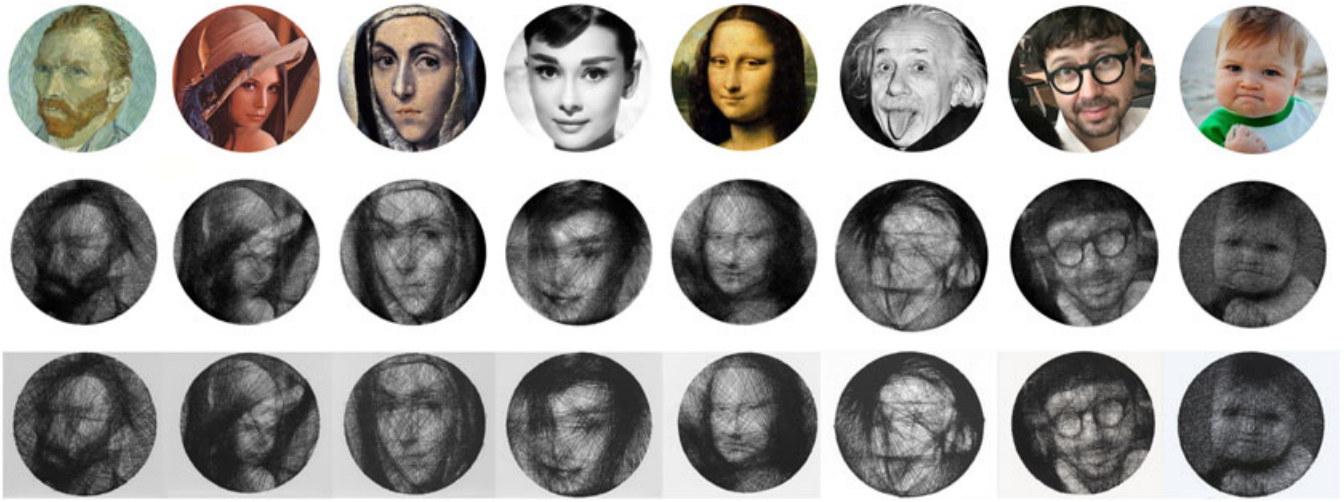

FIGURE 5 Representative results obtained by the proposed approach. Top: input image. Middle: rendering of the thread optimization. Bottom: photo of the fabricated object. From left to right: Van Gogh (self-portrait), Lenna (Lena Söderberg)[‡], The Virgin Mary (by El Greco), Audrey Hepburn, Mona Lisa (by Leonardo da Vinci), Albert Einstein, man with glasses, and the success baby meme[§]. We refer to them as `Van Gogh`, `Lenna`, `Mary`, `Hepburn`, `Mona Lisa`, `Einstein`, `Glasses`, and `Baby`, respectively

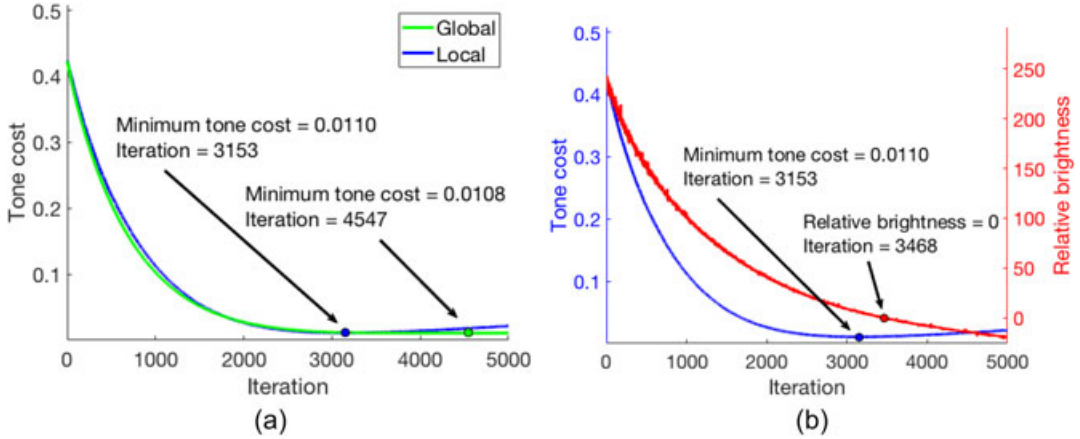

FIGURE 6 Evolution of the global tone cost obtained by the local and global approaches. Iteration on the *x*-axis is equivalent to the number of lines

Given a photo of the Mona Lisa painting, Figure 6a shows the evolution of the global tone cost where the connections are selected by the global and local costs. The local cost reaches a minimum of 0.0110 of tone cost after 3,153 iterations and the global cost a minimum of 0.0108 of tone cost after 4,547 iterations. It is interesting to see that, while the local cost "indirectly" minimizes the global tone cost, its global tone cost is only about 2% higher than the "direct" global cost method. Moreover, the pin optimization took around 6 hr for the global cost, but just around 4 min for the local cost. For completeness, Figure 6b shows the evolution of the brightness difference cost of the selected lines. Figure 7 shows the evolution of the thread result along the iterations of the algorithm for the local cost corresponding to Figure 6a.

The visual results corresponding to the minimum costs of Figure 6a are shown in Figure 8. We measure and compare the PSNR on these images. The PSNR values of the images produced by the local (Figure 8b) and global (Figure 8c) costs

[‡]https://en.wikipedia.org/wiki/Lenna
[§]https://en.wikipedia.org/wiki/Success_Kid

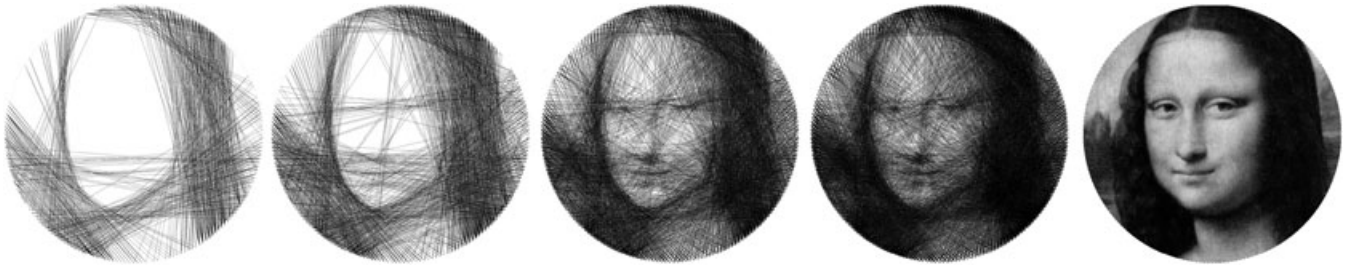

FIGURE 7 Evolution of the thread result at every 750 iterations of the optimization. The target image is shown on the right

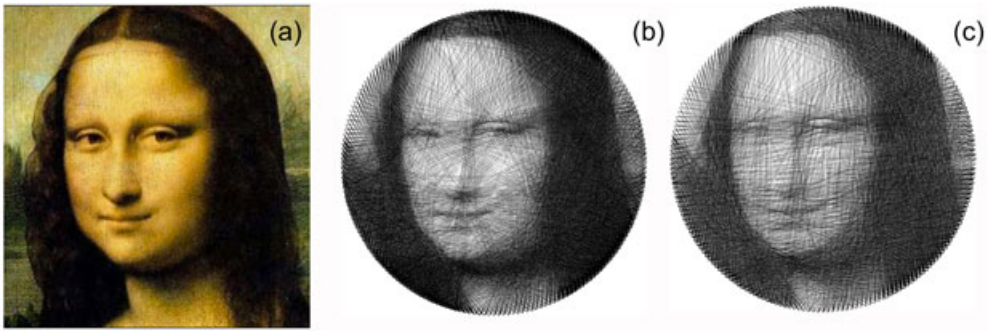

FIGURE 8 Given the Mona Lisa painting as input (a), the results obtained by the local (b) and global (c) costs

are, respectively, 7.51 and 7.64. The relative PSNR difference is just 1.7%, which indicates that the results obtained by the global and local costs are similar in terms of PSNR value.

4.2.2 | Qualitative evaluation with users

In addition to quantitative evaluation, we also conducted a qualitative user study. We recruited 12 participants, and they were shown the input photo, as well as the thread result computed by the global and local costs, in a random order. For both costs, we showed the result when the minimum global tone cost is obtained (see Figure 6 and the discussion in Section 4.2.1). We asked the participants to evaluate the satisfaction statement "I am satisfied with the overall visual quality of the thread result" on a 5-point Likert scale: strongly disagree (1), disagree (2), neutral (3), agree (4), and strongly agree (5). We also asked them to evaluate the resemblance statement "The thread result resembles the input photo" on the same 5-point Likert scale. We conducted the study on all the input images of Figure 5.

The results of the user study are shown in Figure 9. In terms of satisfaction (Figure 9a), the ratings for the global and local costs are respectively as follows: 57% (global) versus 68% (local) of the responses were 4 (agree) or higher, 86% versus 85% were 3 (neutral) or higher, and 14% (global) versus 15% (local) of the responses were negative (disagree or less). On the whole, these ratings suggest that, first, both costs produce satisfying fabrication results overall and, second, both costs lead to a similar level of satisfaction.

To further evaluate the results, Figure 9c shows the evaluation score for each of the input images. It shows that the score range can be large for some input images (e.g., `Van Gogh` or `Baby`) and short for others (e.g., `Mona Lisa` or `Mary`). Globally, eight out of the eight input images for both global and local costs have a median score equal to or higher than 3 (neutral).

User study results for the resemblance to the input photo are available in Figure 9b and Figure 9d. The ratings for the global and local costs are respectively as follows: 65% (global) versus 68% (local) of the responses were 4 (agree) or higher, 88% versus 88% were 3 (neutral) or higher. On the whole, these ratings suggest that both costs provide a satisfying level of resemblance to the input photo and, second, both costs lead to a similar level of resemblance perception. Figure 9d shows the evaluation score for each of the input images. It shows that eight out of the eight input images for both global and local costs have a median score higher than 3 (neutral) or even higher.

In conclusion, given the cost values, the PSNR values, the optimization duration, and the user study evaluation, we conclude that the local cost is preferable over the global cost.

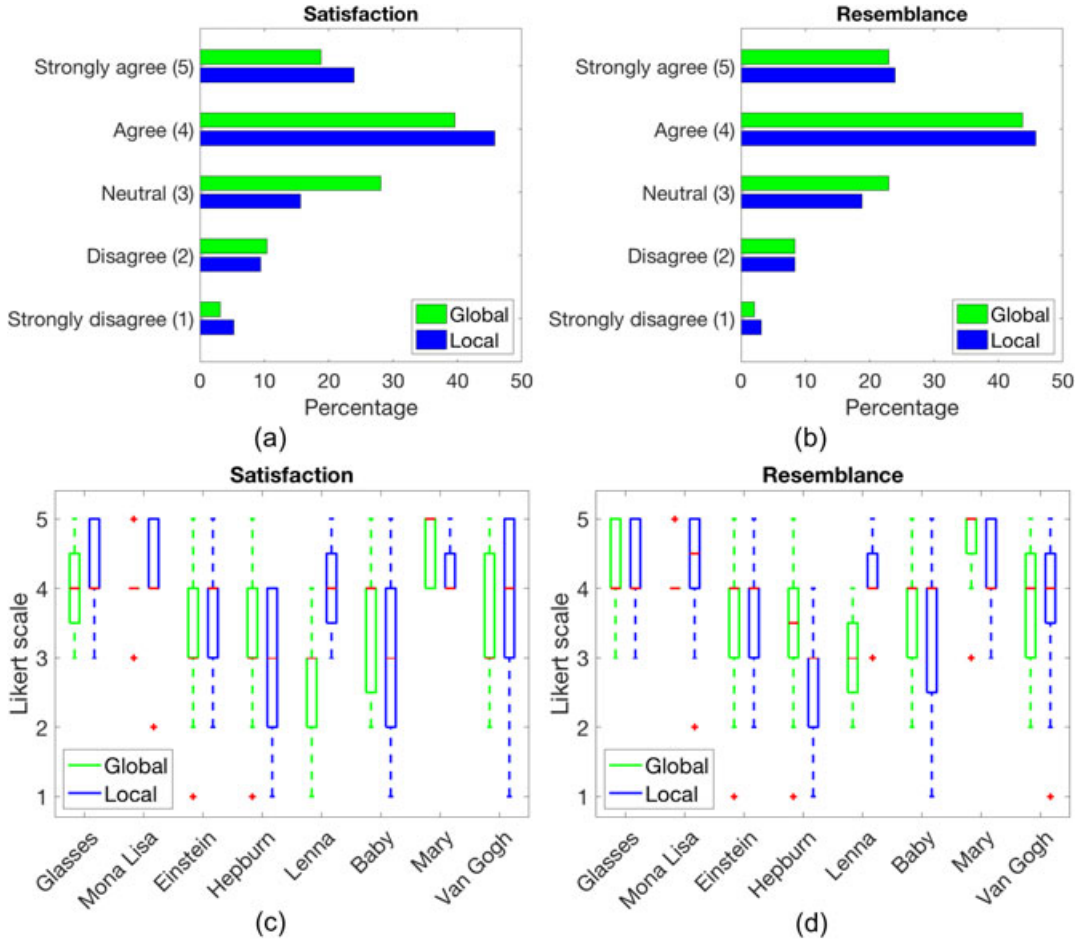

FIGURE 9 User study results for the global and local strategies regarding satisfaction (left) and resemblance to the input photo (right), overall (top) and per input photo (bottom). The result per input photo is displayed by a box plot where the median is drawn in red

4.3 | Additional results

Due to page limit, we invite the readers to refer to the supplementary video and our project website[¶] for additional results and extensions. In the following, we provide short high-level information.

Multiview Our approach can be extended for multiview thread optimization where the sides of a 3D cube can show different images depending on the viewpoint. We consider two input images and assign them on two nonfacing sides of the cube. We render the cube side view from one viewpoint by projecting all the threads along this viewpoint. The multiview cost is computed as the sum of the costs of the two views. For the 3D fabrication, the thread was manually connected, as our current SpiderPrinter is designed for 2D patterns. The video shows a representative result with a 3D cube where a circle is seen from the left view and a heart from the top view.

Colors The supplementary video shows an example of a logo composed of blue and red. We optimize and fabricate the colors layer by layer. For the thread optimization, we first automatically decompose the input logo image into two images: one containing the blue parts of the logo and one containing the red parts. Then, we compute the thread connection for each of these two images independently. For the fabrication, the machine connects the red thread; then, we manually change the red thread spool to the blue one, and then, the machine connects the blue thread.

Stencil Our approach can also be used as a stencil; see examples in the supplementary video for spray painting and for personalized cake decoration by manually sprinkling sugar.

[¶]https://vcail.kaist.ac.kr/projects/SpiderWeb

4.4 | Limitations and future work

In Section 3.2, we introduced a simple-yet-efficient thread optimization algorithm that provides visually satisfying results. In future work, we plan to investigate advanced optimization algorithms to further improve the results quality. In particular, we believe that reinforcement learning has a great potential because it aims to maximize the cumulative long-term reward rather than an immediate short-term reward.

Our current machine allows the fabrication using a thread of a single color at a time. Concretely, for multiple colors, it requires the user to manually substitute the thread spool. Future work will investigate the usage of hardware mechanisms for multicolor object fabrication, such as multiple arms, automatic spool switching mechanism, or thread colorization.

5 | CONCLUSION

We have presented a computational fabrication system to create string art objects using one single thread in such a way that it looks like an input image. Our system runs in a fully automatic manner: the user provides an image, the thread connections are computed by our optimization algorithm, and then, our custom-made machine connects the thread. Experiments with quantitative and qualitative evaluations have successfully demonstrated the quality of our results.

ACKNOWLEDGEMENTS

This work was partially supported by KAIST Interdisciplinary Research on the 4th Industrial Revolution and Adobe Research. Seungwoo Je and Andrea Bianchi were also supported by the National Research Foundation of Korea (NRF) under Grant 2018R1A5A7025409, funded by the Korean government (Ministry of Science and ICT).

ORCID

Jean-Charles Bazin https://orcid.org/0000-0001-7660-4802

REFERENCES

1. Lubin YD, Dorugl S. Effectiveness of single-thread webs as insect traps: sticky trap models. Bull Br Arachnol Soc. 1982;5(9):399–407.
2. McCann J, Albaugh L, Narayanan V, et al. A compiler for 3D machine knitting. ACM Trans Graph. 2016;35(4). Article No. 49.
3. Schüller C, Panozzo D, Grundhöfer A, Zimmer H, Sorkine E, Sorkine-Hornung O. Computational thermoforming. ACM Trans Graph. 2016;35(4). Article No. 43.
4. Galea B, Kry PG. Tethered flight control of a small quadrotor robot for stippling. Proceedings of the 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS); 2017 Sep 24–28; Vancouver, Canada. Piscataway, NJ: IEEE; 2017.
5. Zhao H, Lu L, Wei Y, et al. Printed perforated lampshades for continuous projective images. ACM Trans Graph. 2016;35(5). Article No. 154.
6. Prévost R, Jacobson A, Jarosz W, Sorkine-Hornung O. Large-scale painting of photographs by interactive optimization. Comput Graph. 2016;55:108–117.
7. Shilkrot R, Maes P, Paradiso JA, Zoran A. Augmented airbrush for computer aided painting (CAP). ACM Trans Graph. 2015;34(2). Article No. 19.
8. Skouras M, Thomaszewski B, Bickel B, Gross M. Computational design of rubber balloons. Comput Graph Forum. 2012;31(2pt4):835–844.
9. Skouras M, Thomaszewski B, Kaufmann P, et al. Designing inflatable structures. ACM Trans Graph. 2014;33(4). Article No. 63.
10. Chen W, Ma Y, Lefebvre S, Xin S, Martínez J, Wang W. Fabricable tile decors. ACM Trans Graph. 2017;36(6). Article No. 175.
11. Martin T, Umetani N, Bickel B. OmniAD: data-driven omni-directional aerodynamics. ACM Trans Graph. 2015;34(4). Article No. 113.
12. Birsak M, Rist F, Wonka P, Musialski P. String art: towards computational fabrication of string images. Comput Graph Forum. 2018;37(2):263–274.
13. Kopf J, Lischinski D. Depixelizing pixel art. ACM Trans Graph. 2011;30(4). Article No. 99.
14. Mitra NJ, Chu H-K, Lee T-Y, Wolf L, Yeshurun H, Cohen-Or D. Emerging images. ACM Trans Graph. 2009;28(5). Article No. 163.
15. Chu H-K, Hsu W-H, Mitra NJ, Cohen-Or D, Wong T-T, Lee T-Y. Camouflage images. ACM Trans Graph. 2010;29(4). Article No. 51.
16. Mitra NJ, Pauly M. Shadow art. ACM Trans Graph. 2009;28(5). Article No. 156.
17. Huang H, Zhang L, Zhang H-C. Arcimboldo-like collage using internet images. ACM Trans Graph. 2011;30(6). Article No. 155.
18. Maharik R, Bessmeltsev M, Sheffer A, Shamir A, Carr N. Digital micrography. ACM Trans Graph. 2011;30(4). Article No. 100.
19. Pappas TN, Neuhoff DL. Least-squares model-based halftoning. IEEE Trans Image Process. 1999;8(8):1102–1116.
20. Deussen O, Spicker M, Zheng Q. Weighted linde-buzo-gray stippling. ACM Trans Graph. 2017;36(6). Article No. 233.
21. Kopf J, Cohen-Or D, Deussen O, Lischinski D. Recursive Wang tiles for real-time blue noise. ACM Trans Graph. 2006;25(3):509–518.
22. Chu H-K, Chang C-S, Lee R-R, Mitra NJ. Halftone QR codes. ACM Trans Graph. 2013;32(6). Article No. 217.
23. Wu X. An efficient antialiasing technique. Proceedings of the 18th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '91); 1991 Jul 28–Aug 2; Las Vegas, NV. New York, NY: ACM; 1991.

24. Pang W-M, Qu Y, Wong T-T, Cohen-Or D, Heng P-A. Structure-aware halftoning. ACM Trans Graph. 2008;27(3). Article No. 89.
25. Shen J. Least-squares halftoning via human vision system and Markov gradient descent (LS-MGD): algorithm and analysis. SIAM Rev. 2009;51(3):567–589.
26. Chang J, Alain B, Ostromoukhov V. Structure-aware error diffusion. ACM Trans Graph. 2009;28(5). Article No. 162.

AUTHOR BIOGRAPHIES

Seungwoo Je is currently a PhD student in the Industrial Design department at KAIST. His research interests include tangible interaction, haptics, virtual reality, Wearable, and User-Centered Design.

Yekaterina Abileva is currently a robotics research engineer at Crazing Lab, robotics startup in Korea. Yekaterina received her BSc in Computer Science from KAIST (South Korea). Her research interests include generating and interpreting human motion, pose estimation, sensing and navigation.

Andrea Bianchi is an Assistant Professor in the Department of Industrial Design at KAIST. Andrea received his PhD from KAIST (Korea) in 2012, his masters in Computer Science from NYU (USA) and his BSc in business administration from Bocconi University (Italy). His research interests is primarily in the field of Human-Computer Interaction (HCI), wearable and tangible devices and toolkits for digital fabrication.

Jean-Charles Bazin is Assistant Professor at KAIST, South Korea. He received the MS degree in Computer Science from the Université de Technologie de Compiègne, France, in 2006, and the PhD degree in Electrical Engineering from KAIST in 2011. He worked as Associate Research Scientist at Disney Research Zurich, Switzerland (2014-2016). Before joining Disney Research, he was a postdoc jointly at the Computer Graphics Laboratory headed by Prof. Markus Gross and at the Computer Vision and Geometry Group headed by Prof. Marc Pollefeys at ETH Zurich, Switzerland (2011-2014). He also worked as Postdoctoral Fellow at the Computer Vision Lab of Prof. Katsushi Ikeuchi, University of Tokyo, Japan (2010-2011). He is Area Chair for ICCV 2019 and an invited AI expert in the World Economic Forum Expert Network.

SUPPORTING INFORMATION

Additional supporting information may be found online in the Supporting Information section at the end of the article.

How to cite this article: Je S, Abileva Y, Bianchi A, Bazin J-C. A computational approach for spider web-inspired fabrication of string art. *Comput Anim Virtual Worlds*. 2019;30:e1904. https://doi.org/10.1002/cav.1904