DOI: 10.1111/cgf.14770
EUROGRAPHICS 2023 / K. Myszkowski and M. Nießner
(Guest Editors)

COMPUTER GRAPHICS *forum*
Volume 42 (2023), Number 2

Directionality-Aware Design of Embroidery Patterns

Liu Zhenyuan[1,2], Michal Piovarči[1], Christian Hafner[1], Raphaël Charrondière[1], Bernd Bickel[1]

[1]ISTA, Austria [2]EPFL, Switzerland

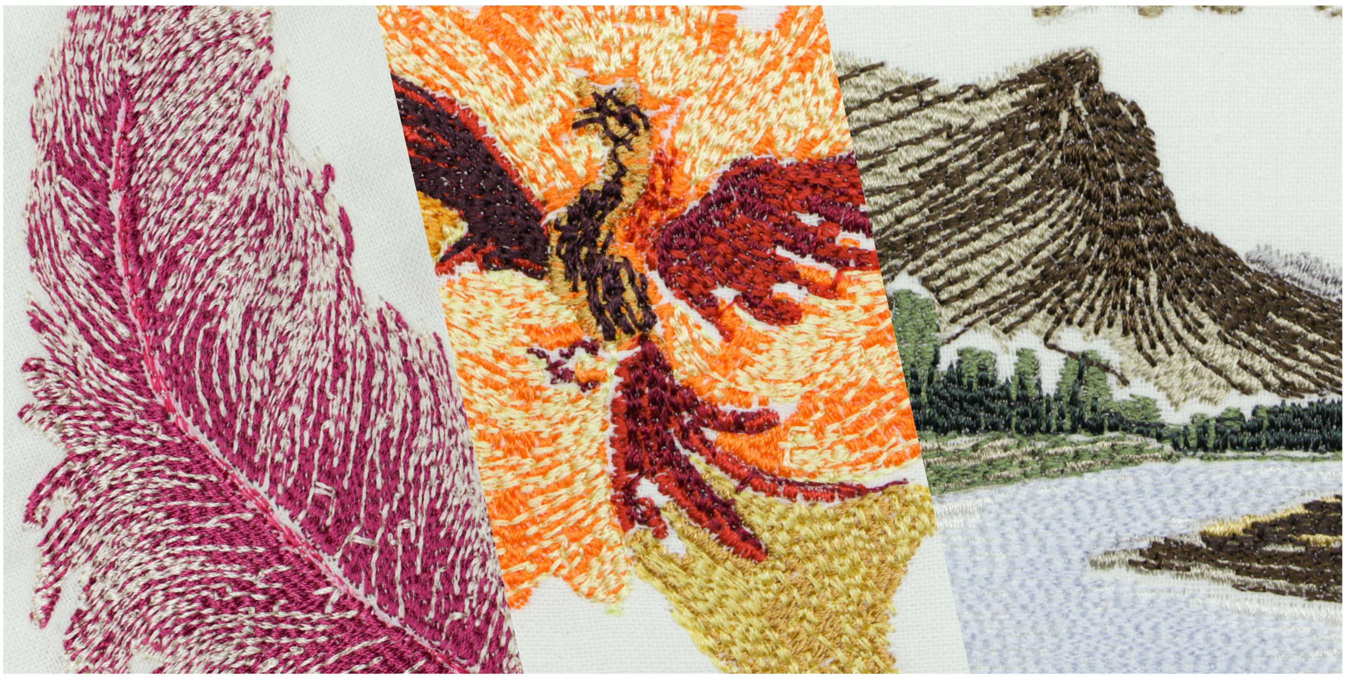


Figure 1: *Close-ups of embroidery patterns generated using our method stitched on ivory cloth; see Figure 14 for full images.*

Abstract

Embroidery is a long-standing and high-quality approach to making logos and images on textiles. Nowadays, it can also be performed via automated machines that weave threads with high spatial accuracy. A characteristic feature of the appearance of the threads is a high degree of anisotropy. The anisotropic behavior is caused by depositing thin but long strings of thread. As a result, the stitched patterns convey both color and direction. Artists leverage this anisotropic behavior to enhance pure color images with textures, illusions of motion, or depth cues. However, designing colorful embroidery patterns with prescribed directionality is a challenging task, one usually requiring an expert designer. In this work, we propose an interactive algorithm that generates machine-fabricable embroidery patterns from multi-chromatic images equipped with user-specified directionality fields. We cast the problem of finding a stitching pattern into vector theory. To find a suitable stitching pattern, we extract sources and sinks from the divergence field of the vector field extracted from the input and use them to trace streamlines. We further optimize the streamlines to guarantee a smooth and connected stitching pattern. The generated patterns approximate the color distribution constrained by the directionality field. To allow for further artistic control, the trade-off between color match and directionality match can be interactively explored via an intuitive slider. We showcase our approach by fabricating several embroidery paths.

CCS Concepts
• ***Computing methodologies*** → ***Shape modeling;***

1. Introduction

The embroidery machine is one of the oldest fully automated devices, starting from designs punched on paper tapes and continuing in modern times with numerically controlled machines featuring several stitching heads. To create the desired appearance, embroidery machines weave colored threads into a cloth substrate. This process is governed by strict hardware constraints, such as a minimal stitch length, and connectedness of adjacent stitches to form stitch lines. Because of these constraints, the fabrication process generates a characteristic anisotropic appearance [GLL*21] with a limited design space. By trial and error, artists have learned to navigate this design space and leverage the anisotropy to create various effects. For example, horizontal and vertical directions are used to convey depth, diagonal lines convey movement and a sense of energy, and curved lines evoke emotions while softening edges. For more embroidery practices in real life, we refer the readers to a book [Nee18] that covers many of the aspects of this traditional handcraft technique.

Despite the importance of thread directionality, commercial embroidery software [Ber21] offers only limited control options, for example, a set of pre-programmed parametric designs. Alternatively, it hides the anisotropy while partitioning via color quantization and filling partitions with randomized stitches; see Figure 11. As a result, reproducing features such as a flowing river requires the manual generation of several regions that approximate the river flow.

One of the key reasons why only limited control over anisotropy is provided is the challenging interaction between spatially varying directionality and thread density. When generating a curve network following a direction field at a given density, it is impossible to satisfy both requirements exactly. Despite the availability of algorithms for the generation of these so-called stripe patterns, none satisfy the unique demands for generating stitching paths and providing enough artistic control. We propose a new algorithm for generating a stitching path, based on sampling the spawning points and terminal points of curve segments that are guaranteed to yield a distribution with the correct *average* directionality and density. In the next step, we include the designer in the loop to strike an aesthetic balance between faithfully reproducing directionality and density also on a *local* level.

The entire pipeline of our interactive algorithm for the generation of embroidery patterns with desired directionality is illustrated in Figure 2. Our input is a multichromatic image representing the density of individual colors and a direction field (Figure 2, Input). The first step, and one of our technical contributions, is the generation of *sources* and *sinks* in the interior and on the boundary of a colored region (Figure 2, Sources). The distribution of these points is guaranteed to be such that the streamlines (Figure 2, Streamlines) traced through the direction field from sources to sinks will reproduce the density given by the colors of the input. To explicitly address the trade-off between adhering to the density and direction field, we introduce an interactive regularization step (Figure 2, Regularization), in which the artist can explore different options, as illustrated in Figure 7. Finally, we postprocess the set of streamlines to yield one continuous path and fabricate it on an embroidery machine (Figure 2, Fabrication).

To demonstrate the usefulness of our approach, we created a design interface that facilitates the conversion of images into colored regions with a direction field and, ultimately, into a stitching path. We showcase the results of our design pipeline by fabricating several automatically generated stitching patterns.

2. Related Work

Embroidery modeling is a long-standing problem in computer graphics, and one of its important aspects is the intricate anisotropic appearance of embroidered threads. For embroidery visualization, there is work that proposes modeling the thread as a bundle of elongated cylinders [CMKM12]. The cylinders can be further refined to incorporate the bending of a thread as it enters and leaves the fabric [GLL*21]. The resulting visualizations yield a faithful representation of the final stitched designs [YSMY16, MS21]. The visualization can be enhanced to automatically convert input images to stitch representation where stitches are parametrized by their starting and ending locations [CSZ17, QXC*19, QCX*20]. Even though these methods can produce images in a realistic embroidery style, their final design tends to contain many disconnected stitches, making it hard to be directly sewed on an embroidery machine.

Fabrication constraints are a key factor in the successful design of embroidery patterns. To guarantee stitchable patterns, commercial software [Ber21] only provides a restricted set of predefined parametric patterns, from which the user can choose. This paradigm was extended to semi-automatically convert an input image into a set of regions with user-selected textures [GLL*21]. An alternative to manual pattern selection is the automatic conversion of images to embroidery designs. These automatic tools mask the anisotropy of the threaded designs by generating patterns with randomized stitch directions. However, this randomization does not exploit the full capabilities of the hardware to use the thread direction for texturing. Thus, researchers investigated how to convert input images into designs suitable for continuous line-drawing systems [KLC07, WT13, LM14, CPG15]. [LHM17] proposed a fully automatic method for generating quilt path from images, with the path constrained being a single loop and aggregating the edge information extracted from the image. Our proposed method is for generating fabricable embroidery designs from images, and the key difference is that we aim to produce colored infills. Therefore, our technique can be seen as a complimentary, where the main color information could be manufactured using our method, and then, the edges could be enhanced with the method proposed by [LHM17].

Non-photorealistic rendering researchers have also investigated the problem of generating images that are composed of strokes with a prescribed density and/or directionality. Automated methods were designed for various painting techniques, including pencil drawings [LXJ12], inks [Ahm14], and oil paintings [HFL11]. Another interesting technique is hatching, where shades are reproduced by tightly spacing parallel lines [WS94, ZISS04, KSZ18, LFH*19]. However, these techniques are tightly tied to the painting method of choice and, as a result, cannot be readily applied to embroidery, where many of the original assumptions, such as color blending, are violated.

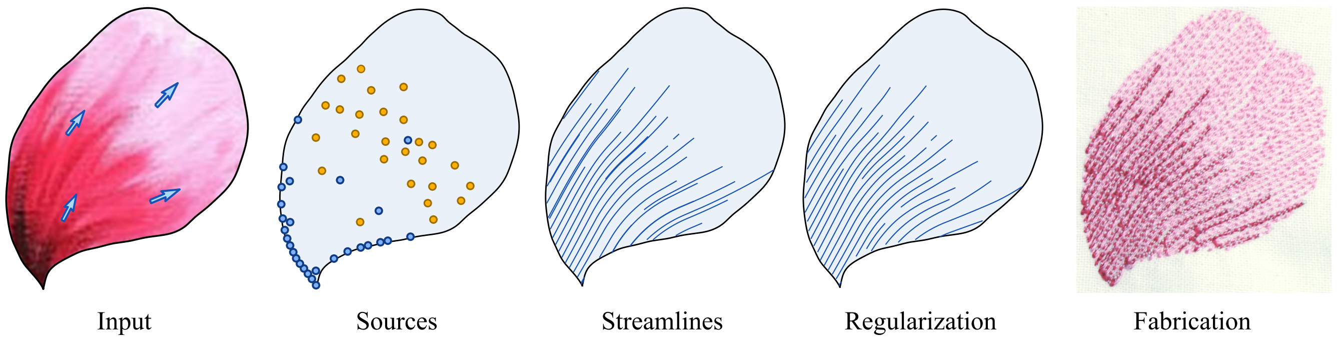


Figure 2: *Our method takes as input a density and a direction field, extracted from an image and user input direction (blue arrows). We sample the sources (blue dots) and sinks (orange dots) according to the divergence field of the combination of density and direction field. Then, we trace and trim streamlines from the sources and sinks. Next, we run a regularization step to promote even spacing between the streamlines. Finally, with postprocessing, a single connected path is constructed and embroidered by a programmable embroidery machine.*

Another option to reproduce the desired density is to generate continuous lines passing through key image features. This typically involves first generating a set of points that approximate the desired image either manually or via stippling techniques [Sec02]. These points can then be treated as a graph, and connecting lines can be generated by solving a traveling salesman problem, or a minimal spanning tree [Ahm15]. The edges of the graph can be further weighted to produce more aesthetically pleasing results [IU09]. Although these methods can capture the density information well, the directionality is often neglected. In contrast, we seek to design a pattern that respects user-supplied density and directionality fields.

Visualization techniques for prescribed density and directionality fields are another option to reproduce an image [TB96]. The techniques are either based on tracing the streamlines directly [JL97, MAD05, LS20] or through a generative noise function [LH06, LLD10]. The image-based methods can be further extended to trace the streamlines on meshes [CDS10, SLCZ09, KCPS13] or utilize curve-based [TWY*20] or image-based [HWYZ20] primitives. For a more detailed overview of these techniques please see [DGK07]. Recently, [BCOM*22] proposed a method to generate infills of a constant density while providing control over anisotropy via a directionality field. In contrast, we aim to control both the density and directionality based on user-supplied input.

Two promising techniques for the generation of stripe patterns are those proposed in [KCPS15] and [TEZ*19, TTZ*20]. The former does not have fabrication as a goal, but yields excellent results for inputs in which the gradient of the density field aligns with the direction field. The latter leverages parametric noise functions to generate images with a prescribed density and directionality. We compare these two methods to ours in Figure 10 and find that our results are more regular and contain fewer artifacts in regions with a changing density. Furthermore, our method provides a higher level of user control by allowing users to interactively explore a range of designs with an intuitive slider.

3. Design of Embroidery Patterns

The design space of sewable embroidery is constrained by hardware limitations. The machine has access to a limited selection of threads and can typically load only up to 10 colors. Moreover, the thread-cutting function is slow and unreliable, often leaving strands that require manual cleaning. Therefore, each color in the design should be carefully considered to avoid unnecessary thread swapping or cutting. We reflect these constraints in our method.

Our pipeline takes as input a simply connected domain $\Omega \subset \mathbb{R}^2$, a density function $\alpha : \Omega \to \mathbb{R}_{>0}$, and an analytical or user-supplied unit vector field $\mathbf{v} : \Omega \to S^1 \subset \mathbb{R}^2$, both of which are assumed to be at least weakly differentiable. The goal of our algorithm is to find a continuous space-filling curve $\gamma : (0, \ell) \to \Omega$ that approximates α and $\mathbf{v}$ in the following sense: First, the direction of $\gamma(s)$ should follow the vector field $\mathbf{v}(\gamma(s))$, or the opposite direction $-\mathbf{v}(\gamma(s))$ everywhere. Second, the distance between neighboring segments of γ that run parallel (or anti-parallel) to each other should be approximately equal to the reciprocal of the density, $1/\alpha$. In the end, γ is converted to a fabricable stitching part. We break down the generation of γ into three steps:

1. Even though the final goal is to produce one continuous curve, we start out by generating a set of *streamlines* that cover Ω. Each streamline is traced from a *source* point to a *sink* point. The first step is to sample these sources and sinks at suitable locations.
2. After tracing the streamlines, they are post-processed to improve the approximation of the direction and density fields.
3. We add connecting segments between neighboring streamlines to form a spanning tree. In a final step, we generate a stitching path γ that traverses the tree in depth-first order, thereby doubling all edges of the tree.

The following subsections elaborate on each step and give the algorithmic details.

3.1. Distribution of Sources and Sinks

Our first goal is to trace a set of streamline segments through the vector field $\mathbf{v} : \Omega \to S^1$, such that two neighboring streamlines are approximately $1/\alpha$ apart. It is intuitive that more streamlines are needed in high-density regions, and fewer streamlines in low-density regions, so we need to spawn or terminate streamlines as needed. We do this by sampling a set of sources and sinks, which mark the locations where streamlines begin and end, respectively. Here, we argue that the distribution of sources and sinks is inherently related to the divergence of the vector field formed by the product of α and $\mathbf{v}$.

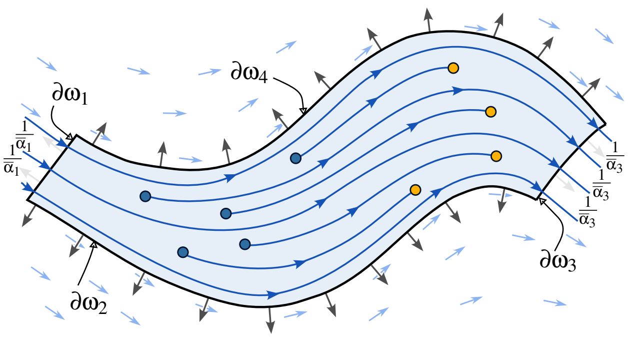

$\partial\omega_1$
$\partial\omega_4$
$\frac{1}{\bar{\alpha}_1}$
$\frac{1}{\bar{\alpha}_1}$
$\frac{1}{\bar{\alpha}_3}$
$\frac{1}{\bar{\alpha}_3}$
$\frac{1}{\bar{\alpha}_3}$
$\partial\omega_3$
$\partial\omega_2$

Figure 3: *Streamtube ω is formed by two curves $\partial\omega_1, \partial\omega_3$ that are orthogonal to $\mathbf{v}$ (light blue arrows in the background), and $\partial\omega_2, \partial\omega_4$ that are tangent to $\mathbf{v}$. The number of sources (5 blue dots), sinks (4 orange points), streamlines across $\partial\omega_1$ (3 entering on the left), and streamlines across $\partial\omega_3$ (4 exiting on the right) satisfy Equation (1).*

Let this vector field be denoted by $\mathbf{Z}(\mathbf{p}) := \alpha(\mathbf{p})\mathbf{v}(\mathbf{p})$. To illustrate the connection between sources/sinks and $\nabla \cdot \mathbf{Z}$, consider a subdomain $\omega \subset \Omega$ shaped like the one in Figure 3, sometimes called a *streamtube* of $\mathbf{v}$. The boundary $\partial\omega$ is formed by four smooth curves $\partial\omega_i$, $i = 1, \ldots, 4$, such that $\partial\omega_1$ and $\partial\omega_3$ are orthogonal to $\mathbf{v}$, and $\partial\omega_2$ and $\partial\omega_4$ tangent to $\mathbf{v}$. Next, assume that we are given a set of streamline segments that satisfy the density requirement of being $1/\alpha$ apart. Because this holds also at the boundary, we observe:

$$\int_{\partial\omega_1} \alpha = \bar{\alpha}_1 \cdot \text{length}(\partial\omega_1) = \#\text{streamlines across } \partial\omega_1,$$

where $\bar{\alpha}_1$ denotes the average of α on $\partial\omega_1$. The last equality holds since each streamline takes a width of $1/\alpha$. Similarly, the integral of α over $\partial\omega_3$ gives the number of streamlines crossing $\partial\omega_3$.

Now we can establish the relationship between sources/sinks and $\mathbf{Z}$, using the divergence theorem in the first step:

$$\begin{aligned} \int_\omega \nabla \cdot \mathbf{Z} &= \int_{\partial\omega} \alpha \mathbf{n} \cdot \mathbf{v} = \int_{\partial\omega_3} \alpha - \int_{\partial\omega_1} \alpha \qquad (1) \\ &= \#\text{streamlines across } \partial\omega_3 - \#\text{streamlines across } \partial\omega_1 \\ &= \#\text{sources} - \#\text{sinks}, \end{aligned}$$

where $\mathbf{n}$ stands for the outward-pointing normal along $\partial\omega$. The last equality holds because the difference between the number of streamlines exiting (across $\partial\omega_3$) and entering (across $\partial\omega_1$) must be equal to the difference between the number of streamlines starting and terminating in ω; see Figure 3 for an example.

This argument can be extended to arbitrary subregions of Ω, because any subregion can be approximated by a union of streamtubes. Thus, we find the following general rule: *The difference between the number of sources and sinks in any subregion of Ω must be equal to the integral of the divergence over that subregion.*

Naturally, we can only follow this rule approximately, because there is a finite number of sources and sinks. Therefore, we cannot reproduce $\int_\omega \nabla \cdot \mathbf{Z}$ exactly for arbitrary $\omega \subset \Omega$. We resolve this issue by relaxing the problem: find a finite partition $\omega_1, \ldots, \omega_n$ of Ω, such that Equation (1) holds exactly for any ω that can be represented as an arbitrary union of these ω_i—see the rectangular partition in Figure 4 for an example. For other subregions at a similar resolution, Equation (1) will only hold approximately.

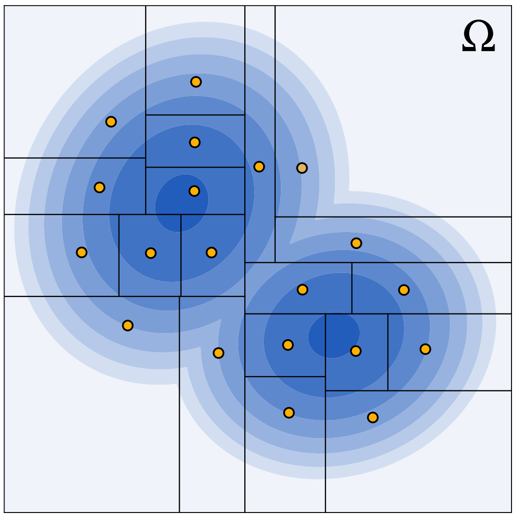


Figure 4: *We leverage a k-d tree to partition the domain Ω into many regions, of which each has the function w integrated to 1. The background color encodes the weight function w; a darker color indicates a larger value of w. A source (orange) is placed at the center of mass of each region.*

Placing sources in negative-divergence regions causes the streamlines to spawn where density should decrease, creating visual artifacts. Also, these streamlines need to be terminated quickly to correct the error, resulting in very short segments. To avoid this, we partition Ω into $\Omega^+ := \{\mathbf{p} \in \Omega : \nabla \cdot \mathbf{Z}(\mathbf{p}) > 0\}$ and $\Omega^- := \{\mathbf{p} \in \Omega : \nabla \cdot \mathbf{Z}(\mathbf{p}) < 0\}$ and perform the sampling of sources only in Ω^+ and of sinks only in Ω^- independently of each other.

3.2. Sampling Sources and Sinks

Sampling points according to a prescribed density field is a long-standing computer graphics problem [LWSF10, MARI17], and there are algorithms that can be used to approximate the input density. However, since our aim is not to visually match a density field but rather to solve Equation (1), we would like the sampling to provide additional guarantees: (1) generate the exact number of sources and sinks as needed; (2) the sources and sinks match positive and negative divergence as closely as possible, and (3) have a fast runtime. In this subsection, we provide a sampling algorithm specifically tailored to solve Equation (1).

Here we describe our algorithm for sampling sources in Ω^+, with special treatment for the boundary $\partial\Omega \cap \Omega^+$. The algorithm for sampling sinks in Ω^- works the same way, except that all appearances of Ω^+ need to be replaced by Ω^-, and of $\nabla \cdot \mathbf{Z}$ by $-\nabla \cdot \mathbf{Z}$. After these steps, we end up with four point sets: sources S_{int} in the interior of Ω^+ and S_{bdry} on the boundary $\partial\Omega \cap \Omega^+$, as well as sinks T_{int} in the interior of Ω^- and T_{bdry} on the boundary $\partial\Omega \cap \Omega^-$; please see Figure 5 for instances of each point set. Also note that these four sets are sampled independently.

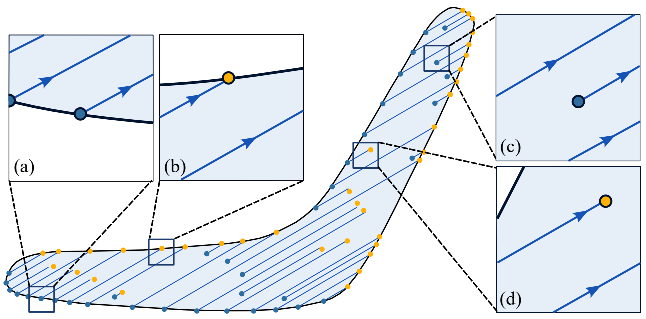


Figure 5: *Sources and sinks sampled on the boundary and in the interior. Instances of $S_{bdry}, T_{bdry}, S_{int}$, and T_{int} are shown in (a) – (d), respectively.*

Sources in the interior. First, we describe the algorithm for generating S_{int}. As mentioned, we do not sample sinks in Ω^+; then, by Equation (1), we see that the integral $\int_{\Omega^+} \nabla \cdot \mathbf{Z}$ must equal the total number of sources. However, the number of sources is an integer, whereas the integral term is most likely not. The smallest modification we can make is to scale the input signal $\mathbf{Z}$, such that the integral becomes an integer. We do so by having $\tilde{\mathbf{Z}} = \frac{\lceil \int_{\Omega^+} \nabla \cdot \mathbf{Z} \rceil}{\int_{\Omega^+} \nabla \cdot \mathbf{Z}} \mathbf{Z}$. The scaling factor usually is close to 1 for a realistic input, because for most of the input we need to generate many sources. After this scaling, $n := \int_{\Omega^+} \nabla \cdot \tilde{\mathbf{Z}}$ gives the number of sources we need to distribute.

We distribute n sources by partitioning Ω^+ into $\omega_1, \dots, \omega_n$ such that $\int_{\omega_i} \nabla \cdot \tilde{\mathbf{Z}} = 1$ for all $i = 1, \dots, n$, and place one source per ω_i. As a heuristic, we place the source point $\mathbf{s}_i \in \Omega^+$ at the location of concentrated divergence, i.e., the center of mass of $\nabla \cdot \tilde{\mathbf{Z}}$ on ω_i:

$$\mathbf{s}_i = \frac{\int_{\omega_i} (\nabla \cdot \tilde{\mathbf{Z}}(\mathbf{p}))\, \mathbf{p}\, \mathrm{d}\mathbf{p}}{\int_{\omega_i} \nabla \cdot \tilde{\mathbf{Z}}}.$$

This guarantees that integrating $\nabla \cdot \tilde{\mathbf{Z}}$ over Ω^+ gives approximately the same result as counting the number of sources.

To construct the partition, we take the axis-aligned bounding box B^+ of Ω^+, and define $w : B^+ \to \mathbb{R}_{\geq 0}$ by

$$w(\mathbf{p}) := \begin{cases} \nabla \cdot \tilde{\mathbf{Z}}(\mathbf{p}) & \text{if } \mathbf{p} \in \Omega^+, \\ 0 & \text{otherwise.} \end{cases}$$

The idea is to compute a k-d tree decomposition of B^+ such that w integrates exactly to 1 on each leaf cell. Our procedure for doing this is shown in Algorithm 1, but we need to verify that Step 6, which subdivides a cell, is always possible.

To show this, consider a rectangle $\Omega = [x_1, x_2] \times [y_1, y_2]$ with $\int_\Omega w = m$. Then, we can define $f(x) := \int_{[x_1,x] \times [y_1,y_2]} w$, which integrates w on a subrectangle of Ω and satisfies $f(x_1) = 0$ and $f(x_2) = m$. From the intermediate value theorem and continuity of f, it follows that there exists $x^* \in (x_1, x_2)$ such that $f(x^*) = \lfloor m/2 \rfloor$. This shows that x^* marks the location of a vertical line that subdivides Ω in the way required by Algorithm 1. In practice, we subdivide along a vertical line if $x_2 - x_1 > y_2 - y_1$, and along a horizontal line otherwise. Algorithmically, we find x^* (or y^*) using binary search. The final partition $\omega_1, \dots, \omega_n$ is given by intersecting every rectangular region at a leaf of the k-d tree with Ω^+.

Algorithm 1 FINDSOURCESINREGION(Ω, w) $\to S_{\text{int}}$

```
 1: $m \leftarrow \iint_\Omega w\,\mathrm{d}A$                              ▷ Integrate over a 2D region
 2: if $m = 1$ then
 3:     $S_{\text{int}} \leftarrow \{\iint_\Omega w(\mathbf{p})\mathbf{p}\,\mathrm{d}\mathbf{p} \,/\, \iint_\Omega w\,\mathrm{d}A\}$          ▷ Center of mass
 4: else if $m > 1$ then
 5:     $n \leftarrow \lfloor m/2 \rfloor$
 6:     Partition $\Omega$ into $\Omega_1, \Omega_2$ such that $\iint_{\Omega_1} w\,\mathrm{d}A = n$
 7:     $S_1 \leftarrow$ FINDSOURCESINREGION($\Omega_1, w$)
 8:     $S_2 \leftarrow$ FINDSOURCESINREGION($\Omega_2, w$)
 9:     $S_{\text{int}} \leftarrow S_1 \cup S_2$
10: end if
```

Sources on the boundary. To generate the set of sources S_{bdry} on the boundary portion $\Gamma^+ := \partial\Omega \cap \Omega^+$, we can employ a strategy very similar to that of the interior, and subdivide the boundary based on a particular density function analogous to $\nabla \cdot \mathbf{Z}$. To derive this density function, we view the boundary as part of the signal $\mathbf{Z}$ by defining a function extension $\bar{\mathbf{Z}}(\mathbf{p}) := \mathbf{Z}(\mathbf{p})$ for $\mathbf{p} \in \Omega^+$, and $\bar{\mathbf{Z}}(\mathbf{p}) = \mathbf{0}$ for $\mathbf{p} \notin \Omega^+$.

This extended signal $\bar{\mathbf{Z}}$ has a discontinuity along Γ^+, but we can write down its divergence as a distribution:

$$\nabla \cdot \bar{\mathbf{Z}}(\mathbf{p}) = \begin{cases} \nabla \cdot \mathbf{Z}(\mathbf{p}) & \text{if } \mathbf{p} \in \Omega^+ \setminus \partial\Omega, \\ \mathbf{n}(\mathbf{p}) \cdot \mathbf{Z}(\mathbf{p})\, \delta(\mathbf{p}) & \text{if } \mathbf{p} \in \Gamma^+, \\ 0 & \text{otherwise,} \end{cases}$$

where δ denotes the Dirac distribution on Γ^+. The term $\mathbf{n} \cdot \mathbf{Z}$ is part of the divergence which is concentrated at Γ^+, and we sample sources from it. Following the same reasoning as before, we scale $\mathbf{Z}$ to be $\tilde{\mathbf{Z}} := \frac{\lceil \int_{\Gamma^+} \mathbf{n} \cdot \mathbf{Z} \rceil}{\int_{\Gamma^+} \mathbf{n} \cdot \mathbf{Z}} \mathbf{Z}$. Then we will distribute the sources as per the integral of term $u := \mathbf{n} \cdot \tilde{\mathbf{Z}}$ using Algorithm 2, which is the one-dimensional equivalent of Algorithm 1. The main difference is that during the partitioning step, which is Step 6, we subdivide a curve into two parts instead of a rectangle. The argument why this is always possible follows—as before—from the intermediate value theorem, used on an antiderivative of u along Γ^+.

3.3. Tracing and Trimming

Our sampling of sources and sinks guarantees that the streamlines will be generated at the correct average density, given by α. However, since source and sink generation are independent, a streamline traced from a source is not guaranteed to flow directly into a sink. A greedy approach, such as assigning every sink to the closest streamline, will fail in general, as illustrated in Figure 6.

Algorithm 2 FINDSOURCESONCURVE(Γ, u) $\to S_{\text{bdry}}$

```
 1: $m \leftarrow \int_\Gamma u\,ds$
 2: if $m = 1$ then
 3:     $S_{\text{bdry}} \leftarrow \{\int_\Gamma u(\mathbf{p})\mathbf{p}\,ds \,/\, \int_\Gamma u\,ds\}$        ▷ Center of mass
 4: else if $m > 1$ then
 5:     $n \leftarrow \lfloor m/2 \rfloor$
 6:     Partition $\Gamma$ into $\Gamma_1, \Gamma_2$ such that $\int_{\Gamma_1} u\,ds = n$
 7:     $S_1 \leftarrow$ FINDSOURCESONCURVE($\Gamma_1, u$)
 8:     $S_2 \leftarrow$ FINDSOURCESONCURVE($\Gamma_2, u$)
 9:     $S_{\text{bdry}} \leftarrow S_1 \cup S_2$
10: end if
```

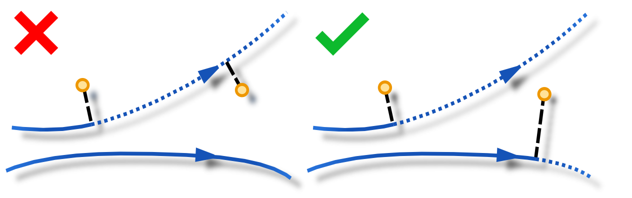
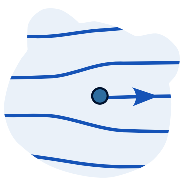


Figure 6: *We intend to ensure that each streamline pairs with one sink. The greedy approach that always assigns the sink to the closest streamline might violate our intention because of an improper order of processing the sinks, whereas solving a weighted assignment problem guarantees that the constraint is satisfied.*

To resolve this ambiguity, we propose the following approach. First, we trace a streamline from every source to the boundary, using the method advocated by [TB96] with a second-order Runge-Kutta integrator. To make sure that every sink trims exactly one streamline while choosing streamlines and sinks close to each other, we formulate an *assignment problem*. The cost associated with assigning a streamline to a sink is given by the shortest distance between them. Thus, a low cost is associated with streamlines that run very close to a sink. The goal is to find an assignment that minimizes the total cost, i.e., the sum of all shortest distances between the sink-streamline pairs. The globally optimal solution can be found via a linear program [Cro16].

After the optimal assignment is found, each streamline is trimmed at the point closest to its assigned sink. This results in a set of streamline segments with the correct *average* density, as illustrated in Figure 2 (center). The next step is to improve the density also locally and to achieve a uniform spacing of $1/\alpha$ between neighboring streamline segments.

3.4. Density vs. Directionality



There is a trade-off between locally improving streamline density and keeping the streamlines aligned with the direction field. The inset (horizontal direction field) illustrates why: Whenever a new streamline emerges from a source, this creates a sudden jump in density at this point. To correct for this jump, the two neighboring streamlines have to flow "around" the new streamline, as shown in the inset figure, which worsens the alignment with the direction field in turn.

The four panels in Figure 7 show different trade-offs between a faithful reproduction of density and directionality. Since the choice between them is purely artistic, we leave it up to the user by exposing a slider that controls the relative weights of a density and a directionality objective.

Notation. We formulate the problem of regularizing the streamlines as a quadratic optimization problem. The optimization variables are the vertex positions of the streamlines, which are represented as discrete curves. We denote the position of the i-th vertex of the k-th streamline by $\mathbf{p}_i^k$, and set $\mathbf{e}_i^k := \mathbf{p}_{i+1}^k - \mathbf{p}_i^k$. The initial position of a vertex (before optimization) is denoted by $\bar{\mathbf{p}}_i^k$, and analogously, $\bar{\mathbf{e}}_i^k := \bar{\mathbf{p}}_{i+1}^k - \bar{\mathbf{p}}_i^k$. Furthermore, we set $\bar{\mathbf{n}}_i^k$ to be a unit vector orthogonal to $\bar{\mathbf{e}}_i^k$.

Energy. In total, we introduce four energy terms: a direction term that penalizes the deviation of streamline edges from their original direction; a density term that penalizes two neighboring streamlines if their distance deviates from the ideal distance $1/\alpha$; and two regularization terms.

The **directionality term** runs over all streamline edges,

$$E_{\text{dir}} = \sum_{k,i} (\mathbf{e}_i^k \cdot \bar{\mathbf{n}}_i^k)^2,$$

and penalizes deviations of the edge from being orthogonal to the normal. The energy of each edge is implicitly scaled by the length of $\mathbf{e}_i^k$, so long edges incur a greater penalty.

The **density term** involves the distance between vertices of neighboring streamlines. To identify pairs of such vertices, we compute a *Delaunay triangulation* on the vertices $\bar{\mathbf{p}}_i^k$, constrained to contain all streamlines edges, as shown in Figure 8. The additional edges present in the Delaunay triangulation will connect vertices of neighboring streamlines that are the closest together – denote the set of these Delaunay edges by $\mathcal{D}$. We use the density at the midpoint of an edge to represent the varying density across the edge. Let $\bar{\mathbf{m}}_{i,j}^{k,l} := \frac{1}{2}(\bar{\mathbf{p}}_i^k + \bar{\mathbf{p}}_j^l)$ the midpoint of an edge in $\mathcal{D}$. Then, the density energy reads as follows:

$$E_{\text{den}} = \sum_{(\bar{\mathbf{p}}_i^k, \bar{\mathbf{p}}_j^l) \in \mathcal{D}} \left(\alpha(\bar{\mathbf{m}}_{i,j}^{k,l})\,(\mathbf{p}_j^l - \mathbf{p}_i^k) \cdot \mathbf{v}^\perp(\bar{\mathbf{m}}_{i,j}^{k,l}) - 1 \right)^2.$$

This energy is small if the projection of a Delaunay edge onto the normal $\mathbf{v}^\perp$ of the direction field $\mathbf{v}$ is close to the ideal distance $1/\alpha$.

In addition, there are two **regularization terms**: one term to prevent stretching of edges and their shifting along the direction field. the other one is the general regularity term;

$$E_{\text{stretch}} = \sum_{k,i} \left(\bar{\mathbf{e}}_i^k \cdot (\mathbf{p}_i^k - \bar{\mathbf{p}}_i^k) \right)^2 + \left(\bar{\mathbf{e}}_i^k \cdot (\mathbf{p}_i^{k+1} - \bar{\mathbf{p}}_i^{k+1}) \right)^2,$$

$$E_{\text{reg}} = \sum_{k,i} \|\mathbf{p}_i^k - \bar{\mathbf{p}}_i^k\|^2,$$

summing over all streamline edges and vertices, respectively.

The total energy is given by

$$E = w_{\text{den}} E_{\text{den}} + w_{\text{dir}} E_{\text{dir}} + w_{\text{stretch}} E_{\text{stretch}} + w_{\text{reg}} E_{\text{reg}}, \quad (2)$$

and its unique minimizer can be found by one sparse linear solve as it is quadratic in $\mathbf{p}_i^k$. The default values of the weights

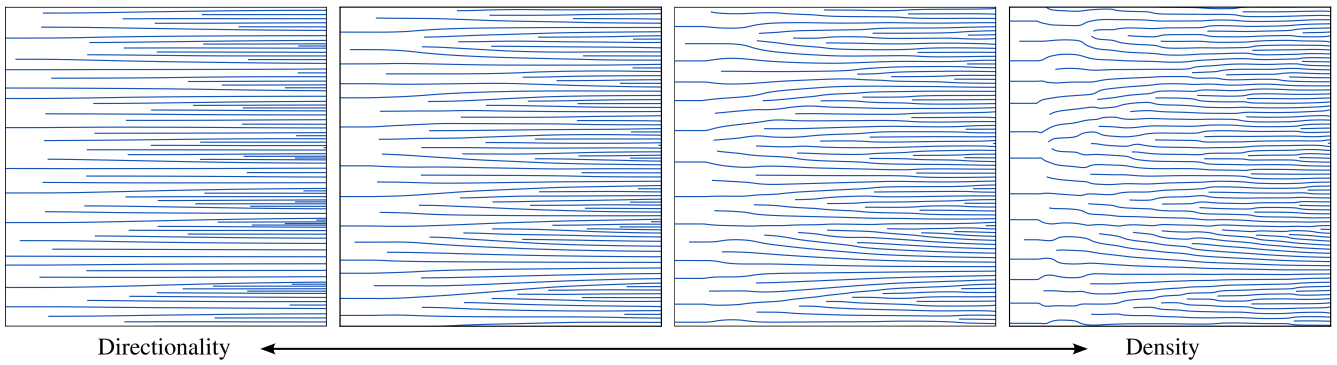


Figure 7: *Our user interface enables the users to interactively explore how the weights change the look of the results via a single slider. It allows artistic control of directionality and density. Four sets of streamlines generated from a horizontal direction field and a linear density field after the regularization step are shown, with different w_{den}, from left to right,* $10^1, 10^2, 10^3$, *and* 10^4.

$w_{\text{den}}, w_{\text{dir}}, w_{\text{stretch}}$ and w_{reg} are $10^0, 10^{-1}, 10^2, 10^{-8}$, respectively. In our design system, the user can interactively change the weights and observe the effect on the streamline pattern to achieve the desired balance between faithfully reproducing density and directionality, as shown in Figure 7. What matters the most is the ratio between w_{den} and w_{dir} as they are used to control the density–directionality tradeoff. The other two weights alleviate undesired artifacts such as the shifting of the whole pattern, or the stretching/shifting of individual streamline, as shown in Figure 9.

3.5. Continuous Stitching Path Generation

The last step in generating a stitching path is to join the set of disjoint streamlines into one connected component. The reason for this is that, during the physical embroidery process, the thread needs to be cut at the end of every stitching path, and several additional stitches need to be made to prevent loose ends. This process tends to warp the underlying piece of cloth and decreases the quality of the embroidery drastically. To avoid this issue, we connect the set of streamlines into one continuous path, which minimizes the number of starts and stops that the embroidery machine must perform.

This entails adding additional edges connecting neighboring streamlines. Inherently, these edges will run counter to the direction field $\mathbf{v}$, so we want to minimize their visibility by preferring short connecting segments in high-density regions. The problem of choosing these edges can naturally be formulated as a minimum spanning tree (MST) problem on the edges of the Delaunay triangulation constructed during the previous step. We define the cost w associated with adding an edge as

$$w(\mathbf{p}_i^k, \mathbf{p}_j^l) = \frac{\|\mathbf{p}_j^l - \mathbf{p}_i^k\|}{\alpha(\mathbf{m}_{i,j}^{k,l})},$$

where $\mathbf{m}_{i,j}^{k,l}$ is defined as the midpoint of the edge $(\mathbf{p}_i^k, \mathbf{p}_j^l)$.

Finally, we convert the MST into a single continuous stitching path by traversing it in depth-first order. We add a stitch every time an edge is traversed forward or backward. This effectively doubles all edges of the tree, which is similar to patterns from commercial embroidery software. This results in a stitching pattern ready for fabrication.

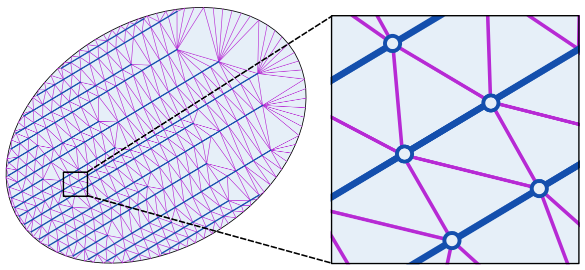


Figure 8: *We perform a Delaunay triangulation that enforces all streamline edges (blue) to be part of the triangulation. The additional edges (purple) connect the vertices of neighboring streamlines.*

4. Results, Comparison and Discussions

In this section, we demonstrate the capabilities of our streamline generation. We first compare our method to state-of-the-art algorithms for streamline generation. Next, we present a user interface to convert input images into regions with density and directionality fields. Finally, we use our user interface to design and fabricate several examples of embroidery designs.

4.1. Alternative Streamline Algorithms

In Figure 10, we compare our method with the state-of-the-art approaches for generating streamline patterns. We use three varying density and direction fields, please see Figure 10 (top). We observe that our method spawns new streamlines as the density increases, and terminates streamlines as density decreases. The streamlines are also distributed such that they remain well-aligned with the prescribed direction field.

The method of [KCPS15] runs at an interactive speed and gives a good match for directionality and density in regions where the density gradient is orthogonal to the direction field. In other regions,

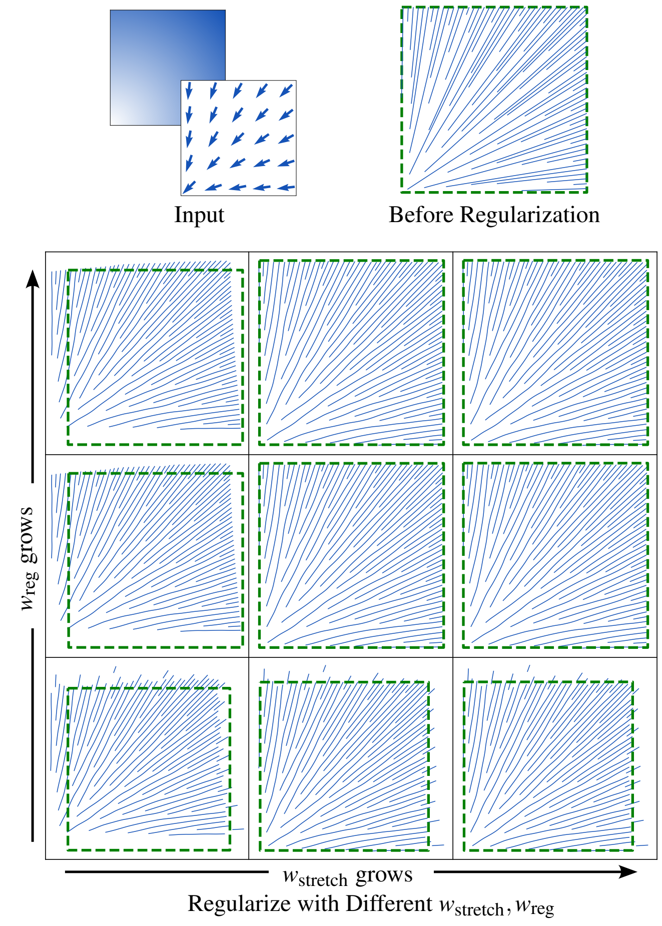

Input
Before Regularization
w_{reg} grows
w_{stretch} grows
Regularize with Different $w_{\text{stretch}}, w_{\text{reg}}$

Figure 9: *From the given input fields, we generate the streamlines and regularize them with different w_{reg} and $w_{stretch}$. The center cell in the 3 × 3 matrix uses the default weights. The green dashed line squares mark the bounding box of the streamlines before regularization.*

the method generates bifurcations that significantly violate the directionality requirement. In contrast, the method of [TTZ*20] runs in real-time, producing streamlines that are well-aligned with the input fields globally but have large local errors, both in direction and density. This is most visible near streamline endpoints, where the curves form hook-like artifacts.

4.2. Commercial Embroidery Software

We also compare our method to the commercial software *Bernina DesignerPlus* [Ber21] by manufacturing a flower petal example. Our method (Figure 11, center) results in a stitching pattern that is aligned with the direction field derived from the brush strokes in the input image. The spacing of the threads produces an intensity variation through half-toning, which results in a red-to-pink gradient that matches the input image when viewed from a distance. The *AutoDigitize* function from the commercial software (Figure 11, right) cannot reproduce the color gradient because it only separates the input image into uniformly colored regions. Furthermore, the

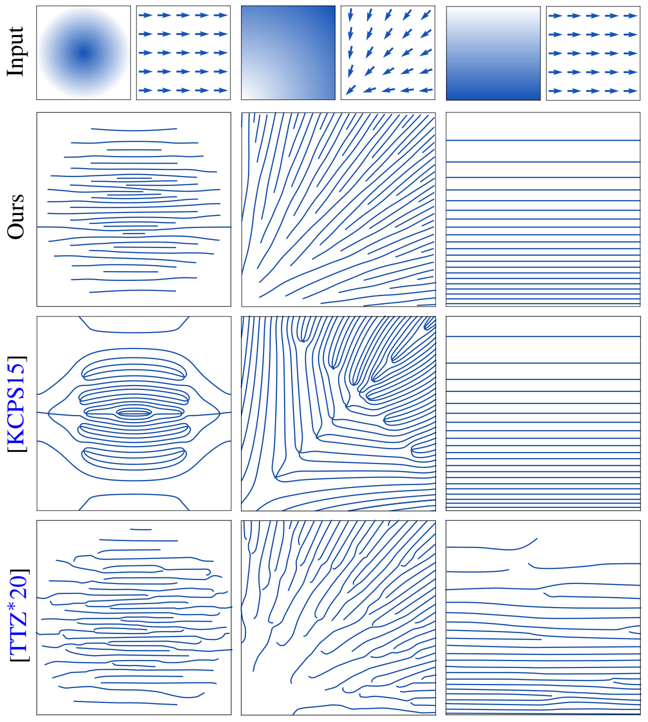


Figure 10: *We compare our method on a set of analytical input fields with the alternative pattern generation approaches of [KCPS15] and [TTZ*20].*

stitching direction is derived from the shape of the boundary of each region, so it cannot generally be made to align with the direction of the brush strokes in the input image.

4.3. User Interface

Regions and Direction Field. We provide a user interface to supplement our method in creating complex designs. The input to our method is the boundary of a region, a direction field, and a density, as defined in Section 3. For the boundary and direction field, we adopt the image annotation tool Labelme [Wad21]. First, the user draws a polygonal boundary to select a region in the image. Then,

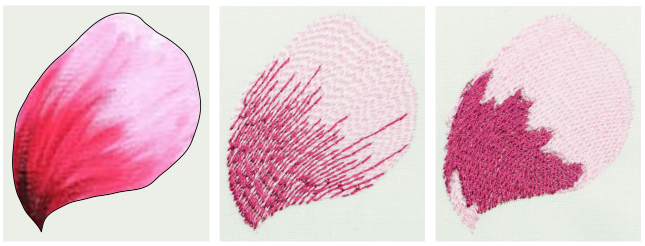


Figure 11: *We use the same image (left) as input to compare our method (middle) with the AutoDigitize function in Bernina DesignerPlus (right) by fabricating patterns.*

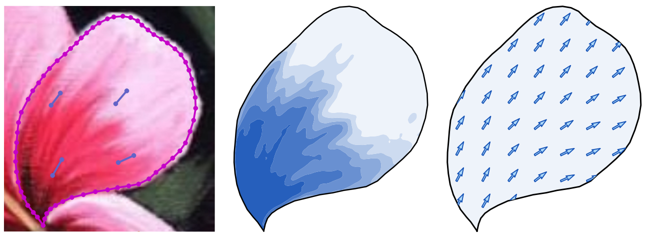


Figure 12: ***Left***: *Boundary and direction annotation via Lableme.* ***Middle***: *Density function* α *extracted from the input, darker color indicates higher density.* ***Right***: *Direction function v obtained by interpolating direction annotation.*

to define the direction field, the user draws a number of line segments within the region (Figure 12, left), which are automatically interpolated to define a continuous direction field (Figure 12, right).

Colors and Density. The density information is extracted automatically from each selected region Ω. To ensure that the color information is smooth, we provide the option to denoise the image using a bilateral filter [TM98]. To reduce the number of threads used and extract the density function, we determine the two colors s_1, s_2 most suitable to represent all colors in Ω as a convex combination.

To best preserve color information, we run a principal component analysis (PCA) on the image in CMY color space and take the first principal axis. Along this axis, we get a line of colors in CMY space, onto which we can project the color of every pixel with minimal loss. We remove the outliers of these projected points and then use the pair of the remaining points furthest to either side of the line as the colors for the region. From the pair, we pick the one with a larger density as the background color s_1, and the other as the foreground s_2. Then, all the projected colors in between can be represented as a convex combination $(1-t)s_1 + ts_2$, with $t \in [0,1]$.

The coefficient t at every point can then be converted into a density field α, as follows: We use s_1 as the background color to uniformly fill the region first, and then stitch s_2 on top to create a color gradient. Assuming the s_2 thread has a width of $b > 0$, and there is a distance of $1/\alpha$ between parallel stitch lines of s_2, the visible portion of s_1 has a width of $1/\alpha - b$. The blend coefficient is then given by $t = \frac{b}{(1/\alpha - b)+b}$, so we need to set $\alpha = \frac{t}{b}$.

Thread Colors. Finally, we map the proposed colors to the color gamut of our embroidery machine. Because the thread selection is limited, we seek to find colors that match the originals while providing good relative contrast. We achieve this goal by formulating a loss function $E : \mathcal{T}^2 \to \mathbb{R}$ for the possible choices of the color pairs,

$$E(t_1,t_2) = \mathrm{dE}(s_1,t_1) + \mathrm{dE}(s_2,t_2) + w \max\{C(s_1,s_2) - C(t_1,t_2), 0\},$$

where s_1, s_2 are the PCA colors of the region, and $t_1, t_2 \in \mathcal{T}$ are the physical thread colors from the set of all available threads $\mathcal{T}$. Furthermore, dE is the Delta E2000 distance [The01], a perceptual color distance measure, and C is the relative contrast as specified by the W3C consortium [W3C16]. We use a weight $w = 10$ to bring the relative contrast into the range of typical magnitudes of color distances. Finally, we exhaustively search for the pair of colors t_1, t_2 that minimizes E.

4.4. Embroidery Prototypes

We fabricated several stitching paths generated by our algorithm using Bernina B590, an embroidery machine that supports automatic embroidery. As a substrate, we used an ivory embroidery cloth and colored polyester threads for stitching. The input images, their corresponding segmentation, generated paths, and fabricated results are shown in Figure 14.

The first example is a stylized FEATHER. We reproduce the feather in two colors: pink as a background and white as a foreground, which is introduced for adding structure. Thanks to the alignment of the direction field with the barbs of the feather, the white thread achieves a believable reproduction of the feathery texture. Our next example is a classic for embroidery, a CHERRY, which we reproduce with a radial direction field. Even in this challenging scenario, our method can reconstruct the highlights from the original image, creating the illusion of spherical cherries. We significantly increase the complexity of the input image with the PHOENIX example. It combines a complex direction field to match the orientation of the individual tongues of fire with gradients to render the body and wings.

For our next set of examples, we use the full images of the SUNSET, AURORA, and MOUNTAIN scenes. The SUNSET demonstrates smooth transitions between different shades of color while preserving the directionality in the water reflection. The AURORA example features complicated paths as it traverses the sky. Finally, the MOUNTAIN example includes foliage and clouds. In all of these images, our method captures the original appearance while highlighting the finer features through the direction field. The threads of each embroidered sample are well-aligned and fill up the entire canvas, leaving no gaps between the segmented regions. Moreover, the cloth substrate only undergoes minimal stretching and warping during embroidery, because we layer at most two stitches on top of each other.

4.5. Timings

To produce the embroidery pattern for a multi-segment image, our pipeline needs to run per segment. However, the computation of each segment is independent and, thus, can be performed in parallel. Therefore, we will discuss the timing for generating an embroidery pattern for a single segment.

We prototype our method in Python, using SciPy for the k-d tree, and Triangle package [She96] for computing constrained Delaunay triangulation. To finish the pipeline shown in Figure 2 and generate the flower petal stitching pattern, it takes 1.58 seconds, of which 0.03 seconds is used for sampling the sources and sinks; 0.79 seconds for tracing streamlines from the sources; 0.01 seconds for matching the sinks and streamlines for cutting; 0.66 seconds for constructing the sparse matrix for the regularization step; 0.02 seconds for linear solve; and 0.07 seconds for connecting streamlines via MST.

We also analyze how our method scales. For given input density and directionality fields, the time consumed for processing a segment depends on the thread width b (Section 4.2). The smaller b is, the more threads (streamlines) are needed to fill the same regions. Varying b for the petal example, we plot the timings of different steps in our pipeline against the number of streamlines in Figure 13. As the problem scales up, the most significant factors of runtime are the streamline tracing and the assembly of the matrix for regularization. These two steps can be accelerated via a parallelized C++ implementation.

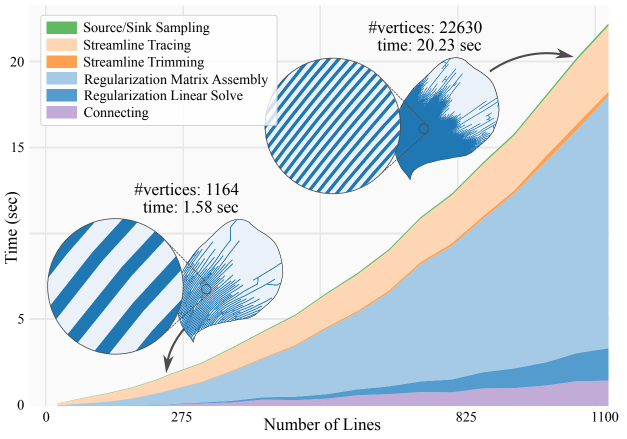


Figure 13: *The runtime of the flower petal example. Varying thread density leads to different numbers of streamlines. As the thread width decreases, more streamlines are needed to approximate the same density field, and hence, the number of streamlines and vertices on the streamlines increases. As the problem scales up, the two most important factors are assembly matrices for regularization and the streamline tracing.*

4.6. Limitations and Future Work

We propose an algorithm that facilitates the generation of embroidery designs with prescribed directionality. However, the anisotropy of the thread appearance could be leveraged not only to convey direction but also to simulate textures. An exciting direction for future work is to include various stitching patterns to enhance the high-frequency feature of the final pattern.

The direction field in our method is currently limited to be continuous and defined on a simply-connected domain. Although this limitation still provides a wide range of inputs as demonstrated by our fabricated samples, direction fields with singularities are not supported at the moment. Likewise, non-simply-connected domains have to be split into simply-connected ones. A promising direction of future work would be to extend the method to handle even these challenging direction fields.

We present a method for generating stitchable embroidery designs from direction and density fields. To provide these inputs, we propose a combination of user interaction and algorithmic processing that splits an input image into regions formed by gradients of two colors each. A possible direction of future work is to generalize the method to consider color overlaps and blending similarly to vector drawing [FLB17]. This would enable processing regions with more colors to decrease the total number of regions.

During the stitching process, the cloth may locally bend or shrink depending on the location of introduced threads. In our samples, this effect is sufficiently small to be ignored. However, in certain cases, it can cause significant distortion of the cloth, making future stitches deviate from their intended location. An interesting avenue for future work would be to explore how the warping of the cloth depends on the directionality and density of the stitched pattern, and to minimize this effect.

5. Conclusion

We present an algorithm for generating stitchable embroidery designs that satisfy the fabrication constraints of an embroidery machine, and approximate the color gradients in an input image. To achieve this goal, we formulate the search for patterns as a streamline generation problem with a density requirement, based on a novel method for sampling spawn and terminal points of streamlines. We also provide a user interface that enables the artist to explore a range of designs that emphasize a regular spacing of stitch lines or strict adherence to the desired direction field. In the last step, we join streamlines into one connected component and convert it into a single continuous path, so that it can be stitched without cutting the thread, thus resulting in a clean design.

We compare our streamline generation algorithm to state-of-the-art methods and find that it produces more regular results for difficult inputs. Furthermore, we fabricate several physical embroidery samples that were designed using our algorithm and user interface.

Acknowledgment

This work was supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 715767 – MATERIALIZABLE) and FWF Lise Meitner (Grant M 3319). We thank the anonymous reviewers for their insightful feedback; Solal Pirelli, Shardul Chiplunkar, and Paola Mejia for proofreading; everyone in the visual computing group at ISTA for inspiring lunch and coffee breaks; Thibault Tricard for helping producing the results of [TTZ*20] in Figure 10. Special thanks to Tatyana Ozerova for the permission to use her work for AURORA. The LANDSCAPE and PHOENIX images were generated by Dall-E [RDN*22]. SUNSET image is from https://www.freepik.com/. Open access funding provided by Ecole Polytechnique Federale de Lausanne.

References

[Ahm14] AHMED A. G.: Modular line-based halftoning via recursive division. In *Proceedings of the Workshop on Non-Photorealistic Animation and Rendering* (2014), pp. 41–48. doi:10.1145/2630397.2630403.

[Ahm15] AHMED A. G.: From stippling to scribbling. In *Proceedings of Bridges 2015: Mathematics, Music, Art, Architecture, Culture* (2015), pp. 267–274. URL: http://archive.bridgesmathart.org/2015/bridges2015-267.html.

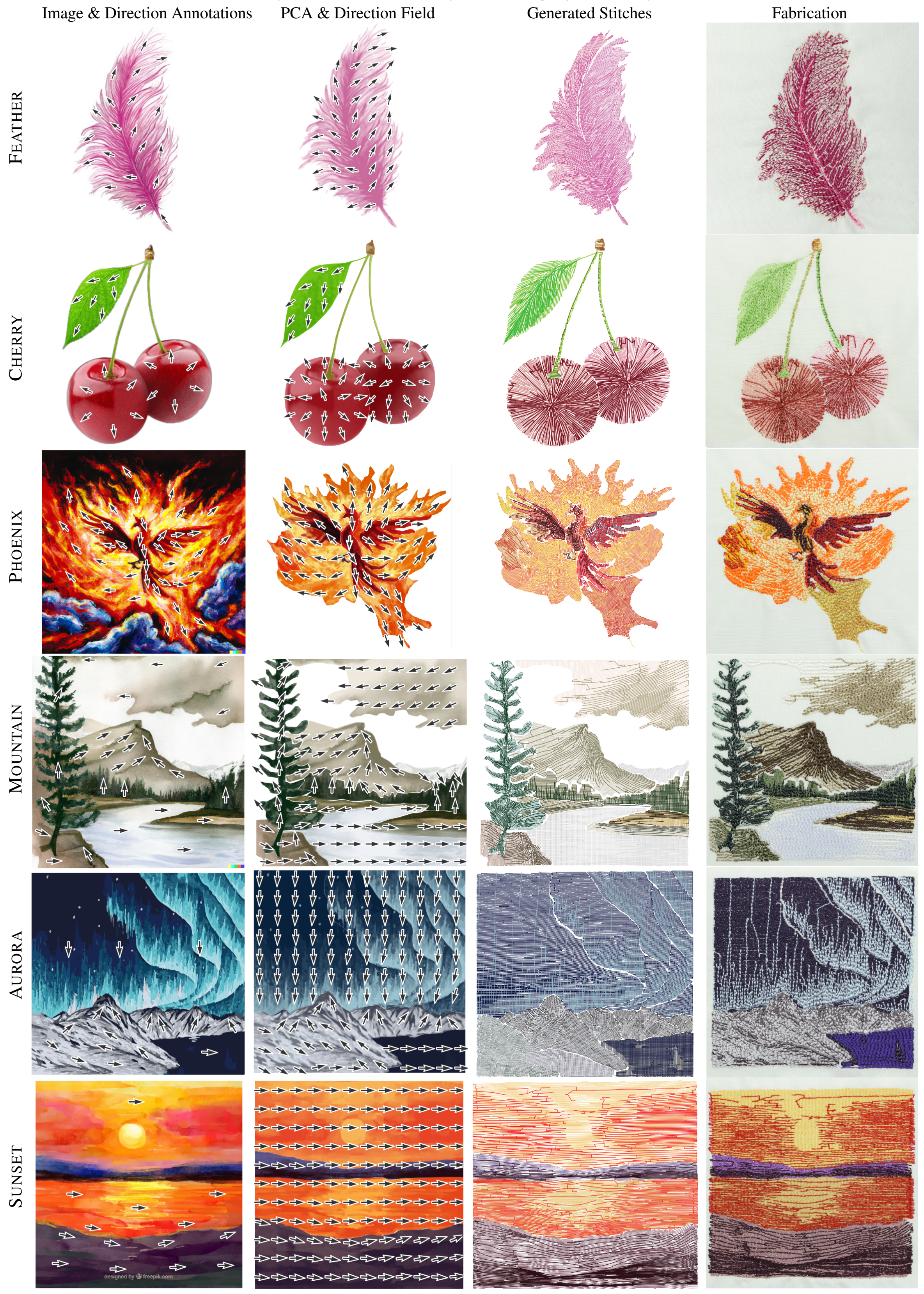


Figure 14: *Different embroidery patterns produced by our method. We show the original image with direction annotation, the segmented PCA image superposed by the direction field, the generated stitching pattern, and the photos of the patterns sewed by an embroidery machine.*

[BCOM*22] BEDEL A., COUDERT-OSMONT Y., MARTÍNEZ J., NISHAT R. I., WHITESIDES S., LEFEBVRE S.: Closed space-filling curves with controlled orientation for 3D printing. *Computer Graphics Forum* (2022), 473–492. doi:10.1111/cgf.14488.

[Ber21] BERNINA INTERNATIONAL AG: Bernina embroidery software 9, 2021.

[CDS10] CRANE K., DESBRUN M., SCHRÖDER P.: Trivial Connections on Discrete Surfaces. *Computer Graphics Forum* (2010), 1525–1533. doi:10.1111/j.1467-8659.2010.01761.x.

[CMKM12] CHEN X., MCCOOL M., KITAMOTO A., MANN S.: Embroidery modeling and rendering. In *Proceedings of Graphics Interface 2012* (CAN, May 2012), Canadian Information Processing Society, pp. 131–139. doi:10.5555/2305276.2305299.

[CPG15] CARLSON C., PALEY N., GRAY T.: Algorithmic quilting. In *Proceedings of Bridges 2015: Mathematics, Music, Art, Architecture, Culture* (2015), pp. 231–238. URL: http://archive.bridgesmathart.org/2015/bridges2015-231.html.

[Cro16] CROUSE D. F.: On implementing 2d rectangular assignment algorithms. *IEEE Transactions on Aerospace and Electronic Systems 52*, 4 (Aug 2016), 1679–1696. doi:10.1109/TAES.2016.140952.

[CSZ17] CUI D., SHENG Y., ZHANG G.: Image-based embroidery modeling and rendering. *Computer Animation and Virtual Worlds 28*, 2 (2017), e1725. doi:10.1002/cav.1725.

[DGK07] DATTA-GUPTA A., KING M. J.: Streamline simulation: theory and practice. doi:10.2118/9781555631116.

[FLB17] FAVREAU J.-D., LAFARGE F., BOUSSEAU A.: Photo2clipart: Image abstraction and vectorization using layered linear gradients. *ACM Transactions on Graphics (TOG) 36*, 6 (2017), 1–11. doi:10.1145/3130800.3130888.

[GLL*21] GUAN X., LUO L., LI H., WANG H., LIU C., WANG S., JIN X.: Automatic embroidery texture synthesis for garment design and online display. *The Visual Computer* (Sept. 2021), 2553–2565. doi:10.1007/s00371-021-02216-0.

[HFL11] HUANG H., FU T.-N., LI C.-F.: Painterly rendering with content-dependent natural paint strokes. *The Visual Computer* (Sept. 2011), 861–871. doi:10.1007/s00371-011-0596-5.

[HWYZ20] HSU C.-Y., WEI L.-Y., YOU L., ZHANG J. J.: Autocomplete element fields. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems* (2020), pp. 1–13. doi:10.1145/3313831.3376248.

[IU09] INOUE K., URAHAMA K.: Chaos and graphics: Halftoning with minimum spanning trees and its application to maze-like images. *Comput. Graph. 33*, 5 (oct 2009). doi:10.1016/j.cag.2008.09.015.

[JL97] JOBARD B., LEFER W.: Creating Evenly-Spaced Streamlines of Arbitrary Density. In *Visualization in Scientific Computing '97* (Vienna, 1997), Springer, pp. 43–55. doi:10.1007/978-3-7091-6876-9_5.

[KCPS13] KNÖPPEL F., CRANE K., PINKALL U., SCHRÖDER P.: Globally optimal direction fields. *ACM Transactions on Graphics* (July 2013), 59:1–59:10. doi:10.1145/2461912.2462005.

[KCPS15] KNÖPPEL F., CRANE K., PINKALL U., SCHRÖDER P.: Stripe patterns on surfaces. *ACM Transactions on Graphics* (July 2015), 39:1–39:11. doi:10.1145/2767000.

[KLC07] KANG H., LEE S., CHUI C. K.: Coherent line drawing. In *Proceedings of the 5th international symposium on Non-photorealistic animation and rendering* (New York, NY, USA, Aug. 2007), Association for Computing Machinery, pp. 43–50. doi:10.1145/1274871.1274878.

[KSZ18] KONG Q., SHENG Y., ZHANG G.: Hybrid noise for LIC-based pencil hatching simulation. In *2018 IEEE International Conference on Multimedia and Expo (ICME)* (2018), IEEE, pp. 1–6. doi:10.1109/ICME.2018.8486527.

[LFH*19] LI Y., FANG C., HERTZMANN A., SHECHTMAN E., YANG M.-H.: Im2pencil: Controllable pencil illustration from photographs. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2019), pp. 1525–1534. doi:10.1109/CVPR.2019.00162.

[LH06] LEFEBVRE S., HOPPE H.: Appearance-space texture synthesis. *ACM Transactions on Graphics (TOG) 25*, 3 (2006), 541–548. doi:10.1145/1141911.1141921.

[LHM17] LIU C., HODGINS J., MCCANN J.: Whole-cloth quilting patterns from photographs. In *Proceedings of the Symposium on Non-Photorealistic Animation and Rendering* (New York, NY, USA, July 2017), Association for Computing Machinery, pp. 1–8. doi:10.1145/3092919.3092925.

[LLD10] LAGAE A., LEFEBVRE S., DUTRÉ P.: Improving gabor noise. *IEEE Transactions on Visualization and Computer Graphics 17*, 8 (2010), 1096–1107. doi:10.1109/TVCG.2010.238.

[LM14] LI H., MOULD D.: Continuous line drawings and designs. *International Journal of Creative Interfaces and Computer Graphics (IJCICG) 5*, 2 (2014), 16–39.

[LS20] LIU S., SONG H.: Flow visualization with density control. In *Computer Graphics International Conference* (2020), pp. 301–312. doi:10.1007/978-3-030-61864-3_26.

[LWSF10] LI H., WEI L.-Y., SANDER P. V., FU C.-W.: Anisotropic Blue Noise Sampling. doi:10.1145/1882261.1866189.

[LXJ12] LU C., XU L., JIA J.: Combining sketch and tone for pencil drawing production. In *Proceedings of the Symposium on Non-Photorealistic Animation and Rendering* (June 2012), Eurographics Association, pp. 65–73. doi:10.5555/2330147.2330161.

[MAD05] MEBARKI A., ALLIEZ P., DEVILLERS O.: Farthest point seeding for efficient placement of streamlines. In *VIS 05. IEEE Visualization, 2005.* (Oct. 2005), pp. 479–486. doi:10.1109/VISUAL.2005.1532832.

[MARI17] MARTÍN D., ARROYO G., RODRÍGUEZ A., ISENBERG T.: A survey of digital stippling. *Computers & Graphics 67* (2017), 24–44. doi:10.1016/j.cag.2017.05.001.

[MS21] MA C., SUN Z.: Multilayered stitch generating for random-needle embroidery. *The Visual Computer* (June 2021). doi:10.1007/s00371-021-02195-2.

[Nee18] NEEDLEWORK R.: *The Royal School of Needlework Book of Embroidery: A Guide to Essential Stitches, Techniques and Projects*. Search Press, 2018.

[QCX*20] QIAN W., CAO J., XU D., NIE R., GUAN Z., ZHENG R.: Cnn-based embroidery style rendering. *International Journal of Pattern Recognition and Artificial Intelligence 34*, 14 (2020), 2059045. doi:10.1142/S0218001420590454.

[QXC*19] QIAN W., XU D., CAO J., GUAN Z., PU Y.: Aesthetic art simulation for embroidery style. *Multimedia Tools and Applications 78*, 1 (2019), 995–1016. doi:10.1007/s11042-018-6002-9.

[RDN*22] RAMESH A., DHARIWAL P., NICHOL A., CHU C., CHEN M.: Hierarchical Text-Conditional Image Generation with CLIP Latents, Apr. 2022. doi:10.48550/arXiv.2204.06125.

[Sec02] SECORD A.: Weighted voronoi stippling. In *Proceedings of the 2nd international symposium on Non-photorealistic animation and rendering* (2002), pp. 37–43. doi:10.1145/508530.508537.

[She96] SHEWCHUK J. R.: Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. In *Applied Computational Geometry: Towards Geometric Engineering*, Lin M. C., Manocha D., (Eds.), vol. 1148 of *Lecture Notes in Computer Science*. Springer-Verlag, May 1996, pp. 203–222. From the First ACM Workshop on Applied Computational Geometry. doi:10.1007/BFb0014497.

[SLCZ09] SPENCER B., LARAMEE R. S., CHEN G., ZHANG E.: Evenly spaced streamlines for surfaces: An image-based approach. In *Computer Graphics Forum* (2009), Wiley Online Library. doi:10.1111/j.1467-8659.2009.01352.x.

[TB96] TURK G., BANKS D.: Image-guided streamline placement. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques* (New York, NY, USA, Aug. 1996), Association for Computing Machinery, pp. 453–460. doi:10.1145/237170.237285.

[TEZ*19] TRICARD T., EFREMOV S., ZANNI C., NEYRET F., MARTÍNEZ J., LEFEBVRE S.: Procedural phasor noise. *ACM Transactions on Graphics* (July 2019), 57:1–57:13. doi:10.1145/3306346.3322990.

[The01] THE INTERNATIONAL COMMISION ON ILLUMINATION (CIE): *Improvement to industrial colour-difference evaluation*. 2001.

[TM98] TOMASI C., MANDUCHI R.: Bilateral filtering for gray and color images. In *Sixth International Conference on Computer Vision (IEEE Cat. No.98CH36271)* (Jan. 1998), pp. 839–846. doi:10.1109/ICCV.1998.710815.

[TTZ*20] TRICARD T., TAVERNIER V., ZANNI C., MARTÍNEZ J., HUGRON P.-A., NEYRET F., LEFEBVRE S.: Freely orientable microstructures for designing deformable 3D prints. *ACM Transactions on Graphics* (Nov. 2020), 211:1–211:16. doi:10.1145/3414685.3417790.

[TWY*20] TU P., WEI L.-Y., YATANI K., IGARASHI T., ZWICKER M.: Continuous curve textures. *ACM Transactions on Graphics (TOG) 39*, 6 (2020), 1–16. doi:10.1145/3414685.3417780.

[W3C16] W3C WORLD WIDE WEB CONSORTIUM: WCAG2.0 HTML Techniques, Oct. 2016.

[Wad21] WADA K.: Labelme: Image Polygonal Annotation with Python, 2021. doi:10.5281/zenodo.5711226.

[WS94] WINKENBACH G., SALESIN D. H.: Computer-generated pen-and-ink illustration. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques* (1994), pp. 91–100. doi:10.1145/192161.192184.

[WT13] WONG F. J., TAKAHASHI S.: Abstracting images into continuous-line artistic styles. *The Visual Computer* (June 2013), 729–738. doi:10.1007/s00371-013-0809-1.

[YSMY16] YANG K., SUN Z., MA C., YANG W.: Paint with Stitches: A Random-needle Embroidery Rendering Method. In *Proceedings of the 33rd Computer Graphics International* (New York, NY, USA, June 2016), Association for Computing Machinery, pp. 9–12. doi:10.1145/2949035.2949038.

[ZISS04] ZANDER J., ISENBERG T., SCHLECHTWEG S., STROTHOTTE T.: High quality hatching. In *Computer Graphics Forum* (2004), vol. 23, Wiley Online Library, pp. 421–430. doi:10.1111/j.1467-8659.2004.00773.x.