Algorithmic Quilting

Christopher Carlson
Wolfram Research
100 Trade Centre Dr
Champaign, IL, 62180, USA
E-mail: carlson@wolfram.com

Nina Paley, Theodore Gray
PaleGray Labs
102 E Main St
Urbana, IL 61801
E-mail: nina@ninapaley.com,
theodore@theodoregray.com

Abstract

A project to produce a "quiltimation", an animation in which every frame is a quilt, motivated the development of automatic methods for generating single-line fills of arbitrary regions. We present one slice through a developing kit of algorithmic quilting tools: a 4-step process for generating fills, each step of which can be varied to yield diverse graphic effects.

Quiltimation

One of the authors (Nina) is an animator. When she began making art quilts in 2011, a running joke among friends and fans was that she would quilt her next animation ("funny" because 2-D animation is already a notoriously labor-intensive medium). In 2013 she realized that technology existed to make the joke a reality. She acquired a 10-needle embroidery machine as well as a bedroom-sized, CNC quilting machine that will stitch an entire quilt under computer control given a line drawing of the stitch paths (figure 1) [1].

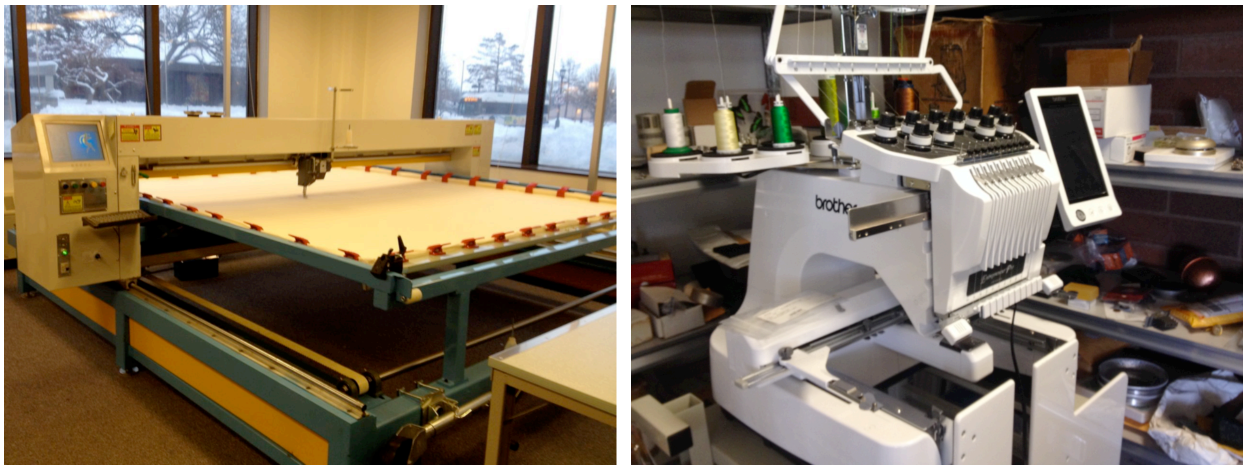

Figure 1: *The CNC quilting and embroidery machines at PaleGray Labs.*

Beyond the appeal of showing that a task as absurd as a "quiltimation" can be done, quilting has several aesthetic qualities to recommend it as a medium for animation. Tactile, bas-relief quilts lend animations a compelling physicality that lies somewhere between traditional hand-drawn animation and claymation. The visual paradox of a static medium in motion lends interest in itself. And the imprecision inherent in an elastic medium like fabric gives animations a charming aesthetic that contrasts with the slick precision of computer animations.

There is little precedent for automated quilted animation, but we know of two examples of embroidered animation: Nicos Livesey's "Tharsis Sleeps" [2], and Aubrey Longley-Cook's "Runaway" [3]. The latter example is stitched by hand, and much more effectively conveys the aesthetic qualities of embroidery than the former, which was made with off-the-shelf software and machine stitched.

The only precedent for quilted animation we know of is the author's "Ziz" [4]. This features an algorithmically generated quilted background, an echo pattern consisting of individual concentric shapes stitched as multiple lines on an embroidery machine with an automatic thread cutter.

Producing even a 3-minute quiltimation at 12 frames per second would require over 2000 quilts. Larger areas of quilts are typically filled with detailed patterns of stitching that hold the layers of fabric and batting together and provide graphic interest. While the geometry of each frame of a quiltimation would be designed by hand, adding the detailed fills to that geometry in each frame manually is beyond contemplation.

Because cutting the thread when CNC quilting is a time-consuming operation that requires a human presence, quilts designed for CNC quilting should have as few thread-cuts as possible. Ideally, an entire quilt can be stitched with a single stitch path. Single-line stitching is stronger and more elegant than patterns with thread cuts, and single-line art stitches faster.

These two requirements–automatic fills and single-line stitch paths–led us to implement a growing collection of algorithmic tools for creating single-line fills of arbitrary regions. Our goal is to assemble a diverse set of algorithmic components that can be combined in various ways to yield a large variety of aesthetically pleasing graphic effects. The toolkit is implemented in the Wolfram Language and runs in Mathematica [8]. The Wolfram Language's high-level algorithms for regions, graphs, and graphics make it ideal for this application.

Digital quilting fill patterns are available from several commercial sources, but these are universally tilings of stock patterns [5][6][7]. They cannot respond to the contents of the frame, morph smoothly, or vary in density along gradients, all desirable qualities of which our fills are capable.

Single-Line Fills

In conventional machine quilting, larger areas are often filled with "free-motion quilting" patterns. These are patterns sewn freehand on a sewing machine according to intuitive rules that govern the pattern (figure 2) [9].

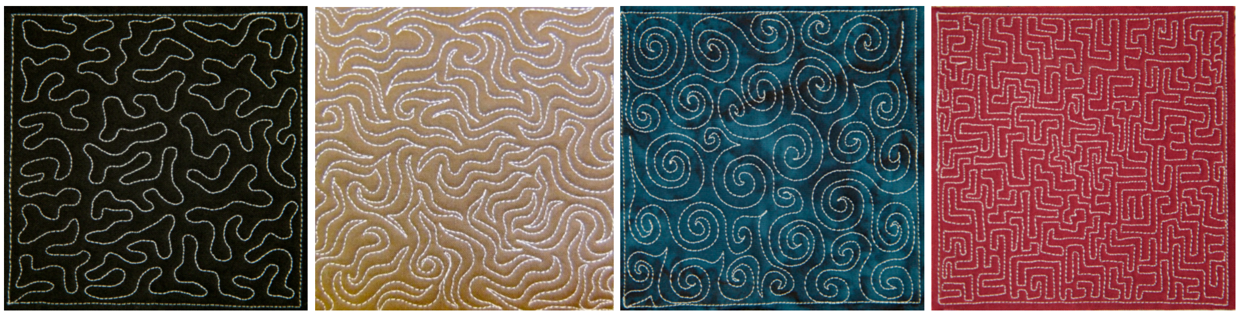

Figure 2: *Free-motion quilting patterns by Leah Day.*

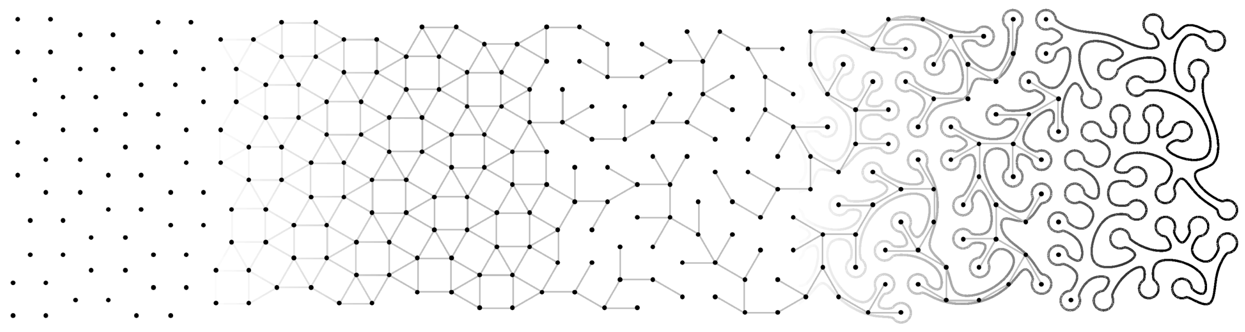

Figure 3: *Steps in the computation of a stitch path: distributing points, connecting points to form a graph, crawling the graph to generate a spanning tree, and traversing the tree to render the stitch path.*

Free-motion quilting is the inspiration for our algorithmic tools, but we have not attempted to duplicate how humans do it. Instead, we have devised methods that lend themselves to computer implementation, but nevertheless provide the components necessary to achieve results reminiscent of free-motion quilting.

We present in this paper one slice through our algorithmic toolkit, a method of generating single-line fills of arbitrary regions that consists of four steps (figure 3):

- distribute points in a region
- connect selected pairs of points to form a graph
- crawl the graph to generate a spanning tree
- traverse the spanning tree to render a stitch path for the region

Each of those steps can be performed in myriad ways, and combining the variations yields a great diversity of graphic effects.

Distributing Points in Regions

Points distributed in a 2D region are the foundation of the structure that will eventually yield a stitch path. The region must be connected to ensure that a single path can be constructed through it, but is otherwise unconstrained, and may have irregular boundaries and contain holes. The stitch path doesn't necessarily pass through the points, but its course through the region is determined by their geometry.

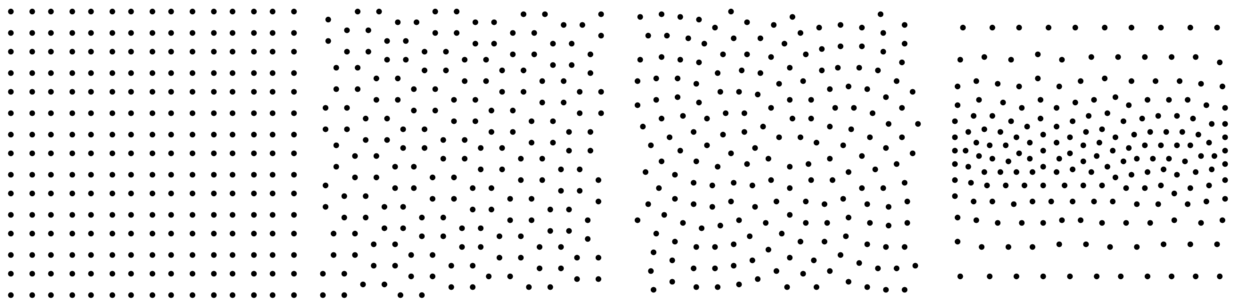

Figure 4: *Point distributions: regular grid, vertices of a 3.3.4.3.4 tessellation, phyllotaxic spiral, and varying density distribution from a weighted centroidal Voronoi tessellation.*

Our initial experiments have used point distributions based on the vertices of regular and semi-regular tessellations [10], phyllotaxic distributions [11], and varying-density distributions based on weighted centroidal Voronoi tessellations [12] (figure 4). The vertices of the 3.3.4.3.4 tessellation yield designs with particularly pleasing combinations of regularity and apparent irregularity, but even regular grids of points can yield intriguing designs, depending on the subsequent steps applied to them.

Constructing Graphs

After points have been distributed in a region, pairs of points are joined with edges to create a graph. Any graph is permissible, but considerations of aesthetics and efficiency generally dictate that points are joined only to other points within a relatively small neighborhood. For distributions with uniform densities, joining points that lie within narrow ranges of distances usually gives good results.

Obtaining pleasing graph structures when points are unevenly dense is more difficult. Connecting *n* nearest neighbors can give pleasing structures in the interiors of regions, but at the edges, where points have fewer neighbors, edge structures become busy and non-uniform, with many edges crossing. The only uneven distributions we have experimented with thus far result from centroidal Voronoi tessellations, where we can use cell adjacencies to determine the connections between centroids. We are investigating methods that will yield pleasing graph structures for arbitrary point distributions.

Figure 5 shows graphs created on the point distributions of figure 4 using distance and cell-adjacency criteria.

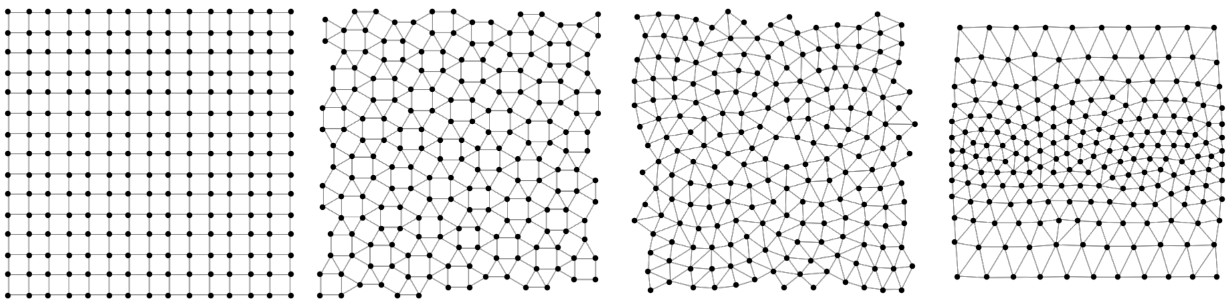

Figure 5: *Graphs constructed on the point distributions of figure 4: 4-nearest neighbor, 5-nearest neighbor, neighbors less than a threshold distance, and Voronoi cell adjacencies.*

Crawling Graphs

The goal of the stitch path computation is to produce a path that visits the neighborhood of every point in the initial point distribution. That goal could be achieved by the computation of Hamiltonian paths or more specifically, traveling salesman paths [13], but there is a simpler method that gives more path variety.

Spanning trees are relatively straightforward to compute, and by varying the computation method, diverse tree structures can be produced. From a spanning tree, we compute a stitch path by traversing the tree depth-first from the root, traversing edges incident at each vertex in angular order around the vertex.

The technique we use to compute spanning trees is conceptually equivalent to placing a graph-crawling robot at a graph vertex and letting it navigate the graph, guided by simple programs that make decisions about which edges to traverse next based on the immediate neighborhood of the current vertex. At each vertex, the crawler selects any number of incident edges to traverse, putting them at the head or tail of a traversal queue. Edges that lead to vertices already visited are disqualified in order to prevent cycles. After edges are queued for traversal, the edge at the head of the traversal queue is dequeued, and the process repeats with the vertex at the other end of the edge.

A second, backtracking queue guarantees under mild conditions that every vertex will be visited before the algorithm terminates, thus ensuring the generation of spanning trees. Each vertex visited along a path is added to the backtracking queue. If at any point the traversal queue is empty, the next vertex with untraversed edges is dequeued from the backtracking queue. The process stops when both queues are empty. If the robot's rules require it to traverse at least one edge when any are available, the algorithm will yield a spanning tree. The structure of the resulting tree depends on the method used to select the edges to traverse, the order in which they are queued, whether they are queued at the head or tail of the queue, and whether vertices are queued at the head or tail of the backtracking queue.

Figure 6 gives a sample of the variety of spanning tree structures various robot rules produce. At each vertex visited, *random depth-first branching* queues a randomly selected edge to the head of the queue. Thus it follows a random path from the root to a leaf of the tree, and then backtracks to a previous vertex with untraversed edges and repeats. *Random depth- and breadth-first branching* queues all of a vertex's edges in order randomly to either the head or tail of the queue. That strategy tends to yield tree structures with long, bushy branch chains. *Breadth-first branching* queues a vertex's edges in order to the tail of the queue, extending a tree layer-by-layer and yielding broad, bushy branch structures. *First^3-last^3 branching* queues the first edge to the head of the queue for three vertices, then the last edge for three vertices. The result is a tree with zigzagging branches. Further traversal strategies yield trees with paths that tend to spiral, paths that weave back-and-forth, and paths that are primarily horizontal or vertical.

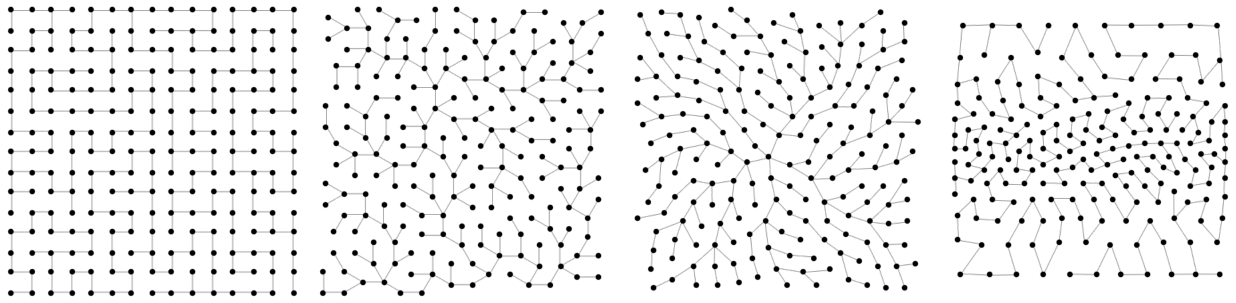

Figure 6: *Spanning trees of the graphs in figure 5: random depth-first branching, alternate depth- and breadth-first, breadth-first, and first^3-last^3 strategies.*

Rendering Stitch Paths

The final step is to traverse the spanning tree to render a stitch path. Traversal starts at the root and proceeds depth-first through the tree, traversing each vertex's edges in counterclockwise order. If the sequence of visited vertex positions is offset to the right of the direction of travel, the result is an outline of the tree.

At each vertex, the last, current, and next vertices are passed to a joint rendering function that adds a segment to a developing path. We have a growing collection of joint renderers that give various effects. Which renderer is applied to a joint typically depends on the joint's angular geometry, often with different renderers applied to convex, concave, and 180° angles (i.e., branch tips). Renderers can add bulbs or finials to branch tips, render concave joints with sharp or filleted bends, and convex ones with angular or rounded elbows. The path information accumulated by the renderers, typically a list of points, is passed to a final finishing step that renders the stitch path as a polyline or B-spline curve.

Figure 7 shows the variety of effects that renderers in various combinations can produce: a constant-width path with rounded bends and "bulbs" at branch tips, a B-spline path with sprout-like "finials" at branch tips, a simple constant-width B-spline path with bulbs, and a polyline path with radiused joints where the offset of the path from the spanning tree is governed by the distance to the nearest neighbor of each vertex.

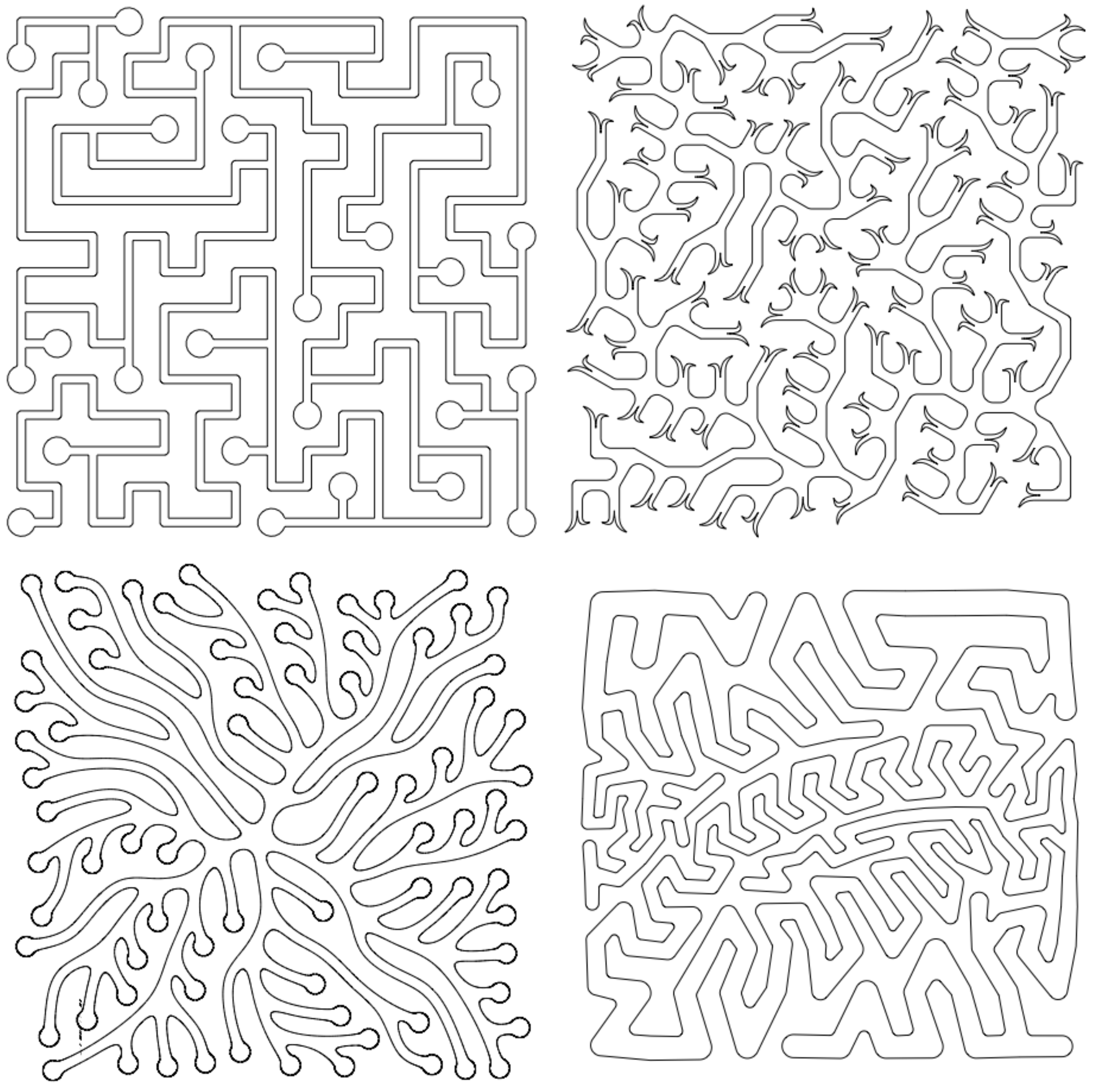

Figure 7: *Stitch paths computed from the spanning trees in figure 6: polyline path with bulbs, B-spline path with "finials", B-spline path with bulbs, and varying width polyline path.*

Figure 8 shows a sampler quilt we stitched to test the effects of various algorithmic fills. The horses are silhouettes of Eadweard Muybridge's famous photographic motion studies [14]. The original is quilted with black fabric; we reproduce it here as a color negative so that the background fills are easier to see. Animations of the quilted frames can be seen at [15]. Although we use different fills (and colors) in each frame, continuity is provided by the outline of the horse.

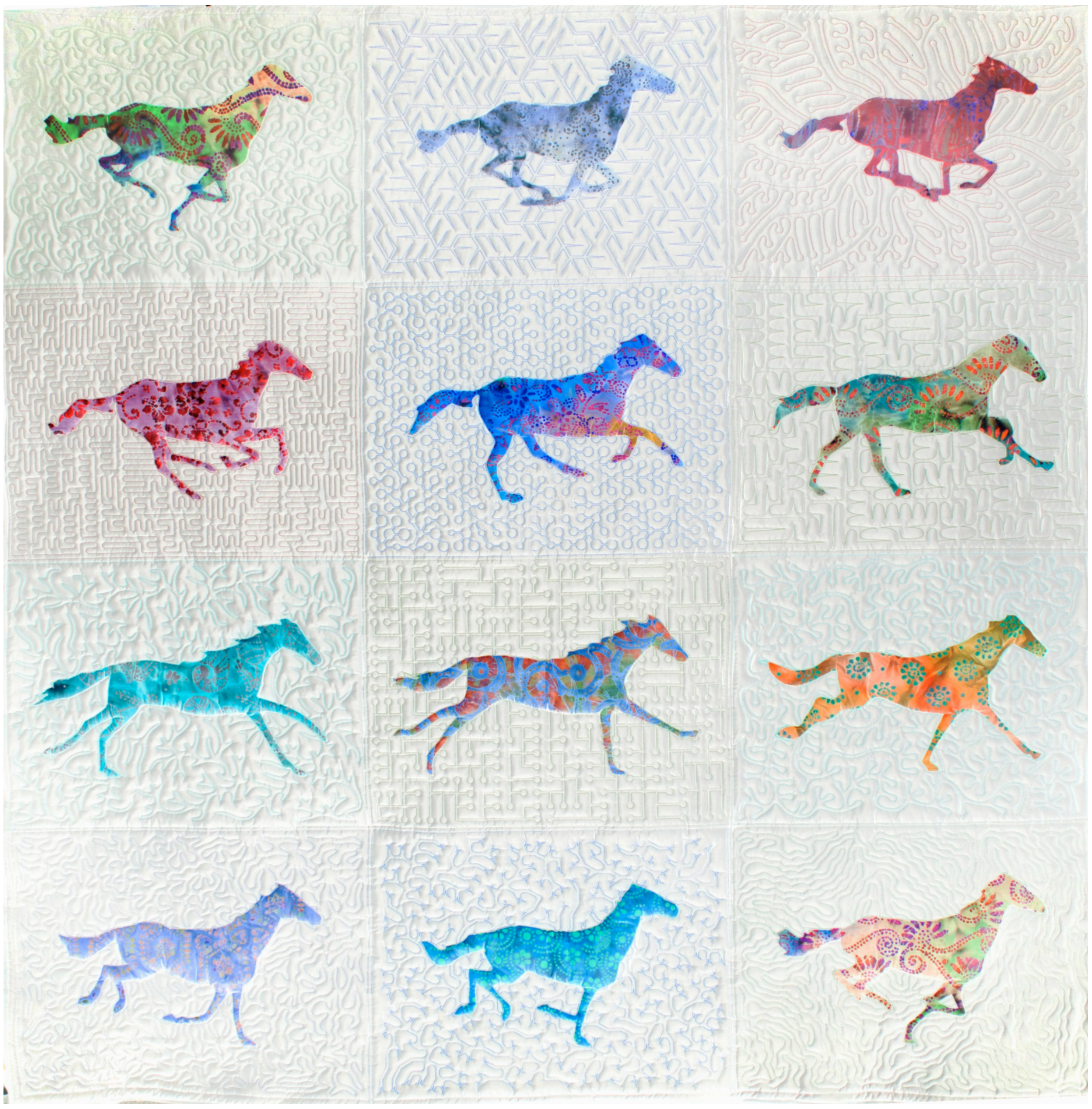

Figure 8: *Sampler quilt stitched with an assortment of algorithmic fills.*

We have many avenues yet to explore and issues to resolve before we're ready to produce a complete quiltimation. We don't yet have a generally applicable algorithm for generating graphically pleasing graphs from arbitrary point distributions. We have not yet thoroughly understood the interactions of graph crawling rules with graph topologies and the structural possibilities they admit. And we haven't begun to address the issue of frame-to-frame continuity required for animations.

It's worth noting that although algorithmic fills are necessary for quiltimation, there are interesting still applications as well. In particular, algorithmic fills could produce compelling appliqué, reverse-appliqué, and double-quilted patterns. (Double-quilting is a technique invented by Theodore Gray for producing rich bas-relief effects by quilting a pattern, adding batting, and quilting again at a larger scale.)

We look forward to enriching our toolkit of algorithmic components, which will increase opportunities to discover algorithmic combinations that yield surprising new fills.

References

[1] PaleGray Labs, "Our Machines", http://palegraylabs.com/our-machines/
[2] Livesey, N, "Tharsis Sleeps", https://vimeo.com/97718226
[3] Cook, AL, "Runaway", http://spoolspectrum.blogspot.com/2010/06/runaway.html
[4] Paley, N, "Ziz", http://blog.ninapaley.com/2013/08/18/embroidermation-test-4/
[5] Quilts Complete, http://www.quiltscomplete.com/departments/computerized.aspx
[6] Intelligent Quilting, http://www.intelligentquilting.com/
[7] Wasatch Quilting, http://www.digitizedquiltingpatterns.com/shop/category/edge-to-edge/
[8] Wolfram Research, *Mathematica*, http://www.wolfram.com/mathematica/?source=nav
[9] Day, L, "Gallery of Free Motion Quilting Designs", http://www.leahday.com/project1/
[10] Grünbaum, B and Shephard, GC, *Tilings and Patterns* (1986), W. H. Freeman
[11] Coxeter, HSM, *Introduction to geometry* (1961), Wiley, p 169
[12] Secord, A, "Weighted Voronoi Stippling", *Proc. Of the International Symposium on Non-Photorealistic Animation and Rendering* (2002), pp 37–43
[13] Bosch, R and Kaplan CS, "TSP Art", *Proceedings of Bridges 2005* (2005) pp 303-310
[14] http://en.wikipedia.org/wiki/File:Muybridge_race_horse_animated.gif
[15] http://ninapaley.com/algorithmicquilting.html

(all URLs as of 2015.04.20)