Animating Line-based Op Art

Tiffany C. Inglis and Craig S. Kaplan
David R. Cheriton School of Computer Science
University of Waterloo
piffany@gmail.com

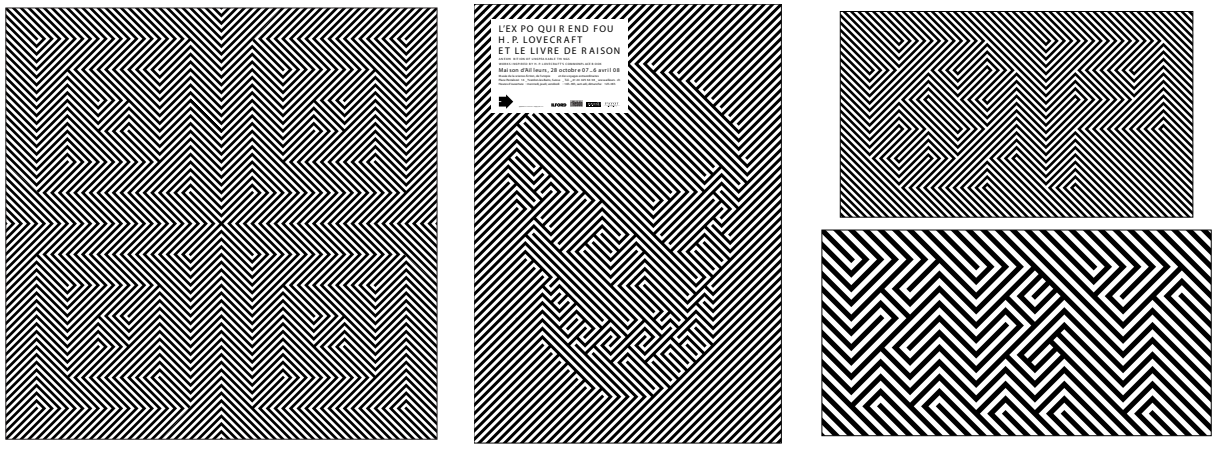
Abstract

We describe a tiling approach to animating line-based Op Art. The tiling must be seamless along the tile edges, and we seek to minimize the number of bidirectional tiles because they interfere with the Op Art illusion. We introduce several algorithms for creating these tilings, and demonstrate that an optimal tiling can be found in polynomial time. Examples of animated Op Art can be found on our website https://sites.google.com/site/tiffanycinglis/research/animating-line-based-op-art.

1 Introduction

Op Art is a form of abstract art that creates striking visual effects often from simple geometric shapes, repetitive arrangements, and highly contrasting colours. It covers a wide range of compositions including Riley's meandering black and white curves, Vasarely's colourful bulging grids, and Kitaoka's large collection of optical illusions [3, 5]. One common style of Op Art, which we call *line-based Op Art*, uses lines oriented in two orthogonal directions. Between two regions, orthogonal lines join up to form a series of the right angles that we perceive as an edge even though no edge is actually drawn; this visual illusion is called an illusory contour [7]. Figure 1 shows several examples of line-based Op Art by artists, graphic designers, and typographers. Being able to see illusory contours allows us to interpret these images as concentric squares, a skull, and letters. Our previous work [3] addressed the problem of creating a variety of Op Art compositions with lines on both square and triangular grids, and with curves. In this paper we extend this concept to animation for line-based Op Art on square grids.

A simple way to create line-based Op Art is to use a modular approach by filling a grid with tiles containing diagonal lines in one of two orientations [4, 9]. If the lines are arranged so that horizontal and vertical edges match up respectively, the resulting composition will be seamless (compare Figures 2a and 2b). Moreover, by ensuring that lines do not intersect tile corners, we can avoid artifacts such as 3-way junctions (see Figure 2c) and line ends (see



(a) (b) (c)

Figure 1: Examples of line-based Op Art include (a) Reginald Neal's *Square of Two* (top layer), 1965; (b) a poster for "An Exhibition of Unspeakable Things – Works inspired by H.P. Lovecraft's Commonplace Book" at Maison d'Ailleurs, Yverdon-les-Bains, Switzerland © by Julien Notter and Sébastien Vigne, 2007; and (c) two Op-Art-inspired fonts, Dioptical by Manolo Guerrero (top) and OPTICA Normal by House42 (bottom).

Figure 2d), and instead create a composition containing only continuous piecewise linear paths (see Figure 2a). With this tiling method, the more lines each tile contains, the more blocky the composition looks, so we want to minimize this value. In order to satisfy the seamlessness and continuity conditions, we need at least two lines in each tile where each line joins two edges, as shown in Figures 3a–3b. These tile patterns are very similar to those of Smith-Truchet tiles [1, 8] except they use straight lines instead of circular arcs. One example of a tiling with its corresponding composition is shown in Figures 3c–3d.

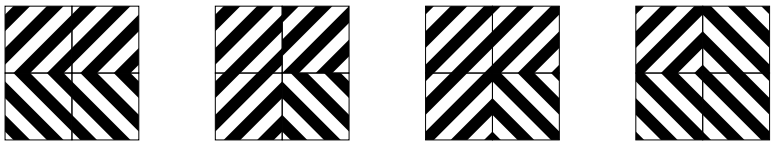
Recently, artists such as Davidope and Myoshka have experimented with animated Op Art compositions [6]. As an example, Myoshka created animated versions of Riley's *Blaze* series by growing zigzag lines from a central source point. In this paper, we are interested in animating line-based Op Art by using animated tiles containing moving lines. While the static case uses only two types of tiles, the animation will require eight different tiles to ensure seamlessness. Certain tiles are less desirable because of their change in intensity during the animation which interferes with the Op Art effect. To mitigate this problem, we introduce several algorithms then analyze the results to show various properties, including a mapping of the tiling problem to a graph problem that is solvable in polynomial time.

2 Animating Op Art

Before discussing how to create animated Op Art, let us introduce some relevant terminology. We start with a grid of tiles (see Figure 3d) where each tile contains two diagonal lines in one of two orientations. The tiles are referred to mnemonically as ***N**-tiles* and ***Z**-tiles* (see Figures 3a–3b), and each one has four *edges*: north, south, west and east. The pattern produced by arranging these tiles in a grid is called a *tiling*. Adjacent tiles share an edge, which may result in a *seam* if the patterns do not match along the edge. A tiling is *seamless* if it does not contain any seams. In the static case, any tiling created by **N**-tiles and **Z**-tiles is seamless.

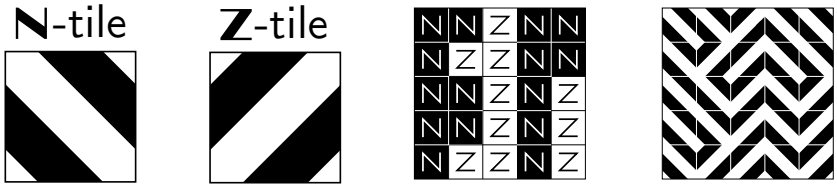
The black lines in a tiling join to form *paths*. An *outer path* is one that starts and ends on the borders of the grid. An *inner path* creates a loop in the interior of the grid. Figure 3d, for example, contains one inner path (in the shape of a caret) and ten outer paths. Each path is formed by a set of tiles, which divides the path into *segments*, and *length* of the path is defined as the number of such segments. For the tiling in Figure 3d (see also Figure 6c), the inner path has length 12.

Throughout this paper, we will use *problem* to refer to a static line-based Op Art composition that we want to animate. A *solution* would be any animated tiling with the same arrangement of **N**-tiles and **Z**-tiles as the static omposition. For a given problem, we use *solution* and *tiling* interchangeably. Later we will discuss the difference between trivial, non-trivial, and optimal tilings.



(a) (b) (c) (d)

Figure 2: Examples of tilings with (a) no seams, (b) two seams, (c) a 3-way junction, and (d) a line end.



(a) (b) (c) (d)

Figure 3: **N**- and **Z**-tiles are arranged in a grid to create line-based Op Art.

2.1 Creating animated tiles

An animated tile is created by applying some continuous transformation to the lines. We want to keep the orientation of the lines fixed, which means the lines are only allowed to be translated. In an **N**-tile, a line can either travel toward or away from the northeast corner; in a **Z**-tile, a line either travels toward or away from the northwest corner. Since each tile contains two lines and each line travels in one of two directions, there are four possible animated tiles.

For now, let both lines in each tile travel in the same direction. We call these tiles *unidirectional*. Consider the two tiles in Figure 4a with lines travelling toward the northeast corner (top) and the southeast corner (bottom). The four arrows outside indicate the *edge directions*—i.e., the direction of movement along the edges. If we create an animated tiling using these two tiles, the result (see Figures 4b–4e) would contain multiple vertical seams because the two tiles do not have matching edge directions along their vertical edges. We can try using more unidirectional tiles, but no matter how they are arranged, there will be seams in general. A simple example to demonstrate this concept is a 3×3 tiling with eight **N**-tiles surrounding one **Z**-tiles (see Figure 5a). The seamlessness condition forces the eight surrounding tiles to have lines travelling in the same direction, but for the centre tile, no matter which direction its lines move in, there will always be either horizontal seams or vertical seams.

The reason unidirectional tiles cannot create seamless tilings in general is that the tiles have a very limited set of edge directions. If we allow *bidirectional tiles*–i.e., tiles with lines travelling in opposite directions–then we have the set of eight tiles shown in Figure 5b. While it is impossible in general to create seamless tilings with unidirectional tiles, it can be done very easily with bidirectional tiles. We simply let the vertices be alternating "sources" and "sinks" so that all four edges around a source vertex are directed toward it, and those around a sink vertex are directed away from it; then we place the animated tiles matching these edge conditions whose line orientations correspond to those of the static Op Art tiling. This tiling, which we will call a *trivial tiling*, is seamless and uses exclusively bidirectional tiles. Any tiling that contains at least one unidirectional tile is called a *non-trivial tiling*.

The problem with using bidirectional tiles is that they create a somewhat disturbing flashing effect. The reason is that, while a unidirectional tile remains 50% throughout the animation, a bidirectional tile oscillates between 25% to 75% black. The flashing interferes with our ability to see the illusory contours from the original static compositions. Therefore, we want to use as few bidirectional tiles as possible to make the animation look more stable. For a given problem, a tiling that uses the minimum number of bidirectional tiles is called an *optimal tiling*. In the following sections, we will discuss how to search for a non-trivial solution, enumerate all solutions, and find an optimal tiling.

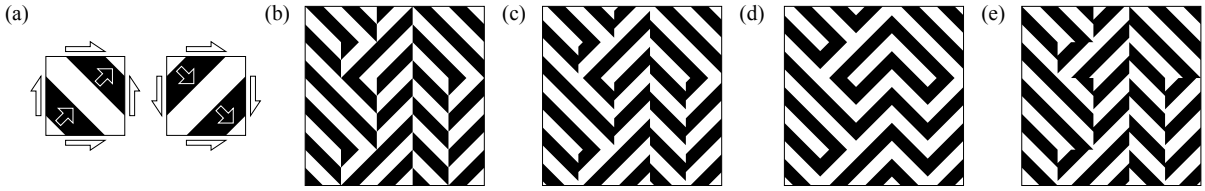


Figure 4: (a) Here are two unidirectional tiles, with the direction of movement and the edge directions both marked by arrows. (b–e) These four frames demonstrate the animation created using only unidirectional tiles.

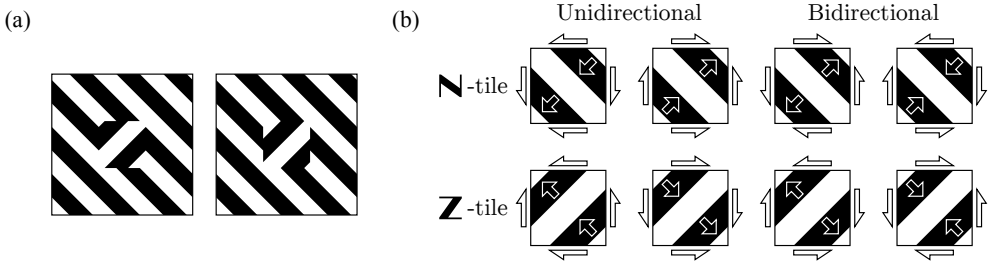


Figure 5: (a) Animating this tiling using only unidirectional tiles will always result in seams. (b) To create seamless animations, we need to use this set of eight tiles containing both unidirectional and bidirectional tiles.

3 Seam-moving algorithm

Our seam-moving algorithm takes a trivial tiling to a non-trivial tiling by removing seams one at a time. The basic operation moves a single seam into a neighbouring edge via an *edge flip*. For example, in Figure 6a, the **N**-tile has a seam on its east edge (indicated by the pair of opposing arrows). By performing an operation f_{NE} that flips the direction of the north and east edges inside the tile, the seam is moved to the north edge. Similarly, f_{SW} can swap seams between the south and west edges. As for a **Z**-tile, f_{NW} and f_{SE} are the two valid seam-moving operations.

Notice that an edge flip is always between two edges joined by a black line, which means a series of flips will move a seam along a path, as shown in Figure 6b. A seam on an outer path can be moved until it is on the grid border, thereby removing it. As for seams on an inner path, we can specify one edge on the path and move all seams onto it, which leaves at most one seam on the path. Now each inner path, if it does not contain any seams, its movement must be one of contraction or expansion. If it contains exactly one seam, then the path would be contracting on one side of it and expanding on the other, but since the two parts cannot join up elsewhere without creating another seam, having exactly one seam in an inner path is impossible. The contradiction implies that our approach will always remove all the seams in an inner path. Therefore every tiling can be made seamless.

If the seams are removed one by one, we waste a lot of time revisiting edges. Instead we will traverse each path exactly once while "pushing" the seams along. The first step is to calculate a distance transform (see Figure 6c) which labels the edges in each path in increasing order. Note that we start the distance transform calculation from the top and left borders of the tiling. Then we traverse the edges of each path in increasing order of the labels. At edge i, check if it is a seam. If it is, flip it along with edge $i+1$, resulting in one of two scenarios—either the seam moves from edge i to edge $i+1$ or two seams are eliminated at once. Otherwise, if edge i is not a seam, do nothing and move onto edge $i+1$. This modified algorithm contains two steps: computing the distance transform and traversing each path, which takes $\mathcal{O}(mn)$ time overall for an $m \times n$ tiling.

4 Properties of the tiling solutions

We are interested in studying the properties of all solutions for a given problem, in particular, the optimal tilings. To find an optimal tiling, we find the corresponding binary integer programming problem that seeks to minimize the number of bidirectional tiles used under some edge constraints, and solve it using MATLAB's `bintprog` function. As for enumerating all the solutions, we use a divide-and-conquer approach: an $n \times n$ problem (assume n is a power of 2) can be split recursively into four $\frac{n}{2} \times \frac{n}{2}$ subproblems until we reach the base case with 2×2 problems which are easily solved by brute force. The merge step involves stitching four $\frac{n}{2} \times \frac{n}{2}$ solutions together and checking if they create any seams along their shared edges. If not, the tiling is added to the solution set.

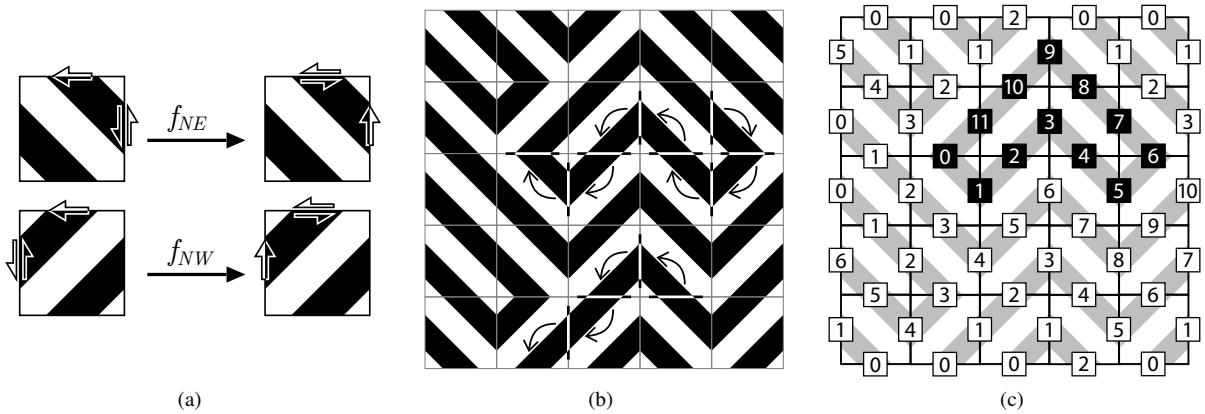


Figure 6: (a) An edge flip swaps the direction of two edges in a tile. (b) The seam-moving relies on a series of edge flips along the paths to remove all seams. (c) The modified seam-moving algorithm uses a distance transform to determine the order of the edge flips.

4.1 Cardinality of solution sets

When enumerating solutions for random 4×4 problem instances, we noticed that the number of solutions was always a power of 2, starting from 256. On closer examination, we realized that this has to do with the number of paths. In a 4×4 tiling, there are exactly 8 outer paths because each one joins 2 border edges and there are a total of 16 border edges (see Figure 7a). Since the seamlessness constraint forces each path to move in one of two directions, there are at least $2^8 = 256$ solutions. A tiling may also have inner paths. Each inner path doubles the number of solutions. Since up to 5 inner paths can exist in a 4×4 problem, the maximum number of solutions is $2^{(8+5)} = 8192$ (see Figure 7b).

For an $m \times n$ problem, the number of solutions is always 2^k, where $k \geq m+n$, since there are exactly $m+n$ outer paths. The number of inner paths ranges from 0 to $L(m,n)$, given recursively by

$$L(m,n) \quad = \quad \begin{cases} L(m-1,n-1) + \left\lceil \frac{m-1}{2} \right\rceil + \left\lceil \frac{n-1}{2} \right\rceil - 1 & \text{if } m,n > 0 \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

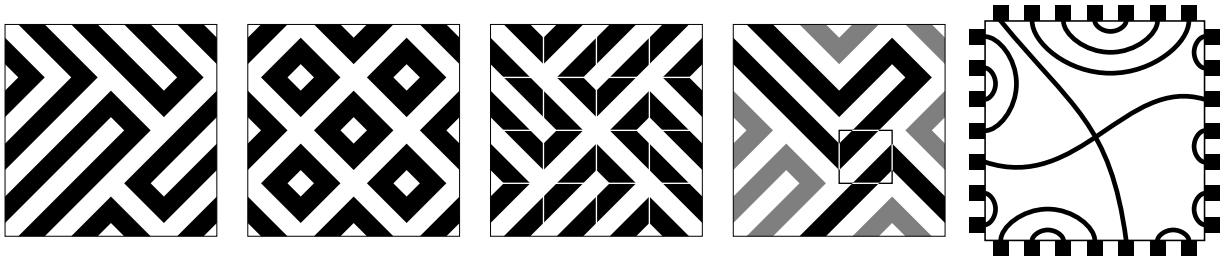
Now we can easily construct any solution for a given problem. Consider the tiling in Figure 7a for example. It has 2^8 solutions, and each solution can be expressed uniquely as a binary string of length 8. Let $(0,0,0,0,0,0,0,0)$ denote a trivial solution. Then $(1,0,0,1,1,0,0,0)$ represents the solution we get taking the trivial solution and flipping the directions of the first, fourth and fifth path so that they travel in opposite directions.

4.2 A lower bound on the cost

Define the *cost* of a tiling as the number of bidirectional tiles used. The optimal solution is the tiling with the lowest cost. For a given problem, we want to analyze the range of possible costs and find a good lower bound. As a test, we generated the complete set of solutions for the 8×8 problem in Figure 8e and found that the costs ranged from 22 to 64 (from the trivial tiling).

To provide a lower bound for the number of bidirectional tiles needed, let us look at a bidirectional tile in relation to the paths. A bidirectional tile contains two segments that are either moving toward each other or away from each other. We can think of it as a white segment that is either contracting or expanding. So the cost of the tiling is the number of such white segments. These white segments join to form white paths that either contract or expand; for brevity, we will call them *bidirectional paths* and define their lengths to be the number of segments they contain.

The cost of a tiling is the sum of the lengths of all bidirectional paths. For a given problem, we can find a lower bound for the cost by identifying paths that must be bidirectional among all the possible tilings. First, a white path that reaches a line end must be bidirectional (see Figures 8a and 8b) because the black path surrounding it has to move uniformly outward or inward. Another scenario in which a white path is forced to be bidirectional is near a 3-way junction (see Figure 8c). In this case, no matter which directions the blacks paths move, at least one of the three white paths must be bidirectional. On the other hand, a 4-way junction can be arranged as shown in Figure 8d to avoid any bidirectional paths.



(a) (b) (c) (d) (e)

Figure 7: (a, b) A 4×4 problem contains at least 8 paths and at most 13. (c) Every path in this tiling has a even length. (d) The black paths are adjacent because the highlighted tile contains segments from both paths. (e) It is impossible to have a problem whose solutions only have odd costs, because constructing it will lead to two paths intersecting.

To find the lower bound on the cost, first locate all the line ends and 3-way junctions as shown in Figure 8e. For each line end, draw the induced bidirectional path. For 3-way junctions, we can strategically place the paths to get a better lower bound. For example, between two adjacent 3-way junctions, instead of drawing two bidirectional paths, we can simply make the adjoining path bidirectional; for an isolated 3-way junction, we would choose the shortest of the three incident paths to be bidirectional. Figure 8e shows an arrangement of bidirectional paths with a total length of 22, which is exactly the cost of the optimal tiling.

4.3 The parity of tiling costs

When plotting histograms of solution costs for various 4×4 problems, we noticed that some problems only have solutions with even costs. We compared these problems to others and found that these problems all contain only paths of even length, such as the one shown in Figure 7c. Why does having even-length paths lead to only solutions with even costs? To answer this question, recall that it is possible to construct any solution by taking a trivial solution and flipping a subset of the paths. First of all, in a 4×4 problem, a trivial solution has a cost of 64, which is even. To prove that all paths have even costs, we need to show that flipping a path with even length will always change the cost by an even amount.

We say two paths are *adjacent* when there are tiles that contain segments from both. A path can also be adjacent to itself if two of its segments appear in the same tile. Figure 7d shows two adjacent paths in black. We say they *share* a segment because there is one tile (highlighted) that contains a segment from each path. Since each tile contains two segments, a path always shares an even number of segments with itself.

Now consider an even-length path. It shares an even number of segments with itself and has k remaining segments, where k is even. Of these k segments, k_b are in bidirectional tiles while the other $k - k_b$ are in unidirectional tiles. When the path is flipped, the bidirectional tiles become unidirectional, and vice versa. The cost associated with this path goes from k_b to $k - k_b$. Since the difference $k - 2k_b$ is even, path-flipping preserves the parity of the cost. Therefore, if all the paths are of even lengths and the trivial solution has an even cost, then every solution has an even cost.

Is it possible to have problems with only odd-cost solutions? If such a problem exists, its trivial solution must have an odd cost—this is only true for $m \times n$ problem where m and n are both odd—and all its paths must have even lengths. Our attempts to construct such a problem failed repeatedly, and in the end, we found a simple proof for why the construction is impossible. Figure 7e shows a 7×7 problem. Each border edge contains 7 path ends (marked by the black squares) where outer paths start and end. It can be shown that for a path to have an even length, it must be either an inner path, an outer path that starts and ends on the same border edge, or an outer path that start and ends on opposite border edges. Since there are an odd number of paths ends on each border edge, by the Pigeon Hole Principle, at least one path must connect to the opposite border edge. Thus one path would join the north-south borders while another one joins the west-east borders. But this contradicts with the fact that **N**-tiles and **Z**-tiles cannot create intersections (i.e. 4-way junctions). Hence there are no problems for which only odd-cost solutions exist.

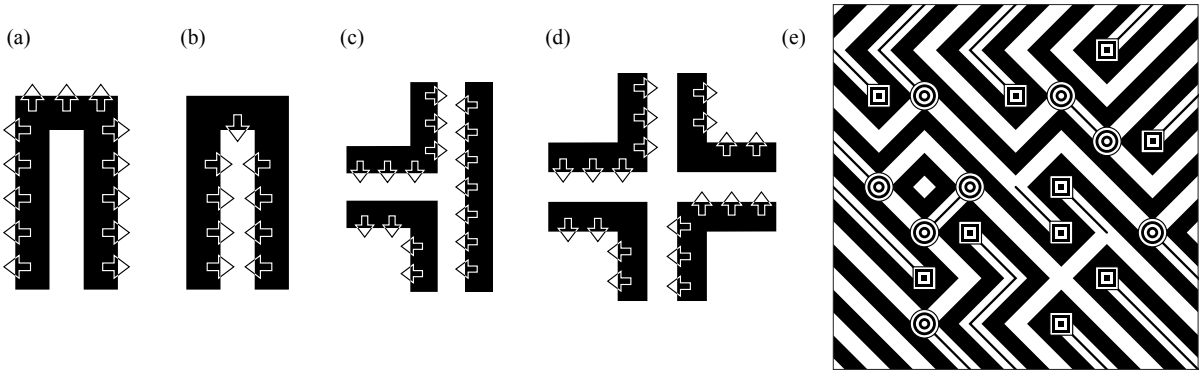


Figure 8: (a–d) Line ends and 3-junctions create bidirectional paths that increase the tiling cost, whereas 4-way junctions can be arranged to avoid bidirectional paths. (e) To determine a lower bound on the cost, first mark the line ends (squares) and 3-way junctions (circles). Then find a set of bidirectional paths with minimum total length.

5 Computational complexity

Finding an optimal solution using binary integer programming is slow for large problems. We want to know if faster algorithms are possible or if the problem is just NP-hard. Consider our problem in terms of contracting and expanding paths. Since the concept of contracting and expanding is not well-defined for outer paths, we first close them all (as shown in Figure 9a) to form loops. For each outer path, there are two ways to close it, but for our purpose it does not matter which way we choose.

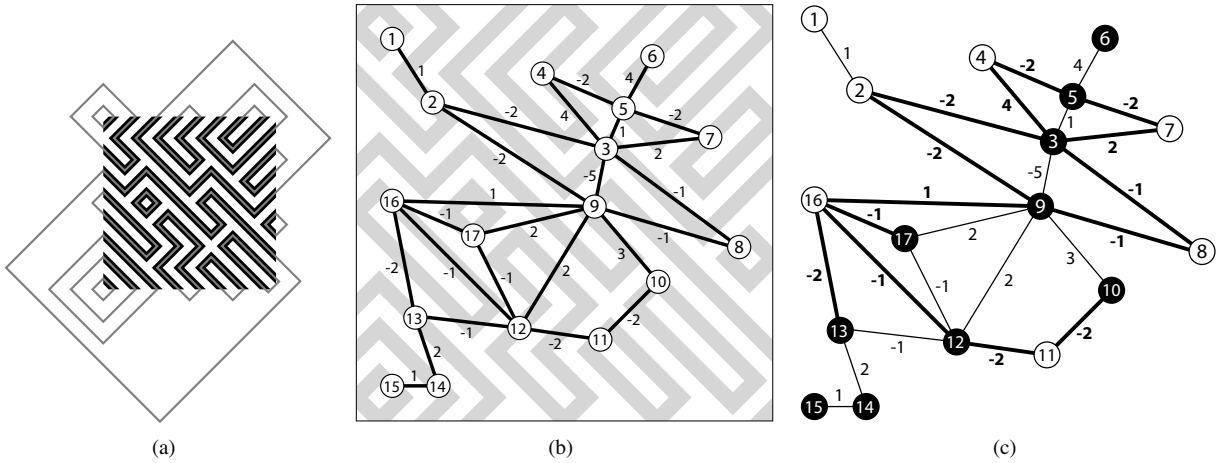
Similar to the concept of adjacent paths, adjacent loops are those that share segments in tiles. For every pair of adjacent loops, it either satisfies the containment relationship (i.e. one loop contains the other) or is disjoint (i.e. the loops do not contain one another). The loops can either move in the same direction (i.e. both contracting or both expanding) or opposite direction (i.e. one contracting and one expanding). If the loops satisfy the containment relationship, then moving in the same direction will not change the cost, while moving in opposite directions will increase the cost by the length of their shared boundary. On the other hand, if two loops are disjoint, then moving in the same direction will increase the cost also by the length of the shared boundary, whereas moving in opposite directions will not change the cost.

This scenario can be described as a graph problem (see Figure 9b). Consider a weighted graph $G = (V,E)$ in which each node represents a loop, and two nodes are joined by an edge if they represent adjacent loops. Let b_{ij} be the length of the shared boundary between loop i and j. Then the edge joining nodes i and j has a weight of b_{ij} if one loop contains the other, and $-b_{ij}$ if they are disjoint. It turns out that finding the optimal tiling is equivalent to finding the weighted min-cut of this graph. We will prove their equivalence by expressing the cost of a tiling in terms of the weight of the corresponding graph cut and showing that minimizing one also minimizes the other. Let C be the cost of a tiling. $C = C_{\text{self}} + C_{\text{pair}}$ where C_{self} is the contribution from the edges each loops shares with itself, and C_{pair} is from the edges shared between adjacent loops. Since C_{self} is fixed for a given problem, minimizing the total cost requires minimizing only C_{pair}.

Now suppose we have a graph cut that partitions the nodes V into subsets V_1 and V_2. The nodes in V_1 and V_2 represent respectively loops that contract and those that expand. Let $E_{\text{CUT}} \subseteq E$ be the set of edges connecting V_1 to V_2. Then C_{pair} can be expressed as

$$C_{\text{pair}} \quad = \quad \sum_{\{e \in E_{\text{CUT}} : w(e) > 0\}} w(e) - \sum_{\{e \in E - E_{\text{CUT}} : w(e) < 0\}} w(e), \tag{2}$$

where the first term is the contribution from loops that satisfy the containment relationship and move in opposite directions, while the second term is the contribution from disjoint loops that move in the same direction.



(a) (b) (c)

Figure 9: (a) Close all outer paths to form loops. (b) Then transform the tiling problem into a graph problem. (c) Finding the weighted min-cut (indicated by the thicker edges and black and white nodes) of this graph is equivalent to finding the optimal tiling.

Let S^- be the sum of the negative edge weights in the graph $G = (V,E)$ and S_{CUT} be the sum of the edge weights in the graph cut E_{CUT}. That is,

$$S^- = \sum_{\{e \in E : w(e) < 0\}} w(e) \quad \text{and} \quad S_{\text{CUT}} = \sum_{e \in E_{\text{CUT}}} w(e). \tag{3}$$

Then we can simplify the expression for C_{pair} to

$$\begin{aligned} C_{\text{pair}} &= \sum_{\{e \in E_{\text{CUT}} : w(e) > 0\}} w(e) - \left[S^- - \sum_{\{e \in E_{\text{CUT}} : w(e) < 0\}} w(e) \right] && (4) \\ &= \sum_{\{e \in E_{\text{CUT}} : w(e) > 0\}} w(e) + \sum_{\{e \in E_{\text{CUT}} : w(e) < 0\}} w(e) - S^- && (5) \\ &= \sum_{e \in E_{\text{CUT}}} w(e) - S^-, && (6) \\ &= S_{\text{CUT}} - S^-, && (7) \end{aligned}$$

Hence C_{pair} is minimized when S_{CUT} is minimized. Figure 9c shows the graph cut that corresponds to an optimal tiling.

Finding the weighted min-cut for a graph with both positive and negative weights is equivalent to a weighted max-cut problem with only non-negative weights. Normally, this mapping would imply that the problem is NP-hard. However, because our graph is derived from a tiling where edges represent loop adjacency, it is planar, and therefore the problem can be solved in polynomial time [2].

6 Conclusion and future work

For a static Op Art tiling consisting of two tile types, we found a set of animated tiles that can always produce a corresponding seamless animation. We described various algorithms for arranging these tiles and analyzed the solution space for interesting mathematical properties. If the goal is to minimize the number of bidirectional tiles, then the tiling problem can be mapped to a weighted max-cut problem for a planar graph, which is solvable in polynomial time.

The presence of birectional tiles has a great impact on the animated composition. Even when they are small in number, the flashing effect of the bidirectional tiles is still much more pronounced than the illusory contours in the static image, which suggests that our design should be based around the placement of these tiles. On the other hand, we wish to create designs that take advantage of the fact that bidirectional tiles oscillate in intensity. For example, it may be possible to create animations that alternate between two greyscale images by controlling the intensity changes more precisely, similar to figurative mosaics constructed from flexible Truchet tiles [1].

References

[1] Robert Bosch and Diane Colley. Figurative mosaics from flexible Truchet tiles. *Journal of Mathematics and the Arts*, 0(0):1–20, December 2012.

[2] F. Hadlock. Finding a maximum cut of a planar graph in polynomial time. *SIAM Journal on Computing*, 4(3):221–225, 1975.

[3] Tiffany C. Inglis and Craig S. Kaplan. Generating Op Art lines. In *Proceedings of the International Symposium on Computational Aesthetics in Graphics, Visualization, and Imaging*, CAe '11, pages 25–32, New York, NY, USA, 2011. ACM.

[4] Slavik Jablan and Kristóf Fenyvesi. The Vasarely Playhouse: Look and combine! In *Proceedings of Bridges 2011: Mathematics, Music, Art, Architecture, Culture*, pages 633–636. Tessellations Publishing, 2011.

[5] Akiyoshi Kitaoka. *Introduction to Visual Illusions*. Tokyo: Asakura-shoten, July 2010.

[6] Olly Payne. Op Art animation. `http://www.op-art.co.uk/op-art-gallery/op-art-animation`. Accessed: 31/01/2013.

[7] Susan Petry and Glenn E. Meyer. *The Perception of Illusory Contours*. Springer, 1987.

[8] Clifford A. Pickover. Picturing randomness with Truchet tiles. *Journal of Recreational Mathematics*, (21):256–259, 1989.

[9] Reza Sarhangi. Modules and modularity in mosaic patterns. *Journal of the Symmetrion (Symmetry: Culture and Science)*, 19(2–3):153–163, 2008.