Algorithmic Quilting Pattern Generation for Pieced Quilts

Yifei Li* Carnegie Mellon University
David E. Breen† Drexel University
James McCann‡ Carnegie Mellon University
Jessica Hodgins§ Carnegie Mellon University

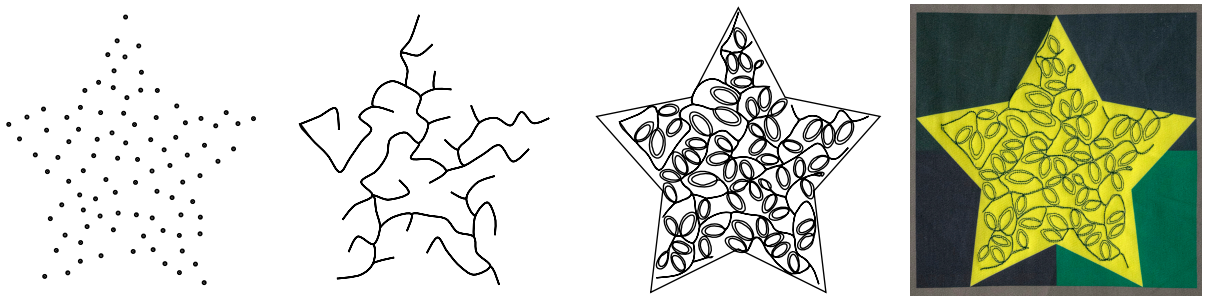

Figure 1: Quilting a star with a vine pattern by starting with randomly distributed points, generating a spanning tree, and adding non-overlapping decorations. A photo of the quilted pattern is shown to the right.

ABSTRACT

Free-motion quilting patterns are functional and decorative patterns sewn on pieced quilts using a single-line continuous stitch path for each region of the quilt. Seven families of quilting patterns are commonly used by quilters [3]. We present an approach for computationally generating three of these families. The user can control the design for each family based on a set of parameters, including the density and general layout of the pattern as well as the decorative elements. Our algorithm starts by sampling a point set in a designated region, generates a skeleton path over that set, then inserts decorative elements along the skeleton. We provide digital and quilted examples for each type of pattern.

Keywords: quilting, pattern generation, spanning tree, texture generation, ornamentation, line drawing, continuous line drawing.

Index Terms: Computing methodologies—Computer graphics—Rendering—Non-photorealistic rendering;

Figure 2: Cameli's categorization from the book *Step-by-Step Free-Motion Quilting* [3] from C&T Publishing. From left to right, top to bottom: Branching, Nestled (Shape Packing), Wanderer, Back-and-forth Echoing, Climber, Emerging, Edge-to-edge. In this paper, we develop algorithms for generating patterns in the Branching, Shape Packing and Wanderer categories.

1 INTRODUCTION

A quilt is a textile composed of three layers of fabric: a cloth top, batting, and a woven cloth back. Two key components define the aesthetics of a quilt: the piecing and the quilting. In most quilts, the top layer of fabric is *pieced* together from small pieces to create a pattern. The process of quilting involves sewing together the three layers of the quilt so that they do not slide with respect to one another, even when the quilt is washed. Quilters design the quilting pattern so that it complements the pieced design.

Though quilting has traditionally been done by hand, machine quilting using long-arm quilting machines has become more common. Such machines are capable of quilting over an entire quilt top using manual or, in some cases, computer control. Computer control requires the quilter to input a 2D digital pattern of stitching paths. The design of such a pattern has several constraints. First, the stitch pattern should consist of continuous paths, because starting and ending each path on a computer-controlled quilting machine requires work from the quilter. Ideally, the pattern should consist of a *unicursal*, or single-line continuous path for at least large regions, if not the entire quilt. Second, because the stitches must be dense enough to hold the three layers of the quilt together, the stitch lines should be distributed roughly uniformly over the quilt and with a minimum density. Third, the design should respect the aesthetics of the pieced quilt.

An important category of quilting techniques is free-motion quilting (FMQ). In FMQ, the quilter can freely move the fabric by hand under the stitching head to create designs with curves. Quilters have come up with common FMQ design themes that make use of primitives, a flower for instance, and repeat it over one clearly-defined region, usually a quilt block. These designs often incorporate randomness to avoid displeasing repetition when adapting the pattern to a specific region of the quilt. The irregularity of the resulting pattern is part of the appeal of FMQ. Christina Cameli proposed an empirical categorization of common FMQ patterns in her book *Step-by-Step Free-Motion Quilting* [3]. Cameli proposed seven forms of FMQ: Nestled (Shape Packing), Branching, Edge-to-Edge, Emerging, Back-and-Forth Echoes, Wanderers and Climbers (Figure 2). Each of these categories combine simple shapes using a different set of rules.

In this paper, we describe a system that generates a stitch path

*e-mail: yifeil1@cs.cmu.edu
†e-mail: david@cs.drexel.edu
‡e-mail: jmccann@cs.cmu.edu
§e-mail: jkh@cs.cmu.edu

for pieced quilts from a small number of user-controlled parameters (Figure 3). We propose a framework that combines a spanning tree or a TSP tour with decorative elements to generate families of FMQ designs. Our system supports Shape Packing, Branching and Wanderer categories with automatic algorithms controlled by the user via design elements and a set of intuitive parameters, such as pattern density and design element scale.

This paper provides a starting point for semi-automatically generating free-motion quilting patterns using algorithms that have been explored for other applications in the graphics community. Quilting patterns are a special category of 2D patterns that ideally need to be traceable in one path, as well as dense enough to hold three layers of fabric together. The user can control the aesthetics of patterns by defining the decorative elements and intuitive parameters such as pattern density and the scale of the decorative elements. Quilters can benefit from free-motion quilting patterns generated algorithmically by saving time on designing the pattern, and people less skillful in quilting will be able to produce interesting quilts with the aid of our system.

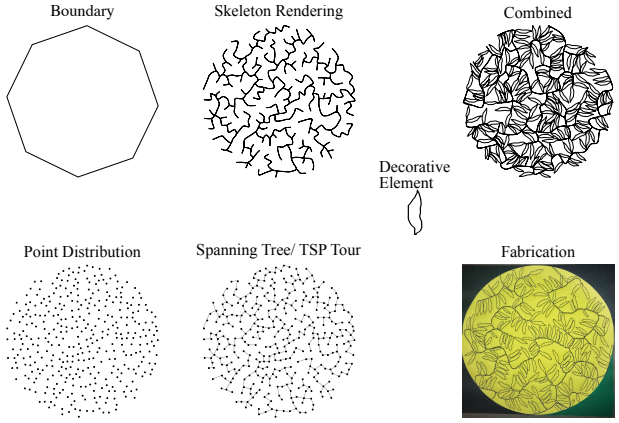

Figure 3: Pipeline. Given a region boundary and a decorative element, the algorithm generates a skeleton path and places the decorative element following the guidelines of user-defined parameters.

2 Background

Researchers in the field of computer graphics have explored many pattern generation systems. Wong et al. [20] described a method to automatically generate floral ornament designs, many of which resemble the output of the quilting algorithms presented here, particularly the vine-like patterns. Although the basic idea of growing ornaments along a path is similar to the Branching algorithm in this paper, ornament design does not have the constraint that the generated paths need to be traceable in a course with few interruptions. Anderson and Wood [1], Mech and Miller [16], and Lu et al. [15] presented applications that utilize a user-defined curve to help generate 2D ornaments that follow the design principles of repetition, balance, growth, and geometric constraints. Defining such curves on the scale of a quilt manually would be quite time-consuming.

Kaplan and Bosch [10] presented a technique for constructing a continuous line drawing based on a user-supplied image by first generating a point set, then solving the Traveling Salesman Problem (TSP) over the points to obtain a unicursal path, while Li and Mould [13] used a similar framework but instead employed a tree-based technique to obtain the path for the same application. TSP tours and tree-based paths are different in that the former does not revisit points, while the latter visits each node exactly twice. Both features are useful for the Wanderer and Branching algorithms presented here. Wong et al. [19] used a graph-based technique after image processing for the same application. Our application lacks an input image and therefore cannot use the same idea.

Pedersen and Singh [17] described a method to generate labyrinths and mazes from a meandering path. Kaplan and Cohen [11] proposed a method for automating the construction of Celtic knotwork. Each of these works outputs a unicursal path of a pattern category and can be directly used in free-motion quilting. However, patterns commonly used by quilters differ from these patterns in terms of aesthetics and structure.

The generation of quilting patterns has received little attention. The commercial software available to quilters (e.g., Art & Stitch [18], Creative Studio [6], and Electric Quilt [7]) provide drawing, layout and previewing capabilities, along with the ability to output the design to a variety of quilting machine file formats, but offer no generative or algorithmic methods for pattern design. Others have explored tools that assist with piecing. Igarashi et al. [9] designed an interactive tool that helps quilters to visualize pieced quilt blocks.

Carlson et al. [4] proposed a method that first generates a point set, then uses spanning trees to generate single-line filled quilting patterns. Liu et al. [14] described a method for creating a quilting pattern from a photograph. This method seeks to depict objects for a non-pieced whole-cloth quilt, by identifying lines and fill patterns that both represent the photograph and can be traced with a single stitching path.

Our framework is inspired by the work of Carlson et al. [4], Kaplan and Bosch [10], and Li and Mould [13]. Our work incorporates the idea of converting point sets to spanning trees or TSP tours in order to obtain a unicursal path. We show how varying point generation methods and using a TSP tour or spanning tree can generate a wide variety of common free-motion quilting patterns.

3 Method Overview

Given a quilting region, our method generates a quilting path consisting of a skeleton and decorative elements. Users can define the aesthetics of the pattern by controlling the density of the quilting path, the size and (in the case of Branching) the angle of the decorative element with parameters. Our pipeline, illustrated in Figure 3, consists of two phases: generation of the skeleton path and decoration of the path. The skeleton path dictates the overall layout and density of the pattern, as well as the stitching order. The decorative element is a motif designed by the user, such as a leaf or a heart, that can be inserted along the skeleton path.

To design a pattern, the user inputs a region boundary, usually representing the area of one or several quilt blocks, specifies a point distribution method and a decorative element. The decorative element will be added to the skeleton path with one of a few different methods. At each step, the user specifies parameters to control the final design.

4 Skeleton Path Generation

The first step in our algorithm generates a skeleton path inside the region boundary provided by the user. We expose parameters to the user that control the density of the path, as well as the degree of randomness. To generate the skeleton path, the algorithm first distributes points inside the region boundary using a user-selected point distribution method. A path conversion method then converts points into a unicursal path.

4.1 Point Distribution

We incorporated two point generation methods: tessellation and random point sampling (Figure 4). The resulting skeleton paths vary in regularity and each can be useful depending on the desired aesthetics and the nature of the decoration that will be applied.

The tessellation methods supported by our system include three regular tessellations: triangle, grid, hexagonal, and one semi-regular tessellation: 3.3.4.3.4 tessellation [8], which tessellates regular triangles and squares such that five regular polygons meet at every

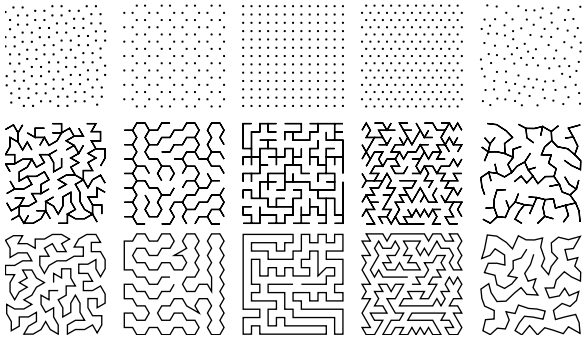

Figure 4: Each column shows a point set generated using a distribution method (top) its corresponding spanning tree (middle) and TSP tour (bottom). Point set from left to right: 3.3.4.3.4 tessellation, hexagonal tessellation, grid tessellation, triangle tessellation, Poisson-Disk sampling.

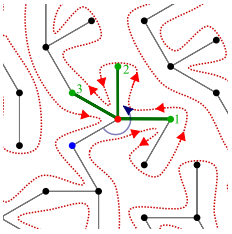

Figure 5: Children (green) of a tree node (red) are visited in CCW order, starting from the parent node (blue).

vertex: two regular triangles, followed by a square, an equilateral triangle and a square. Our system supports one random point sampling method: Poisson-Disk sampling. Poisson-Disk sampling guarantees a minimum spacing between points to allow the insertion of decorative elements. These algorithms for point distribution provided adequate variety for the patterns we generated but other methods would be easy to add.

The user specifies a point distribution distance d that determines the density of the points. For tessellations, d defines the distance between each point and its nearest neighbor. In the case of Poisson-Disk sampling, d is the minimum distance between two points.

4.2 Path Conversion

This step converts the generated point set to a stitching order for the quilting region. There are two options for path conversion: a spanning tree-based method and a TSP-based method. The former generates a path that potentially branches out at a point, while the latter generates a tour over the point set (Figure 4). Both methods generate a skeleton path that starts and ends at the same point. A post-processing step is then applied to render the skeleton path .

Spanning Tree Method The spanning tree-based method is inspired by the work of Carlson et al. [4] and the idea is also used by Li and Mould [13]. To convert the generated points into a minimum spanning tree, points should be connected to their neighbors without creating crossings. These constraints can be met by solving an Euclidean Minimum Spanning Tree problem. Our system first constructs a complete graph on all n points. The weight of each edge is the Euclidean distance between its two endpoints. Our system implements Kruskal's algorithm [12] to generate a minimum spanning tree on the graph. The spanning tree is then traversed in a depth-first order, where the children of each tree node are visited in counterclockwise order (Figure 5).

Travelling Salesman Problem Method The TSP-based method

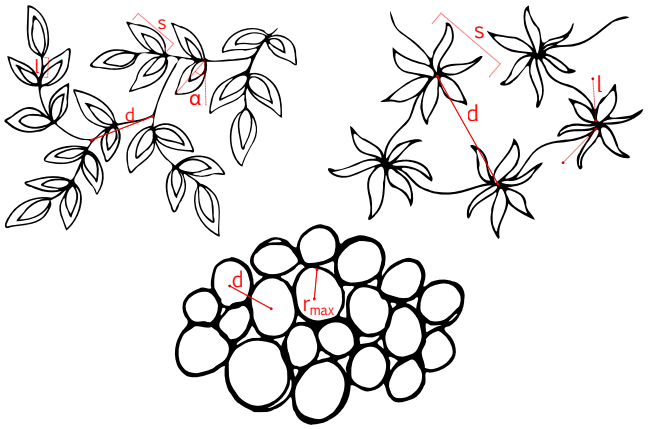

Figure 6: Parameterization of three quilting pattern families. s controls element scale. d controls point distribution distance. Patterns are illustrated by Christina Cameli [3]. Top left: Branching pattern parameters. l controls the spacing between decorative elements. α controls the angle between the decorative element and branch at the insertion point. Top right: Wanderer pattern parameters. l controls the length of the tangent of the two legs of the primitive. Bottom: Shape packing parameters. r_{seed} controls maximum pebble size.

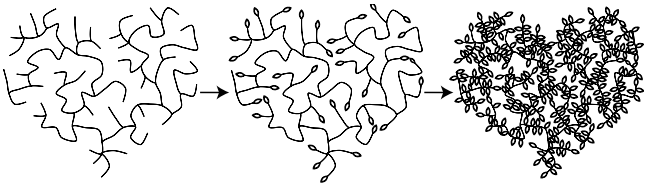

Figure 7: The skeleton path (left) of a Branching pattern is generated from a spanning tree traversal with Catmull-Rom interpolation. The algorithm then attempts to place decorative elements at the tip of each branch (middle). Branches are then traversed to place decorative elements along branches.

is inspired by the work of Kaplan and Bosch [10]. The generated points are passed to the Concorde TSP solver [2], which generates a simple tour over the point set without any crossings.

4.3 Skeleton Path Post-Processing

This step is needed for branching patterns. After the stitching order for points is determined, the skeleton path is rendered using Catmull-Rom splines [5] to interpolate between each pair of points. This process generates a vine-like path that can later be combined with decorative elements such as leaves and flowers to produce the final quilting pattern.

5 Quilting Pattern Generation

Our framework provides algorithms for generating three families of free-motion quilting patterns described by Cameli: Branching, Wanderer and Shape-Packing [3]. Branching places decorative elements along a spanning tree-based skeleton path to mimic the look of plant branches. Shape-Packing places decorative elements on the branching nodes of a spanning tree-based skeleton and grows each node until the elements are tightly packed. Wanderer places decorative elements along a TSP-based skeleton path.

5.1 Branching

Branching mimics the natural look of plants, where the decorative elements, such as leaves or flowers, are placed on alternating sides of the stitching path. We use a skeleton path generated from

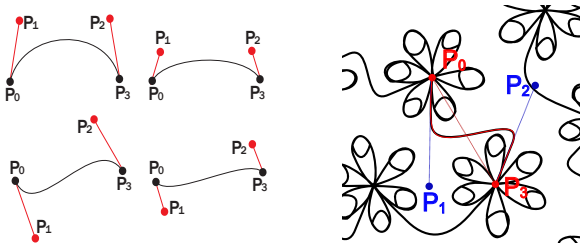

Figure 8: Left: When the control points $\mathbf{P_1}, \mathbf{P_2}$ fall in the same half space defined by the endpoints $\mathbf{P_0}$ and $\mathbf{P_3}$, the curve has one hump (top row). Otherwise the curve has two humps (bottom). Curves with longer tangents at the endpoints (left) have larger curvature. Right: The direction of the tangents $\overrightarrow{\mathbf{P_0P_1}}$ and $\overrightarrow{\mathbf{P_3P_2}}$ of the connecting curve (red) comes from the legs of the decorative element. The length of each tangent is $l \cdot ||\mathbf{P_0P_3}||$.

Poisson-Disk point distribution and Catmull-Rom interpolation for this design. We choose Poisson-Disk for point generation in order to achieve varied length in tree branches. We interpolate the branches to mimic plant stems. While the skeleton path generated from Poisson-Disk sampling fits well with natural decorative elements, such as leaves, skeleton paths generated from tessellations fits better with more geometric decorative elements.

We require the user to input an approximate collision geometry such as the convex hull for the decorative element. The user specifies four parameters (Figure 6) to customize the look of the pattern:

1. *Point distribution distance d*: controls the initial point distribution distance for the Poisson-Disk sampling, which affects the length of the branches in the generated minimum spanning tree.
2. *Decorative element size s*
3. *Gap length l*: controls the spacing between decorative elements.
4. *Angle* α: controls the direction of each decorative element relative to the direction of the skeleton path.

The algorithm for Branching (Figure 7) is presented below:
Step 1: Build convex hull for collision geometry. Scale decorative element and convex hull with input size s.
Step 2: Subdivide skeleton path with gap length l.
Step 3: Place decorative elements at the tip of each branch in the tangent direction in a depth-first traversal order. A collision detection and resolution procedure (described below) is run simultaneously. This operation is motivated by the look of real world plants.
Step 4: Traverse the tree starting from the root in depth-first order. Place decorative elements along branch segments at alternating sides, and angle the element at the user-specified initial angle α. The collision detection and resolution procedure is run simultaneously.
Collision Detection and Resolution: When attempting to place a decorative element, the convex hull of the collision geometry is tested against the convex hull of already placed elements and the region boundary, as well as the skeleton path. If collision occurs, the algorithm attempts to resolve the collision by changing the size, orientation and (optionally) placement side of the element. The element is deleted if all attempts fail to fit in the element at that location, and the algorithm moves forward to the next segment. In our implementation, an element can be scaled down to at most $0.5s$, and rotated towards the branch tip up until $\alpha = 60^\circ$. Our implementation accelerates collision detection by first using axis-aligned bounding boxes, then the full collision geometry.

5.2 Wanderer

In Wanderer, a primitive is repeated and joined by another instance through wavy lines while the skeleton path wanders through an open space. The primitives are evenly scattered in the space. These properties can be met using a TSP-based skeleton path while placing decorative elements at the center of each distributed point.

The user specifies three parameters (Figure 6): point distribution distance d, decorative element size s and tangent length l. The function of point density and decorative element size is similar to that in Branching. Tangent length controls the "smoothness" of the connecting curve. We also require the input primitive to include two legs, which are used by the algorithm to direct the direction of the connecting curves. The user should supply primitives with varying angles between the two legs.

To generate the wavy connecting lines, the algorithm connects primitives using a cubic Bézier curve. We observe that for a cubic Bézier curve defined by $\mathbf{P_0}, \mathbf{P_1}, \mathbf{P_2}, \mathbf{P_3}$, when the two control points $\mathbf{P_1}, \mathbf{P_2}$ lie in the different half spaces define by the line $\mathbf{P_0P_3}$, the connecting curve has two humps in opposite directions (Figure 8). When they lie in the same half space, the connecting curve has one hump. Further, the length of the tangents at $\mathbf{P_0}$ and $\mathbf{P_3}$ controls the smoothness of the hump. These two properties allow the system to vary the look of connecting curves.

The point stitching order is first generated using the TSP-based method via the Concorde TSP Solver [2]. The algorithm then replaces each point $\mathbf{C}$ with a primitive. The primitive's center is translated to $\mathbf{C}$, and the bisector between two legs is rotated to the direction of $\overrightarrow{\mathbf{CC_{prev}}} + \overrightarrow{\mathbf{CC_{next}}}$, where $\mathbf{C_{prev}}$ and $\mathbf{C_{next}}$ is the previous and next point in the skeleton path respectively. The primitive is selected from user inputs such that after rotation, the angle between each tangent and $\mathbf{P_0P_3}$ is closest to 60°. This rotation procedure makes sure that the connecting curve has a sufficient amount of curvature. After rotation, the starting points of the two legs are used as endpoints $\mathbf{P_0}$ and $\mathbf{P_3}$. The ending points of the two legs $\mathbf{P_1}, \mathbf{P_2}$ are scaled such that tangent length $||\mathbf{P_0P_1}||$ and $||\mathbf{P_2P_3}||$ is $l \cdot ||\mathbf{P_0P_3}||$.

5.3 Shape Packing

Shape packing attempts to pack the decorative elements as close as possible so that each element touches some of its neighbors. This packing is a popular pattern for filling the background of a quilt with packed circles, with pebbling being the simplest version. In our examples, we choose to use a tessellation method for point generation because we want the tree nodes to be uniformly distributed. We then place decorative elements at each tree node, and grow them until they touch each other.

A general form of shape packing with decorative elements can be reduced to solving pebbling first, then replacing each circle with a decorative element by using each circle as the element's minimum bounding circle. This solution only packs the bounding circle of each decorative element, not the elements themselves. Further discussion on this issue is in Section 7.

5.3.1 Pebbling

There are two goals for pebbling. First, circles should be as tightly packed as possible. Second, we want a variety of pebble sizes to give the pattern a random appearance. We employ a skeleton path produced from tessellated points for this design, and render each tree node of the resulting minimum spanning tree as one pebble. The pebbling pattern can then be traversed by using any tree traversal algorithm with one pebble being drawn as each tree node is reached.

Users specify two parameters (Figure 6): point distribution distance and seed size. Point distribution distance d is used for the tessellation point generation. In pebbling, this distance also becomes the upper limit on the size of each pebble, because the distance between the center of each tree node and its child is d. The initial size of the pebble on the root node can be controlled with a scale parameter r_{seed}, whose value should be limited to $0.5 \leq r_{seed} \leq 1$. On a skeleton path where the branches of the generated spanning tree are of equal length, for example the ones generated by tessellation methods, $r_{seed} = 0.5$ will pack pebbles of equal size before

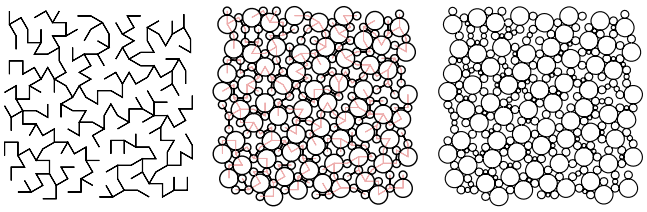

Figure 9: An example of the pattern after each step of phase 1. From left to right: underlying tree skeleton, initial placement of pebbles after step 2, short line segments replaced with small pebbles after step 3.

size optimization, whereas $r_{seed} = 0.7$ will pack pebbles with alternative sizes of $0.3d$ and $0.7d$ before size optimization. By limiting $r_{seed} \geq 0.5$, $r_{seed}d$ is the maximum radius a pebble can reach before the size optimization phase, during which the pebble can grow larger than this limit in order to be packed.

The algorithm has three phases: size determination, size optimization and rendering. The size determination phase sets an initial size for each pebble by examining its parent and avoiding collisions with nearby pebbles (Figure 9). The size optimization phase grows pebbles by allowing them to move in their neighborhood until they are tangent to at least three other pebbles (Figure 10). The rendering phase renders each pebble into a stitch path.

Let $C_i = (p_i, r_i)$ define the pebble of tree node T_i centered at point p_i with radius r_i. Let S be the set of pebbles of tree nodes that have been visited. Note that p_i is already determined by point distribution. Our algorithm attempts to maximize each r_i without collision with other pebbles:

Size determination: (Figure 9)

Step 1: Set $C_0 = (p_0, r_{seed}d)$ pebble of tree root T_0. Visit C_0.

Step 2: When visiting $C_i = (p_i, r_i)$, add C_i to visited set S. For each child T' of T_i, whose pebble is $C' = (p', r')$, set $r' = \min_{(p,r)\in S} dist(p, p') - r$. Visit C'.

Step 3: If a collision with pebbles in S ever occurs, meaning $r' < dist(p_i, p') - r$, there will be a gap between the child C' and its parent C_i. Our algorithm tries to fill in the gap by constructing a new pebble $C' = (\frac{p_i+p'}{2}, \frac{d-r_i-r'}{2})$ and inserting it between C' and C_i.

Size optimization: (Figure 10)

Step 1: Find each leaf node with pebble $C = (p, r)$ that only touches its parent T' with pebble $C' = (p', r')$; grow the pebble by shifting its center in the direction of $\overrightarrow{p'p}$ and expand its radius until it touches another pebble in the neighborhood or boundary. To generate a pattern that fully packs the region boundary, the algorithm overgrows the pebbles at the border by disabling collision detection with the boundary, then trimming any path segments outside of the boundary. Figure 11 shows an example of generated pebbling patterns using both options.

Step 2: For all pebbles $C = (p, r)$ that only touch two other pebbles $C_1 = (p_1, r_1)$ and $C_2 = (p_2, r_2)$, grow C by shifting its center in the direction of $\frac{1}{2}(\frac{\overrightarrow{p_1p}}{|p_1p|} + \frac{\overrightarrow{p_2p}}{|p_2p|})$ while simultaneously expanding it until it touches another pebble or boundary. Collision detection at the border is disabled as in the previous step. This step ensures that all pebbles will touch at least three pebbles.

Rendering: (Figure 11) Our system renders pebbles starting at the root tree node. The algorithm starts by drawing a portion of the circle in a counterclockwise direction, and branches out to draw children in depth-first order. Figure 11 shows an example of rendering pebbles and the final pebbling pattern.

5.3.2 Shape Packing

Shape packing is a generalization of pebbling. Instead of packing circles, primitives of arbitrary shape are now packed. Our approach

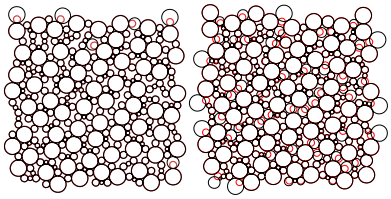

Figure 10: An example of pebbling after each step in phase 2. Left and right show pebbling after step 1 and step 2 respectively. At each stage, the pebbling from previous stage is overlaid in red.

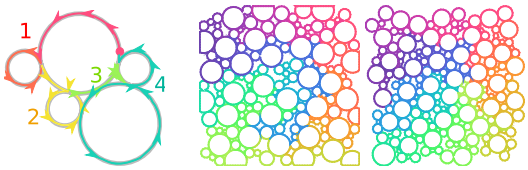

Figure 11: Left: Example of visiting order of a pebble's children. Middle: A final pebbling pattern where collision detection at the boundary is disabled in order to overgrow the pebbles. Path segments outside of the boundary are then trimmed and connected. Right: a final pebbling pattern where collision detection at the boundary is strictly enforced. The global traversal order for both patterns is colored.

generates an underlying pebbling pattern first, then replaces each pebble with a primitive, where the pebble is the minimum bounding circle of the primitive. In Figure 12, each pebble is replaced with a heart. The starting point of a heart is aligned with the ending point of its parent.

6 Results

We used Java to implement our algorithms. In our implementation, each path is represented as a list of three types of SVG path commands: "line to" and "move to", with a 2D point parameter representing the destinations, and "curve to" with two extra points representing the control points of a cubic Bézier curve. The path is then output in a .PAT file format, which is supported by our computer-controlled quilting machine.

Digital patterns. We provide a selection of digital patterns generated using our system. Figures 13 - 18 show how varying parameters changes the look of the patterns for each category. Figure 13 shows Pebbling patterns generated by using 3.3.4.3.4 tessellation with increasing r_{seed} values. Figure 14 shows two sets of four zoomed-in views of Branching patterns using a fixed skeleton path, two decorative element sizes and different initial angles. Figure 15 shows three examples of Wanderer patterns with increasing tangent length l. Figure 16 shows Shape Packing patterns using different primitives for each column. The top and bottom rows show patterns generated with a fixed point distribution and different r_{seed} values. Figure 17 and Figure 18 show various Branching and Wanderer patterns generated using our algorithm. A comparison between Cameli's [3] designs

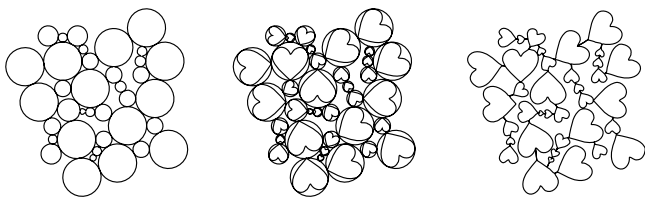

Figure 12: Packing a heart shape inside of circles by first generating a pebbling pattern, then replacing each pebble with a heart.

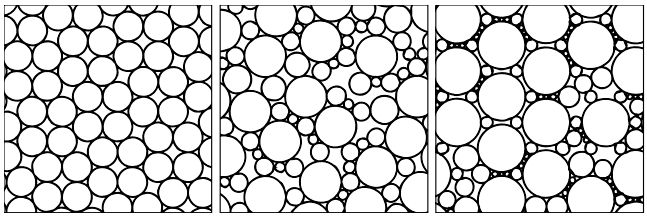

Figure 13: Three examples of pebbling generated with a fixed point distribution distance and r_{seed}=0.5,0.7,0.8 respectively.

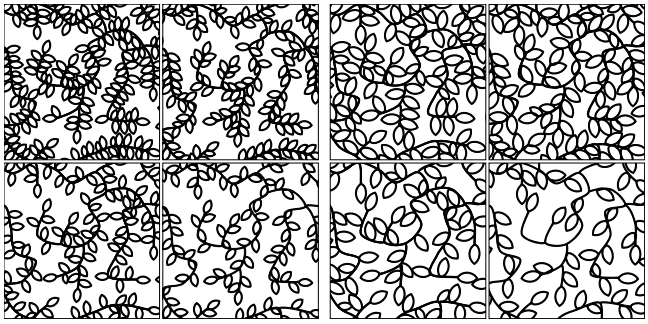

Figure 14: Two sets of four Branching patterns generated using the same skeleton path and decorative element. The left set uses a smaller decorative element size than the right set. In each set, the initial angle is increased from left to right and the gap size is increased from top to bottom.

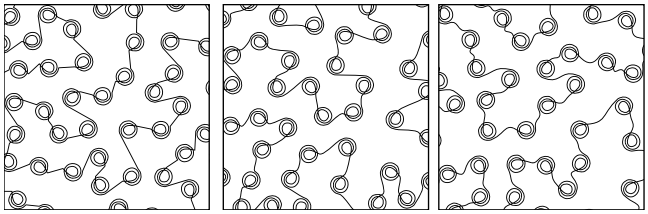

Figure 15: Three zoomed-in views of Wanderer generated with tangent length $l = 0.0, 0.2, 0.5$ respectively. Note the increased curvature of the connecting curve between each pair of primitives.

and patterns generated using our algorithm is shown in Figure 19.

Fabricated quilts. The final quilting path was sewn on an Innova brand 30" long-arm quilting machine, with "Autopilot" computer control software. The stitch density was set to 15 stitches per inch.

Because piecing is a painstaking manual process, we simulated the look of a pieced quilt by designing a block pattern using Photoshop and printing the design on lightweight cotton twill through Spoonflower. The center layer uses low-loft needle-punched cotton batting. The bottom layer uses plain cotton muslin.

For each pattern category, we quilted four designs on simple quilt blocks of 8in $\times$ 8in. We also created three quilts of 29in $\times$ 29in using patterns generated by our system. Figure 20 shows two quilted Pebbling patterns (top) and two Shape Packing patterns (bottom). Figure 21 and 22 show four quilted Branching and Wanderer patterns respectively. Notice the variety that is achieved even within one pattern family, especially when combined with contrasting or matching thread color. In our full quilts, Figures 23 - 25, we took advantage of the randomized nature of our system, making separate, but similar, patterns for similar blocks.

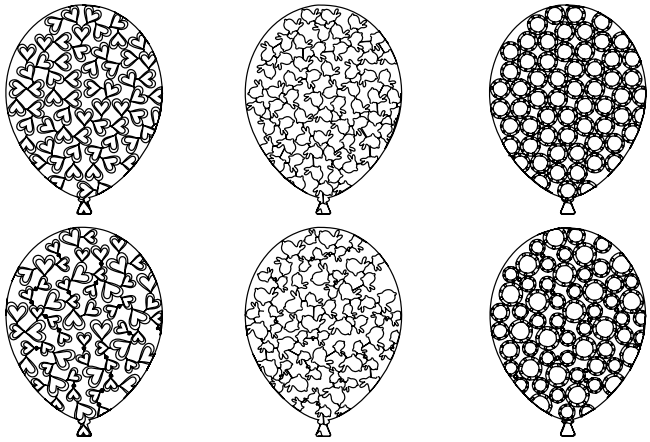

Figure 16: Three sets of shape packing using a heart, a rabbit and "Porthole" by Cameli respectively using the same point set with 63 points. The top and bottom row show results with r_{seed}=0.5 and r_{seed}=0.6 respectively.

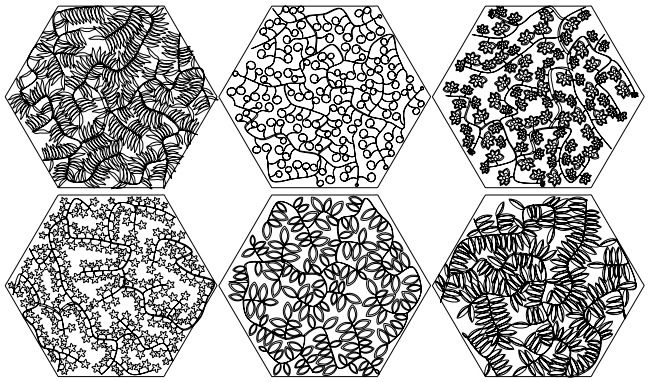

Figure 17: Six examples of Branching patterns generated using our system with a variety of parameters. All examples are generated from Poisson-disk point sampling. Gap length l is set to 0.1 to obtain a packed look. Point set size is around 80.

7 Discussion

In this section we describe additional patterns that we are able to generate, as well as the limitations and performance of our system.

7.1 Additional Pattern Generation

Our framework can easily be extended to generate additional popular free-motion quilting patterns. Stippling, also referred to as meandering, is one of the most popular forms of free-motion quilting patterns that is not included in Cameli's [3] categorization. Stippling can be generated using our algorithm by defining one extra skeleton path post-processing method, fixed-width fill, in addition to Catmull-Rom [5] interpolation. It can be produced by first generating a fixed-width fill pattern where the branch's width is equal to $\frac{1}{3}$ of the point distribution distance, and then interpolating the generated path. Fixed-width fill takes in one parameter, the width of each branch w, and adds an offset equal to $\frac{w}{2}$ perpendicular to the direction of the branches, while traversing up and down a tree branch. Figure 26 shows two quilted fixed-width fill patterns (left) and two stippling patterns (right).

Edge-to-edge quilting and all-over quilting (Figure 27) are two popular techniques that involve repeatedly quilting a "tile." Such tiles are carefully designed so that the quilting path will automati-

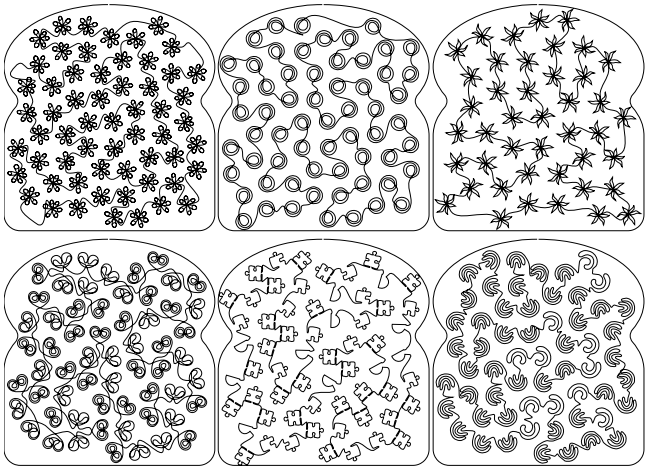

Figure 18: Six examples of Wanderer patterns generated using our system with a variety of parameters. Tangent length l is set to 0.5 to obtain smooth connections between decorative elements. Point set size is around 60.

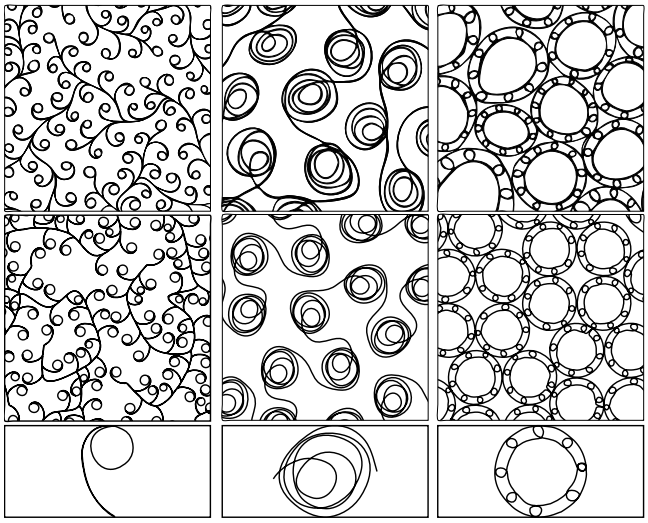

Figure 19: A comparison between Cameli's designs (top) and imitating patterns generated using our framework (bottom) using one decorative element traced from the sketches (bottom). From left to right, the designs are *"Dainty"* (Branching), *"Dizzy"* (Wanderer) and *"Porthole"* (Shape packing), designed by Cameli [3] from C&T Publishing. The "Dainty" imitation uses Poisson-Disk sampling and initial angle 0. The "Dizzy" imitation uses 3.3.4.3.4 tessellation and tangent length $l = 0.5$. The "Porthole" imitation uses 3.3.4.3.4 tessellation and $r_{seed} = 0.5$.

cally connect when placing two tiles side-by-side, sometimes also vertically. When the tiles only connect horizontally, they can be quilted on edges, given the name "edge-to-edge". When the tiles connect both horizontally and vertically, they can be quilted on a whole quilt, given the name "all-over". Our algorithm for Branching can generate such tiles by distributing two special vertically-mirrored points during point distribution, using the tree node containing the start point as the root of the generated spanning tree and modifying the traversal algorithm such that the skeleton path terminates at the end point. Collision tests can be modified such that leaves will not touch each other when tiles are placed side by side.

The quilted patterns are close to their digital representation, although stretching of the fabrics and inaccuracies in the path following of the machine can cause misalignment during quilting. The approximation to the path caused by the finite stitch size may smooth out sharp features of the decorative element when a sharp turn falls in between two stitch points. This problem can be mitigated by carefully designing the stitch path of the decorative elements, although the automatically generated paths cannot be fixed in this way.

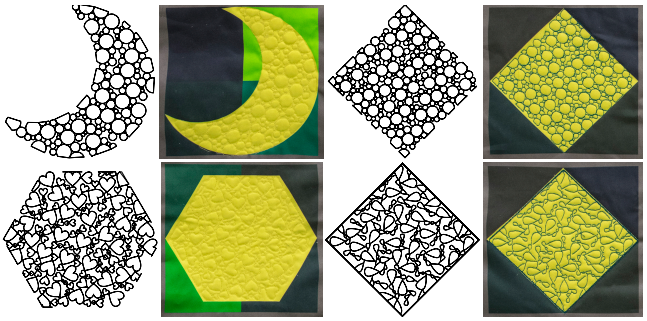

Figure 20: Each of the four quilt blocks use Shape Packing patterns with points generated from 3.3.4.3.4 tessellation with r_{seed}=0.7. The top two quilt blocks use a Pebbling pattern. The bottom two quilt blocks use Shape Packing with a heart (left) and a dewdrop (right) respectively. The moon quilt block has a larger point distribution distance d than the others.

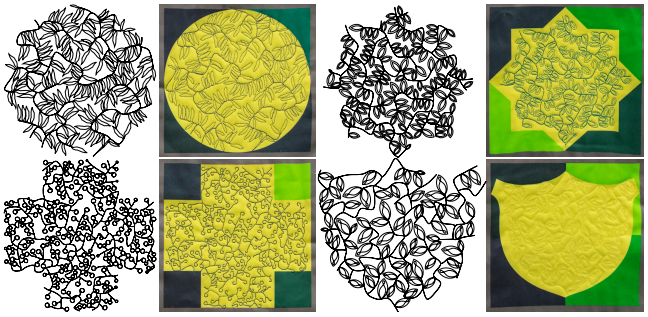

Figure 21: Four quilt blocks using a Branching pattern with initial angles set to 0.

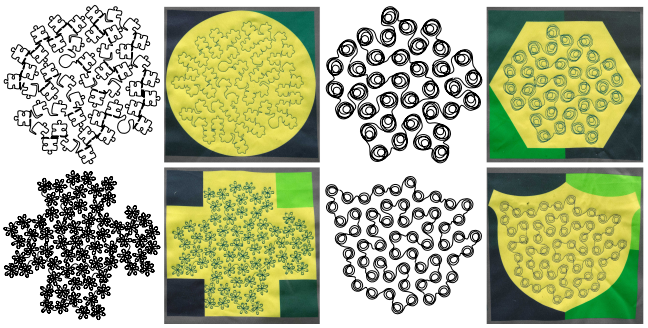

Figure 22: Four quilt blocks using a Wanderer pattern.

7.2 Performance

In practice, our framework is efficient for generating patterns used for quilting, although further optimizations can be implemented. Generation time for a few example instances is shown in Table 1, measured on a MacBook Pro Early 2015 laptop. When generating the spanning tree, our system first constructs a complete graph on all n points, which has $\frac{n(n-1)}{2}$ edges and takes $O(n^2)$ time to construct.

Figure 23: A 29in $\times$ 29in quilt using a combination of pebbling, shape packing with a heart and a dewdrop, branching with a leaf and a spiral, and stippling generated from hexagonal and grid tessellations designed using our system.

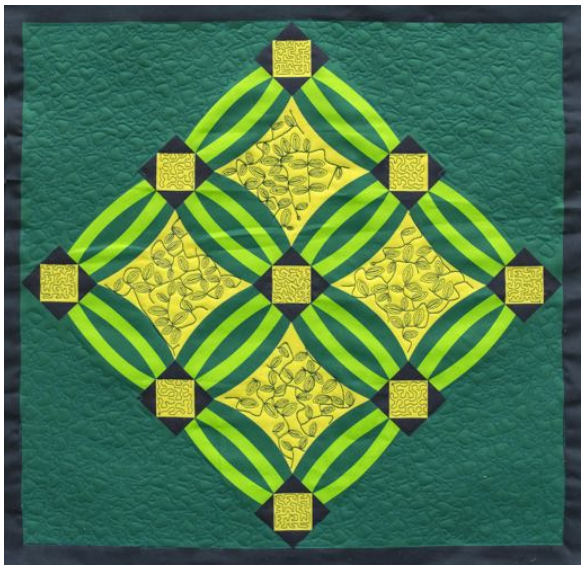

Figure 24: A 29in $\times$ 29in quilt using a combination of shape packing with a heart, branching with a leaf and a spiral, and stippling with hexagonal, grid and 3.3.4.3.4 tessellation designed using our system.

Figure 25: The front side (top) and back side (bottom) of a 29in $\times$ 29in quilt using a combination of wanderer with loops, branching with a leaf and stippling with grid tessellation designed using our system.

We could use Delaunay triangulation instead of a complete graph, which will give $O(n)$ edges and take $O(n \log n)$ time to construct, where n is the number of points distributed. Running the Kruskal's algorithm [12] over the graph with n vertices takes $O(n^2 \log n)$ time. When constructing the spanning tree, we can also accelerate collision testing in Branching by using a hierarchical acceleration structure such as a bounding volume hierarchy.

7.3 Limitations

Our system currently supports three regular, one semi-regular and one random tessellations. Other point distributions could be explored to generate skeleton paths with different aesthetics.

Our algorithm only ensures the bounding circle of each decorative element is packed, not the elements themselves, which leaves gaps between elements and requires the algorithm to connect elements with a short line segment in order to traverse them. We explored a potential solution by building a spanning tree that conforms to the geometry of the element. During point distribution, the decorative element's minimum bounding circle is calculated, and the direction of all points touching the bounding circle is computed. Points are then propagated in these directions. Figure 28 shows a pattern generated using this algorithm. However, this algorithm relies on the geometry of the decorative element. When the decorative element only has two points touching the minimum bounding circle, for example, this algorithm will not be able to distribute points uniformly in the region. An optimization-based solution can be explored to pack elements tightly, although this might significantly increase the run time of the program. It is also non-trivial to come up with a traversal order for the packed elements.

Our current system only supports scaling and rotation for deco-

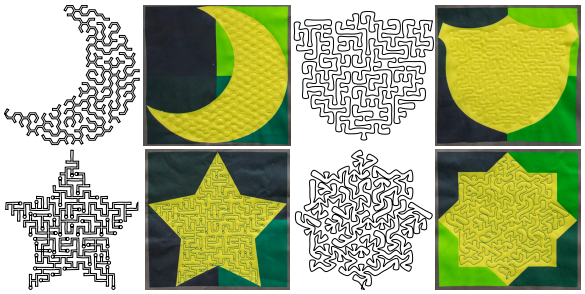

Figure 26: The two quilt blocks on the left each uses a fixed-width fill pattern with $w = \frac{1}{5}d$. The rendered skeleton path of the star is also combined with a decorative element (circle) by placing the element at the tip of each tree branch. The two quilt blocks on the right use a stippling pattern. From upper-left corner in clockwise order, the pattern is generated using hexagonal tessellation, grid tessellation, triangle tessellation and grid tessellation.

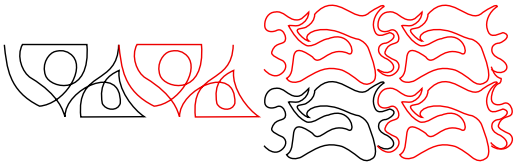

Figure 27: A hand-drawn edge-to-edge design (left) and an all-over quilting design (right). Both designs are complementary horizontally, while all-over design is also complementary vertically.

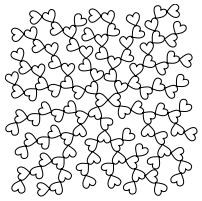

Figure 28: An example of shape packing of hearts, where the underlying spanning tree structure is built using points propagated in the directions of the element that touches its minimum bounding circle.

rative elements. These two operations are sufficient to generate the pattern families shown in this paper, although it is not clear how Emerging, Echoing, and Climber patterns (shown in Figure 2) can be generated using only these two operations.

Our implementation requires only one template for the decorative element. We could ask the user to input a library of decorative elements with slight variations and randomly place them in the pattern in order to achieve more interesting results.

8 Future Work

In Cameli's categorization of FMQ patterns [3], Emerging, Echoing, and Climber patterns (shown in Figure 2) allow more free-form transformations and will require a more advanced underlying representation of the decorative elements as well as collision and packing algorithms.

A user interface for user sketching, real-time preview and integrated spline manipulation tools that allow path segment editing on the output will allow quilters to iteratively improve designs.

Table 1: Generation time of example instances using our framework. Each instance is run 5 times and the mean time is computed.

Instance	Point Set Size	Mean Generation Time
Heart Packing	118	0.62s
Leaf Branching	118	0.77s
"Dizzy" Wanderer	102	0.67s
Heart Packing	892	8.09s
Leaf Branching	892	6.59s
"Dizzy" Wanderer	755	6.63s

References

[1] D. Anderson and Z. Wood. User driven two-dimensional computer-generated ornamentation. In *Proceedings of the 4th International Symposium on Advances in Visual Computing*, ISVC '08, pp. 604–613. Springer-Verlag, Berlin, Heidelberg, 2008.

[2] D. Applegate, R. Bixby, V. Chvatal, and W. Cook. Concorde tsp solver, 2006.

[3] C. Cameli. *Step-by-Step Free-Motion Quilting: Turn 9 Simple Shapes into 80 Distinctive Designs*. C & T Publishing, Inc., Lafayette, CA, 2015.

[4] C. Carlson, N. Paley, and T. Gray. Algorithmic quilting. In D. M. Kelly Delp, Craig S. Kaplan and R. Sarhangi, eds., *Proceedings of Bridges 2015: Mathematics, Music, Art, Architecture, Culture*, pp. 231–238. Tessellations Publishing, Phoenix, Arizona, 2015. Available online at `http://archive.bridgesmathart.org/2015/bridges2015-231.html`.

[5] E. Cohen, R. Riesenfeld, and G. Elber. *Geometric Modeling with Splines*. A K Peters, 2001.

[6] G. Q. M. Company. Creative Studio Software, 2017. [Online; accessed April 4, 2018].

[7] T. E. Q. Company. Electric Quilt Software, 1999 - 2018. [Online; accessed April 4, 2018].

[8] M. Ghyka. *The geometry of art and life*. Dover Publications, 1977.

[9] Y. Igarashi and J. Mitani. Patchy: An interactive patchwork design system. In *ACM SIGGRAPH 2015 Posters*, SIGGRAPH '15, pp. 10:1–10:1, 2015.

[10] C. S. Kaplan and R. Bosch. Tsp art. In R. Sarhangi and R. V. Moody, eds., *Renaissance Banff: Mathematics, Music, Art, Culture*, pp. 301–308. Bridges Conference, Southwestern College, Winfield, Kansas, 2005. Available online at `http://archive.bridgesmathart.org/2005/bridges2005-301.html`.

[11] M. Kaplan and E. Cohen. Computer generated celtic design. In *Proceedings of the 14th Eurographics Workshop on Rendering*, EGRW '03, pp. 9–19, 2003.

[12] J. B. Kruskal. On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem. In *Proceedings of the American Mathematical Society, 7*, 1956.

[13] H. Li and D. Mould. Continuous line drawings and designs. *Int. J. Creat. Interaces Comput. Graph.*, 5(2):16–39, July 2014.

[14] C. Liu, J. Hodgins, and J. McCann. Whole-cloth quilting patterns from photographs. In *Proceedings of the Symposium on Non-Photorealistic Animation and Rendering*, NPAR '17, pp. 7:1–7:8, 2017.

[15] J. Lu, C. Barnes, C. Wan, P. Asente, R. Mech, and A. Finkelstein. Decobrush: drawing structured decorative patterns by example. *ACM Trans. Graph.*, 33:90:1–90:9, 2014.

[16] R. Měch and G. Miller. The *Deco* framework for interactive procedural modeling. *Journal of Computer Graphics Techniques (JCGT)*, 1(1):43–99, Dec 2012.

[17] H. Pedersen and K. Singh. Organic labyrinths and mazes. In *Proceedings of the 4th International Symposium on Non-photorealistic Animation and Rendering*, NPAR '06, pp. 79–86, 2006.

[18] L. van der Heijden. Art & Stitch Software, 2010 - 2018. [Online; accessed April 4, 2018].

[19] F. J. Wong and S. Takahashi. A graph-based approach to continuous line illustrations with variable levels of detail. *Comput. Graph. Forum*, 30:1931–1939, 2011.

[20] M. T. Wong, D. E. Zongker, and D. H. Salesin. Computer-generated floral ornament. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '98, pp. 423–434, 1998.