Edge-Constrained Tile Mosaics

Robert Bosch
Dept. of Mathematics
Oberlin College
Oberlin, OH 44074
(bobb@cs.oberlin.edu)

Abstract

We present a new method for reproducing images as mosaics, forcing adjacent edges of adjacent tiles to match.

1 Introduction

It is an outstanding time to be a mosaicist! Not only can you emulate the great artists of the last several thousand years and build with pieces of stone, glass, and pottery, but with the assistance of computers, you can construct intricate large-scale mosaics out of everyday objects like dice [8], dominoes [1,2,3,7,8,9], LEGO® bricks [5], toy cars [8], spools of thread [8,15], baseball cards [8,14], foodstuffs [10], photographs [13,14], and even individual frames of films like *Star Wars* and *It's a Wonderful Life* [11].

In this paper, we build upon the tradition of Knowlton, the father of computer-assisted mosaicking [12], and introduce *edge-constrained tile mosaics*—mosaics whose building-block tiles must be arranged so that the patterns on adjacent edges of adjacent tiles match one another.

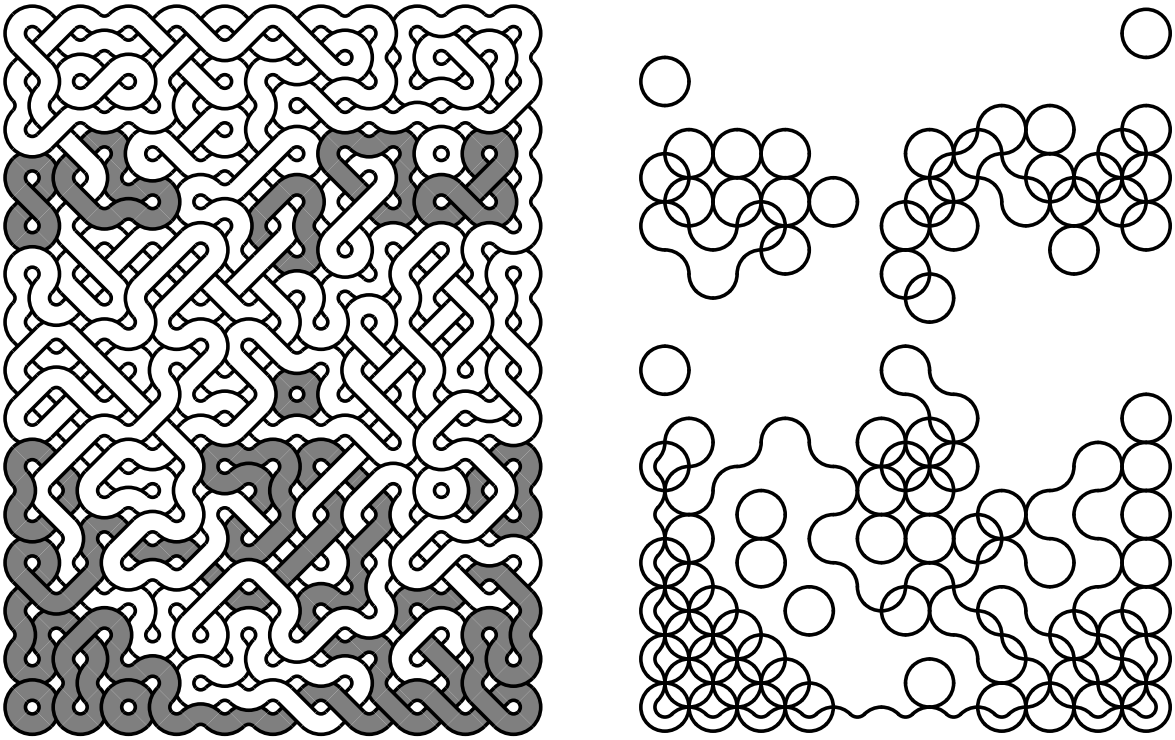

Figure 1: da Vinci's *Mona Lisa* as (a) a knot-tile mosaic, (b) an arc-tile mosaic ($m = 30$ and $n = 22$)

Figure 1 displays two edge-constrained tile mosaics (ECTMs) based on a section of Leonardo da Vinci's *Mona Lisa.* At first glance, it is not clear that these pictures are in fact mosaics; yet Figure 1(a) is indeed an ECTM constructed out of the 24 knot tiles displayed in Figure 2, and Figure 1(b) is indeed an ECTM constructed out of the 36 arc tiles displayed in Figure 4. (Note that if we lay down the knot tiles in an edge-to-edge fashion so that their edge patterns match, the resulting mosaic will appear to contain loops of light thread and dark thread that interact in a way that brings to mind Celtic knotwork.)

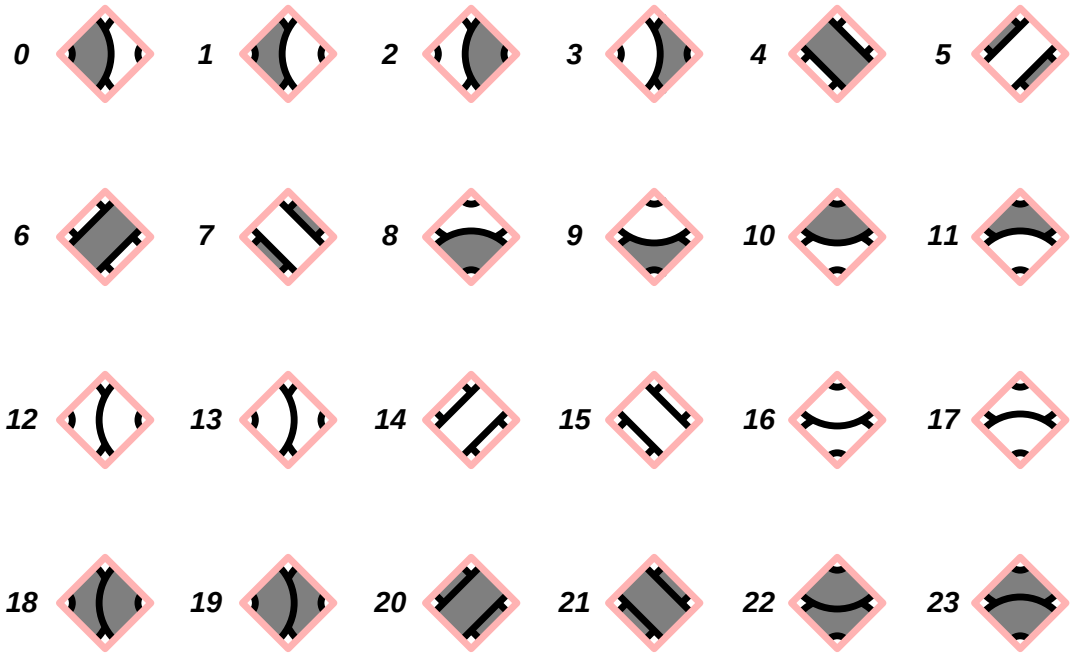

Figure 2: the 24 knot tiles

2 Knot-tile Mosaics

In this section, we describe how to build ECTMs out of the 24 knot tiles displayed in Figure 2. Our first step is to represent the edge pattern of each tile $t \in T = \{0,1,\ldots,23\}$ by an array

$$\big[NE(t), SE(t), SW(t), NW(t)\big],$$

allowing each entry to take on the value "•" (when a strand of dark thread passes through the edge) and "○" (when a strand of light thread passes through it). Next, we assign to each t a brightness value b_t, using a 0-to-1, black-to-white scale. One simple scheme that works quite well (surprisingly!) and has significant advantages (which will be discussed shortly) is to set b_t equal to one fourth the number of "○" values in t's edge-pattern array. This scheme gives each tile with two dark threads (row 4 of Figure 2) a brightness of 0, each tile with one dark thread and one light thread (rows 1 and 2 of Figure 2) a brightness of $1/2$, and each tile with two light threads (row 3 of Figure 2) a brightness of 1. Tile 0 ends up with a brightness value of $b_0 = 1/2$ since

$$\big[NE(0), SE(0), SW(0), NW(0)\big] = [\circ, \circ, \bullet, \bullet].$$

The third step is to chop both the target image and the initially blank canvas into a grid of diamonds $D = \{(i,j) : 0 < i < m,\ 0 < j < n,\ (i+j) \bmod 2 = 1\}$. For the small "braid" mosaic displayed in Figure 3, $m = 10$ and $n = 6$. For the mosaic displayed in Figure 1(a), $m = 30$ and $n = 22$. We let D_{SE} and D_{SW} stand for

the sets of diamonds that have southeast and southwest neighbors, respectively, and we denote the brightness of diamond (i, j) of our target image by $\beta_{i,j}$, once again using a 0-to-1, black-to-white scale.

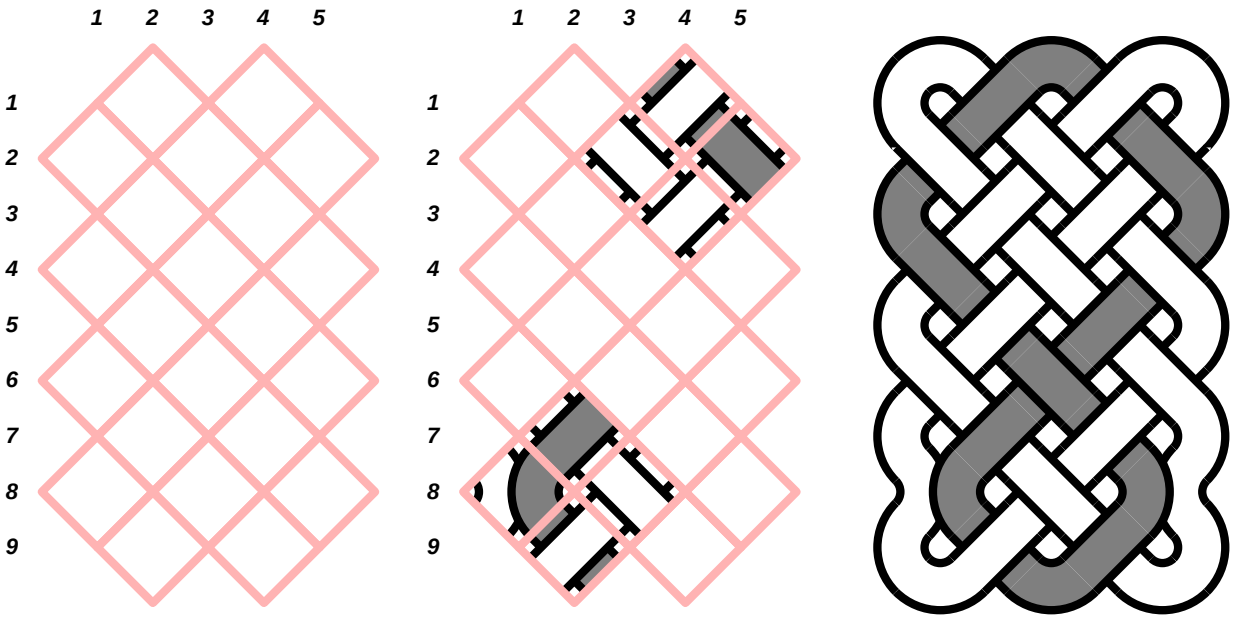

Figure 3: (a) gridded canvas, (b) partial mosaic, (c) final mosaic ($m = 10$ and $n = 6$)

The final step is to arrange the tiles on the canvas. To do this, we must answer, for each tile t and each diamond (i, j), a yes-no question: "Are we going to place a copy of tile t in diamond (i, j)?" To model this, we introduce a binary decision variable for each tile t and diamond (i, j), setting $x_{t,i,j} = 1$ if we do indeed place a copy of tile t in diamond (i, j) and setting $x_{t,i,j} = 0$ otherwise. Note that to obtain Figure 3(b), we set the following binary decision variables equal to 1: $x_{5,1,4}$, $x_{15,2,3}$, $x_{4,2,5}$, $x_{14,3,4}$, $x_{6,7,2}$, $x_{2,8,1}$, $x_{15,8,3}$, and $x_{5,9,2}$.

With the $x_{t,i,j}$ variables, it is relatively straightforward, following the example set forth in [1], to formulate the tile arrangement problem as an integer program (IP) [17], a mathematical optimization problem that has a linear objective function, linear constraints (inequalities and/or equations), and decision variables that take on integer values:

$$\text{minimize} \quad z = \sum_{t \in T} \sum_{(i,j) \in D} (b_t - \beta_{i,j})^2 x_{t,i,j} \tag{1}$$

$$\text{subject to} \quad \sum_{t \in T : SE(t) = \bullet} x_{t,i,j} = \sum_{t \in T : NW(t) = \bullet} x_{t,i+1,j+1} \qquad \text{for each } (i,j) \in D_{\text{SE}} \tag{2}$$

$$\sum_{t \in T : SE(t) = \circ} x_{t,i,j} = \sum_{t \in T : NW(t) = \circ} x_{t,i+1,j+1} \qquad \text{for each } (i,j) \in D_{\text{SE}} \tag{3}$$

$$\sum_{t \in T : SW(t) = \bullet} x_{t,i,j} = \sum_{t \in T : NE(t) = \bullet} x_{t,i+1,j-1} \qquad \text{for each } (i,j) \in D_{\text{SW}} \tag{4}$$

$$\sum_{t \in T : SW(t) = \circ} x_{t,i,j} = \sum_{t \in T : NE(t) = \circ} x_{t,i+1,j-1} \qquad \text{for each } (i,j) \in D_{\text{SW}} \tag{5}$$

$$\sum_{t \in T} x_{t,i,j} = 1 \qquad \text{for each } (i,j) \in D \tag{6}$$

$$x_{t,i,j} \in \{0,1\} \qquad \text{for each } t \in T, (i,j) \in D. \tag{7}$$

Equation (1) states our objective: when we assign values to the $x_{t,i,j}$ variables, we want to minimize z,

the sum of the cost terms $(b_t - \beta_{i,j})^2 x_{t,i,j}$. Note that each time we set an $x_{t,i,j}$ variable equal to 1, we end up adding the corresponding cost coefficient $(b_t - \beta_{i,j})^2$ to z. These $(b_t - \beta_{i,j})^2$ coefficients encourage us to think hard about brightnesses every time we place a tile t in a diamond (i, j). The closer b_t is to $\beta_{i,j}$, the lower our cost.

The first four sets of constraints handle the edge pattern matching. Sets (2) and (3) work together to make sure that the southeast edge pattern of the tile placed in diamond (i, j) matches the northwest edge pattern of the tile placed in diamond $(i+1, j+1)$. The (2) constraints focus on dark threads, and the (3) constraints focus on light ones. Each states that the total number of tiles placed in diamond (i, j) that have a specified shade of thread coming out of their southeast edge must equal the total number of tiles placed in diamond $(i+1, j+1)$ that have that same shade of thread coming out of their northwest edge. Sets (4) and (5) play roles similar to (2) and (3), but for the pairing of the southwest edge of the tile placed in diamond (i, j) with the northeast edge of the tile placed in diamond $(i+1, j-1)$.

The remaining constraints, (6) and (7), are much easier to understand: together, they assert that we must place exactly one tile in each diamond of the canvas. (Note that if we were to drop (7), we'd have a *linear* program, not an *integer* program, and the optimal solution could very well have $x_{t,i,j} = x_{s,i,j} = 0.5$, which would tell us to place half of tile t *and* half of tile s in diamond (i, j)!)

As written above, the knot-mosaic IP has a total of $2|D_{\text{SE}}| + 2|D_{\text{SW}}| + |D|$ linear constraints and $|T| \cdot |D|$ binary decision variables. To generate the knot-tile mosaics in Figures 1(a) and 6, we actually added a pair of constraints—similar to those in sets (2) through (5)—for every pair of adjacent diamonds on the boundary of our canvas. These constraints guarantee that we can close off every loop of thread. For example, we added the pair of constraints

$$\sum_{t \in T : NE(t) = \bullet} x_{t,2,1} = \sum_{t \in T : NW(t) = \bullet} x_{t,4,1} \qquad \text{and} \qquad \sum_{t \in T : NE(t) = \circ} x_{t,2,1} = \sum_{t \in T : NW(t) = \circ} x_{t,4,1}$$

to ensure that we can connect the northeast edge of the tile placed in $(1,2)$ to the northwest edge of the tile placed in $(1,4)$.

Here's another fine point: If we use the tile-brightness-assignment scheme described earlier, we can partition T into eight classes of tiles with common brightness values and edge patterns: $\{0,1\}$, $\{2,3\}$, $\{4,5\}$, $\{6,7\}$, $\{8,9\}$, $\{10,11\}$, $\{12,13,\ldots,17\}$, and $\{18,19,\ldots,23\}$. Doing this enables us to reduce $|T|$ from 24 to 8, which cuts the number of binary decision variables by a factor of two thirds. From a practical standpoint, this enables us to solve the integer progams much more quickly.

When all of this is done, for $m = 30$ and $n = 22$ the integer program has 1528 constraints and 2432 binary decision variables. For $m = 60$ and $n = 44$, it has 6348 constraints and 10144 binary decision variables. Commericial optimization software packages such as ILOG's CPLEX [6] can handle problems of this size, using a branch-and-bound (divide-and-conquer) procedure [17]. Each of the knot-tile mosaics displayed here—in Figures 1(a) and 6—required only seconds for CPLEX on a Pentium IV laptop.

3 Other tiles

It is easy to modify the knot-tile integer program for use with other sets of tiles, but the resulting integer programs can get quite large and difficult to solve.

Figure 4 displays a set of 36 arc tiles. Figures 1(b) and 7 display mosaics constructed out of them. Each arc tile has 0, 2, 4, 6, or 8 circular arcs on it. Each arc starts at the center of the tile and ends on an edge, at one of eight possible termination points (two per edge). (Examine the rightmost tile in the bottom row of Figure 4, and note that this tile's eight arcs terminate at eight different points—the eight possible termination points.) Each termination point is the termination point of at most one arc.

The arc-tile integer program has more constraints and more binary variables than the knot-tile integer program. As one would expect, for fixed values of m and n, arc-tile IPs are much harder to solve than knot-

Figure 4: the 36 arc tiles

tile IPs. CPLEX required about five minutes for the mosaic displayed in Figure 1(b) and almost a day for the mosaic displayed in Figure 7!

Figure 8 displays a mosaic constructed from a set of shaded, modified Truchet tiles [16]. There are 504 tiles in this set, and the corresponding integer program is both enormous and very difficult to solve. A small subset of these tiles is displayed in Figure 5.

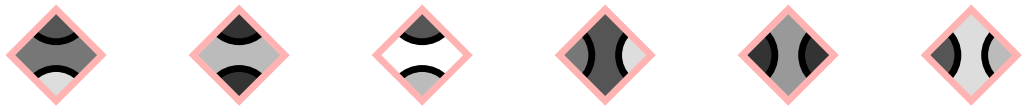

Figure 5: a subset of the 504 shaded modified Truchet tiles

And Figure 9 displays a mosaic constructed from a set of 74 square “Paul Brown” tiles, inspired by elements of the Australian artist’s pieces *The Book of Transformations* and *Chromos* [4].

4 Future Work

We believe that this area is rich, both in terms of art and mathematics. Our integer programming approach can be applied to a wide variety of sets of tiles. Not only can one experiment with the designs drawn on the tiles, but also the shapes of the tiles. In addition to the "diamonds" (which, of course, are squares rotated 45 degrees) we used in Figures 1-8 and the squares we used in Figure 9, one could use equilateral triangles, regular hexagons, or even Penrose tiles.

When the set of tiles is large, the integer programs we use to solve the tile arrangement problem can end up being quite large and difficult to solve. We could be discouraged about this, but we prefer to view the difficulty as a challenge. We look forward to trying to improve our IP formulation, to devising alternate (and hopefully superior) IP formulations, and to investigating non-IP-based approaches.

Acknowledgements

I am very grateful to the two anonymous referees. Their suggestions improved this paper considerably.

References

[1] R.A. Bosch, "Constructing domino portraits," in *Tribute to a Mathemagician*, ed. B. Cipra et al., A.K. Peters, 2004, 251-256.

[2] R. Bosch, "Opt Art," *Math Horizons*, February 2006, 6-9.

[3] R. Bosch, "Domino Artwork," `www.dominoartwork.com`, Accessed on April 24, 2007.

[4] P. Brown, "Paul Brown - art $<$ $>$ technology," `www.paul-brown.com/GALLERY/INDEX.HTM`, Accessed on April 24, 2007.

[5] E. Harshbarger, "Eric Harshbarger's LEGO pages," `www.ericharshbarger.org/lego/`, Accessed on April 24, 2007.

[6] ILOG CPLEX, "High-performance software for mathematical programming and optimization," `www.ilog.com/products/cplex/`, Accessed on April 24, 2007.

[7] K.C. Knowlton, "Representation of designs," *U.S. Patent 4,398,890* (August 16, 1983).

[8] K. Knowlton, "Knowlton mosaics," `www.knowltonmosaics.com`, Accessed on April 24, 2007.

[9] D.E. Knuth, *The Stanford GraphBase*, Addison Wesley, 1993.

[10] J. Mecier, "Jason Mecier," `jasonmecier.com/gallery.html`, Accessed on April 24, 2007.

[11] J. Salavon, "Jason Salavon — The Grand Unification Theory," `www.salavon.com/GUT/GUT.shtml`, Accessed on April 24, 2007.

[12] A. Seckel, *Masters of Deception: Escher, Dalí, & the Artists of Optical Illusion*, pages 163-178, Sterling Publishing, 2004.

[13] R.S. Silvers, "Photomosiacs: putting pictures in their place," M.S. Thesis, The Media Lab, Massachusetts Institute of Technology, 1996.

[14] R. Silvers, *Photomosaic Portraits*, Viking Studio, 2000.

[15] D. Sperber, "Installation Art, Sculpture, and Public Commissions," `www.devorahsperber.com`, Accessed on April 24, 2007.

[16] E. Weisstein, "Truchet Tiling," From *MathWorld*—A Wolfram Web Resource, `mathworld.wolfram.com/TruchetTiling.html`, Accessed on April 24, 2007.

[17] L. Wolsey, *Integer Programming*, Wiley-Interscience, 1998.

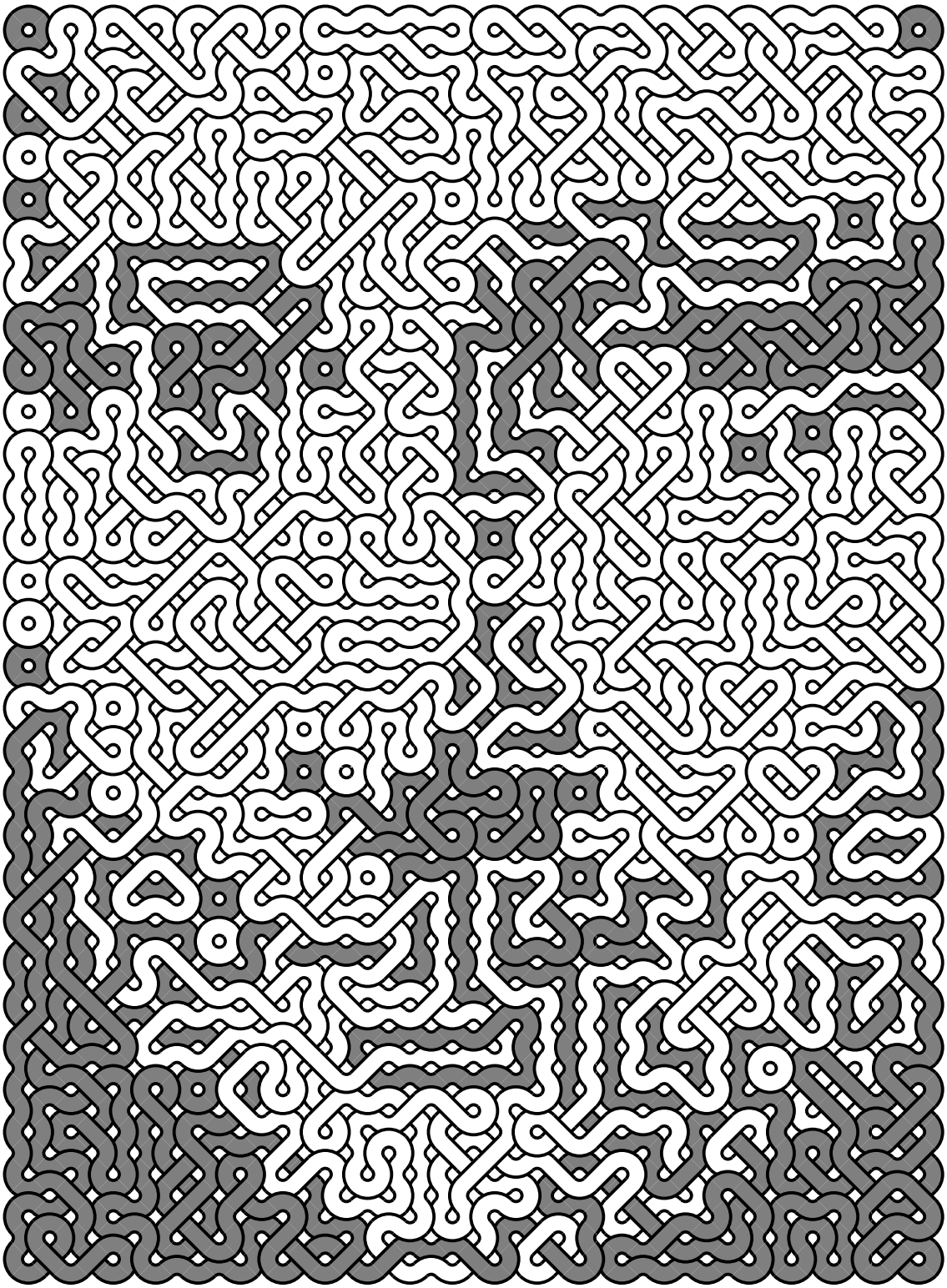

Figure 6: knot-tile mosaic ($m = 60$ and $n = 44$)

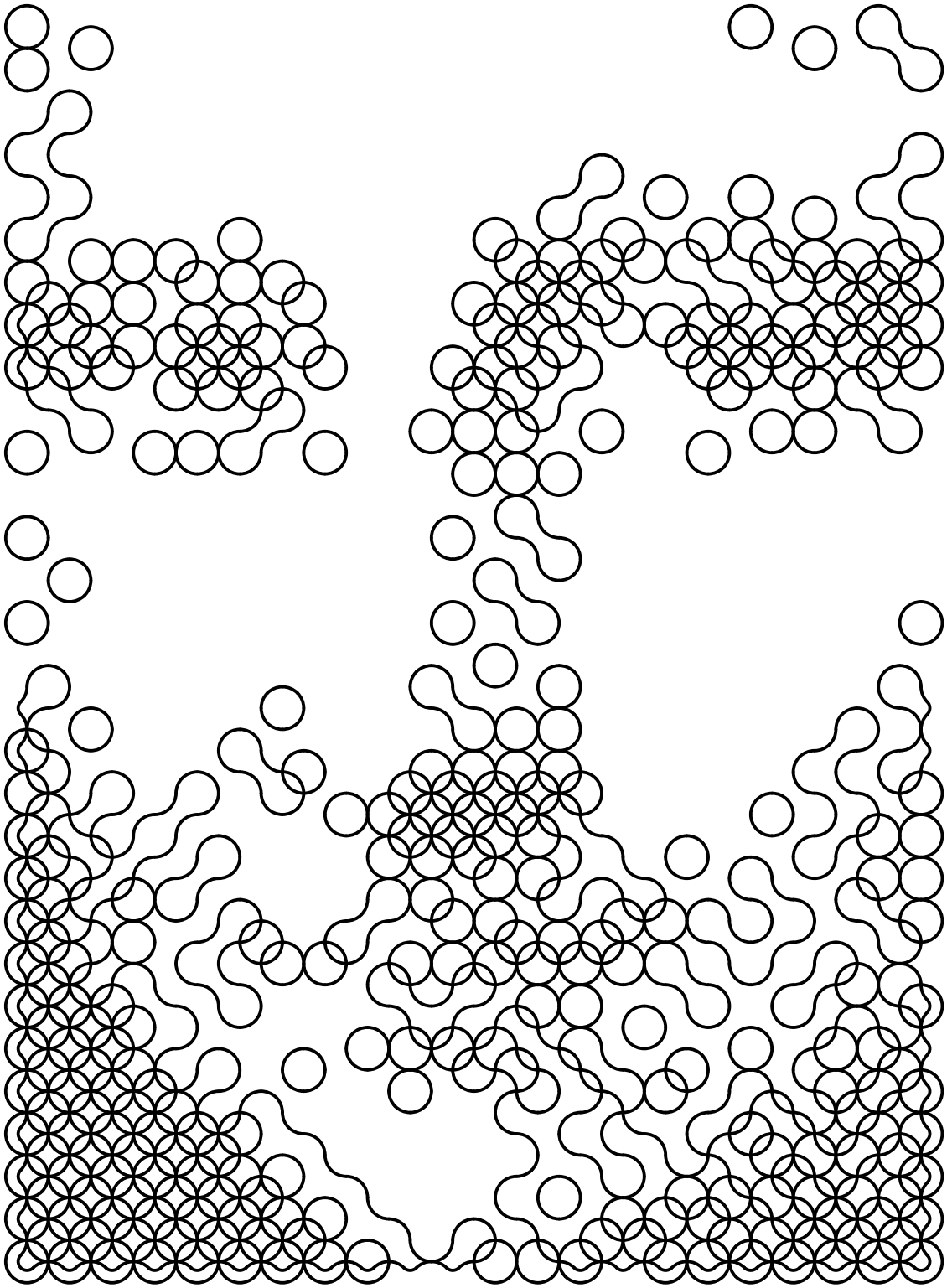

Figure 7: arc-tile mosaic ($m = 60$ and $n = 44$)

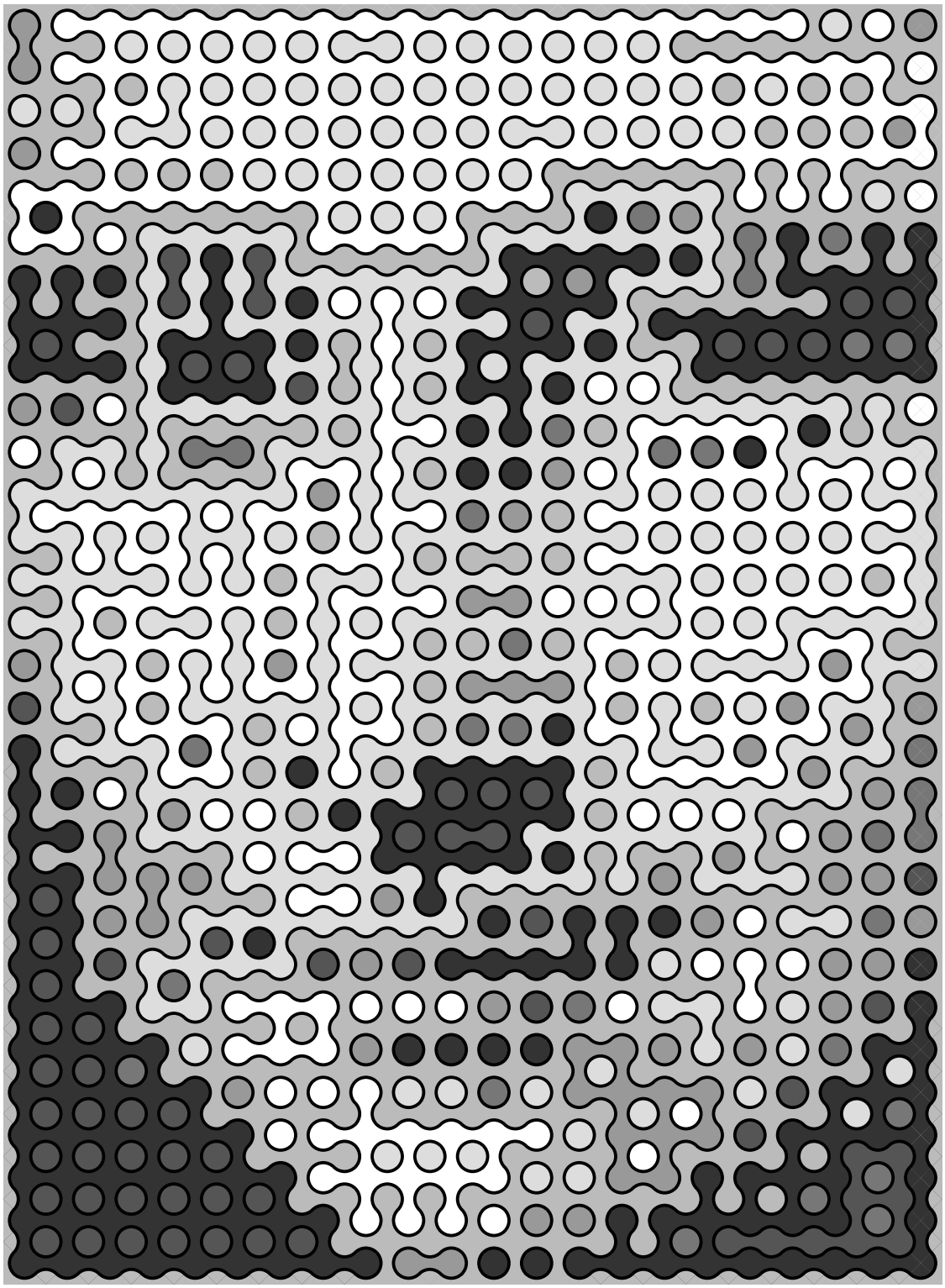

Figure 8: shaded, modified, Truchet-tile mosaic ($m = 60$ and $n = 44$)

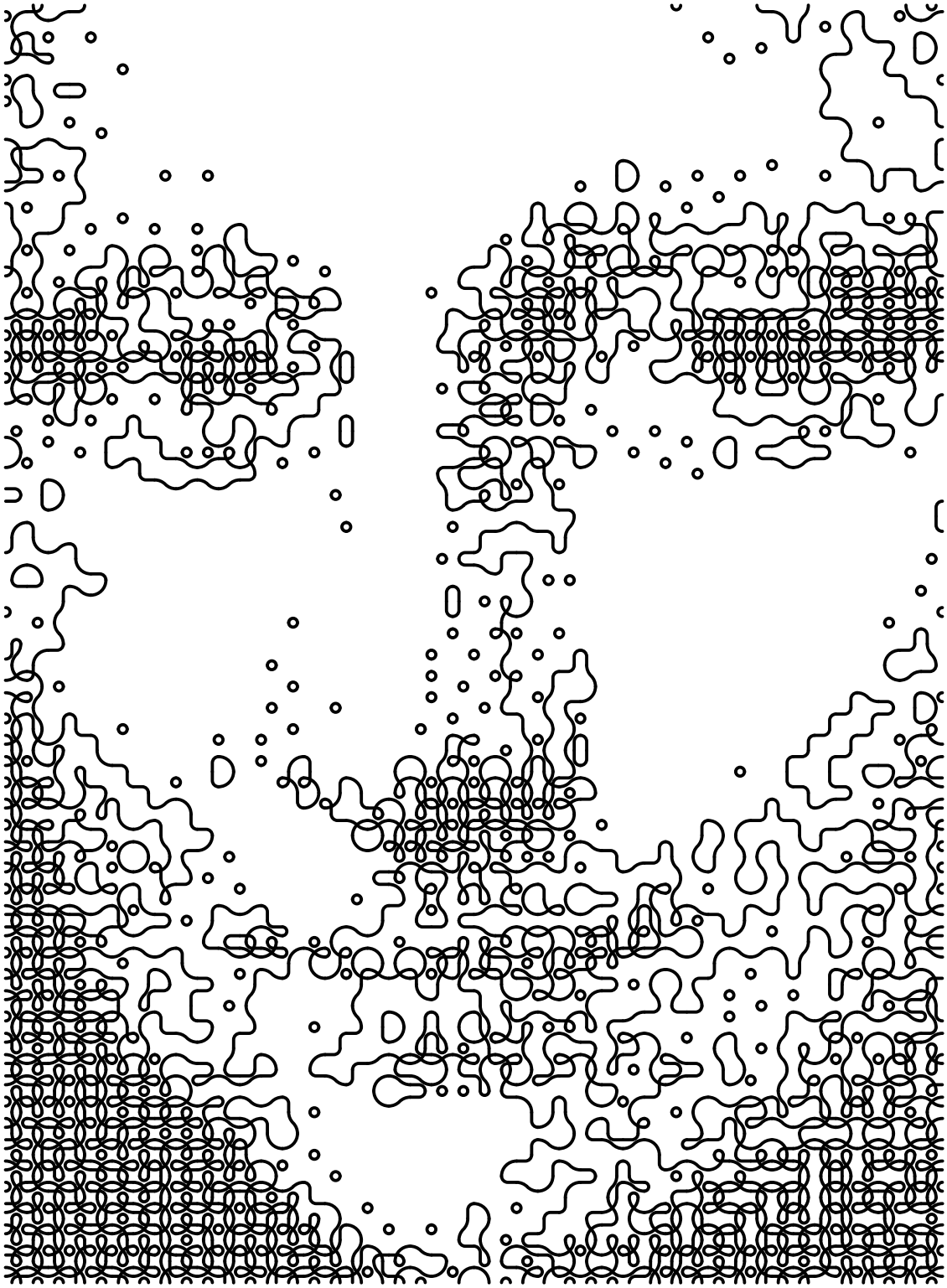

Figure 9: “Paul Brown”-tile mosaic ($m = 60$ and $n = 44$, square grid)