Symmetry: Culture and Science
Vol. 26, No. 2, 203-216, 2015

SECTION TITLE

A SYSTEM FOR GENERATING PAPER SLICEFORM ARTWORK

Yongquan Lu, Erik D. Demaine

Address: MIT Computer Science and Artificial Intelligence Laboratory
32 Vassar St., Cambridge, MA 02139, USA
E-mail: yqlu@mit.edu
Address: MIT Computer Science and Artificial Intelligence Laboratory
32 Vassar St., Cambridge, MA 02139, USA
E-mail: edemaine@mit.edu

Abstract: *Sliceform paper art is an art form where long strips of paper are cut, folded and slotted together to form intricate geometric configurations. We present a system that streamlines the design process for such artwork, which are conventionally highly time-consuming to assemble. Main features of the system include a powerful user interface for specification of custom pattern motifs and an algorithm that traces patterns in a polygonal tile-based geometric configuration to compute strip dimensions. We exhibit some representative results generated by our system.*

Keywords: sliceform, geometry, paper, art, symmetry.

1. INTRODUCTION

Sliceform paper artwork is an artform where long strips of paper are cut, folded and slotting together to form geometric configurations. Historically, paper sliceforms have omitted the folding step and focused on 3D shapes; the technique specifically discussed in this paper was first pioneered by artists such as Chris Palmer and Jeff Rutzky (2009), and have since been popularized by others such as Christiane Bettens. Fig. 1 shows a representative example. While the physical results are

beautiful, assembling a single piece can takes 8 hours or more, if tasks like calculating strip dimensions are done manually. To streamline the design process and promote the art form, we set out to develop a system that automates away these manual tasks.

Figure 1: Clockwise from top left: Crystalline (2014), Festival (2010), Garden (2010), Iridescence (2011), Constellation (2013) and Girih (2010). All pieces were created by the first author.

Since this medium involves weaving strips in intricate ways, it is natural to build on existing visualization techniques for designing and composing Islamic star patterns. Much literature has been written on the subject; in particular, our process is based

on Kaplan's approach (Kaplan, 2000), where geometrical configurations are formed by inscribing polygonal tilings with geometric motifs. While much of the existing artwork is based on traditional Islamic motifs, in theory there is no such restriction.

However, existing techniques to design Islamic star patterns only span the first two steps of the paper sliceform design process (see Figure 2) — inscribing motifs in tiles creates a visual illusion of long strips that weave in and out of each other, but the system has no notion of a contiguous strip. Furthermore, an ideal system should also be able to automatically render these contiguous strips in a form that can be printed and cut out for assembly easily.

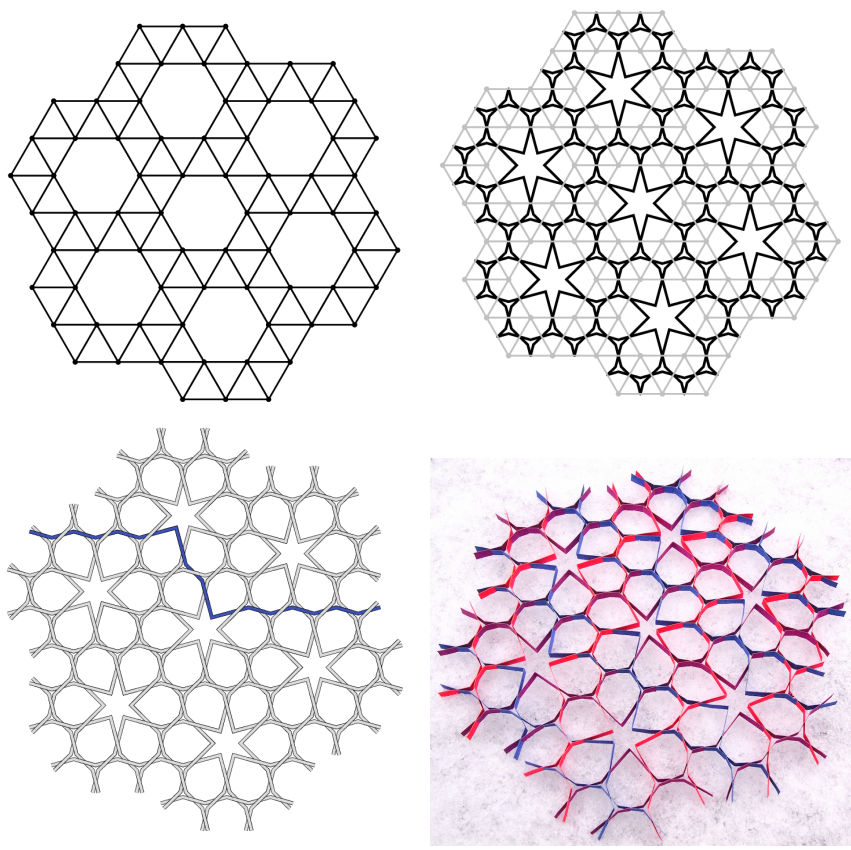


Figure 2: Four stages of paper sliceform design, illustrated via Pentille (2015). First, a tiling of polygonal tiles is specified (top left). Next, patterns are inscribed in each polygonal tile (top right). These patterns are then traced across tile boundaries to form contiguous strips (bottom left). Finally, they are rendered, printed and assembled to form the artwork (bottom right).

This paper presents a system called **Wallpaper** (see Figure 3) that streamlines each step of the design process, from polygon generation and pattern specification to strip conversion and finally SVG rendering. The implementation of several design elements (such as the preset tiling options) take design cues from previous visualization tools like Kaplan's Taprats applet (2000), but other components of the system are completely new.

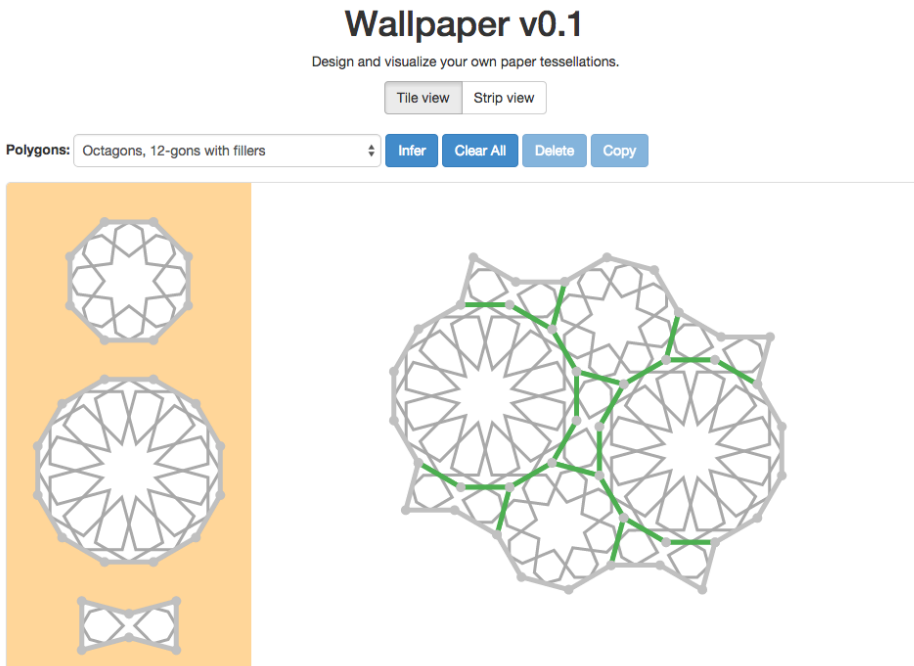


Figure 3: A screencap of the main Wallpaper user interface.

In particular, we focus especially on two novel components of the application:

1. a flexible GUI for users to specify pattern motifs with customizable symmetric properties
2. an algorithm that traces patterns across polygonal boundaries to determine relative segment lengths and intersection positions

While Wallpaper was specifically built to generate this genre of artwork, we will also demonstrate how subcomponents of the application turn out to be accessible pedagogical tools that illustrate concepts related to tilings and symmetry.

2. SHAPE GENERATION

Wallpaper's primary interface (Figure 3) consists of a canvas, where polygonal tiles can be assembled, and a palette on the left, consisting of the currently available tiles. Adding new tiles to the canvas can be accomplished by dragging and dropping from the palette.

Inside the canvas, clicking on edges snaps the tiles together; clicking on them again breaks them apart. Copying and deleting functionality is available for easy assembly of large tessellations.

We were able to offer a large number of preset palettes, corresponding to different tilings, via a dropdown. These preset palettes include Archimedean tilings and others commonly found in Islamic geometric patterns as catalogued by Abas and Salman (1995) or Bourgoin (1973). Figure 4 below shows various sample tile assemblies — the selection demonstrates that it is possible not only to create periodic tilings by regular polygons, but also tilings incorporating irregular polygons or aperiodic tilings.

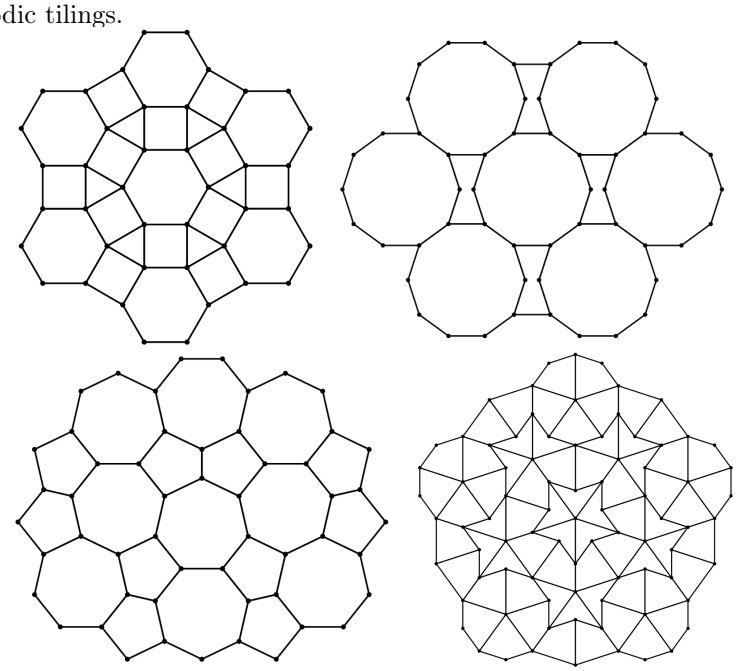


Figure 4: From left to right: Wallpaper assemblies of a 3-4-6-4 Archimedean tiling, a tiling of decagons and hexagonal fillers, an almost-regular tiling by heptagons and pentagons and a Penrose kite-and-darts tiling.

The user is also given the option of creating new custom tiles via dragging vertices, or specifying an array of internal angles and side lengths. This feature enables Wallpaper to support novel tilings we may not have anticipated beforehand.

3. PATTERN DESIGN

Our system allows users to specify the patterns to inscribe in each polygonal tile. Section 3.1 presents support for existing techniques to reconstruct common Islamic motifs, while section 3.2 presents our original interface for flexible construction of custom patterns with variable symmetric properties.

3.1. Stars, rosettes and inferred motifs

As acknowledgement of this technique's origins in Islamic star patterns and Moroccan zillij artwork, three types of common motifs are offered by default for regular polygons — stars, rosettes, and extended rosettes.

Wallpaper incorporates the geometric constructions of these motifs first worked out by Lee (1995). Here, they are all parameterized by the contact angle as well as the motif depth (the number of edges spanned by a single strip); the user can smoothly vary these by dragging sliders.

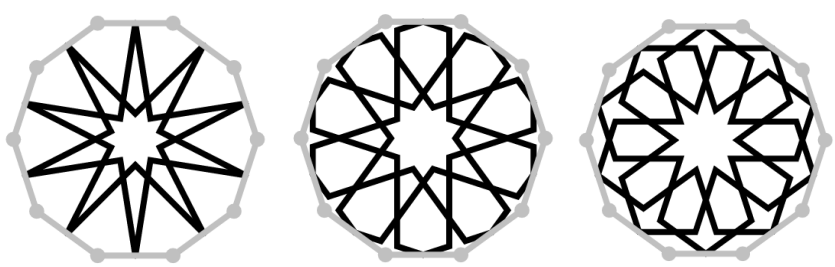


Figure 5: From left to right: parametrized star, rosette and extended rosette motifs.

We also implement a greedy inference algorithm first proposed by Hankin that extends rays from each edge of the tile. This creates an aesthetically pleasing design that fills in blank spaces in the geometry dynamically based on boundary conditions (see Figure 6).

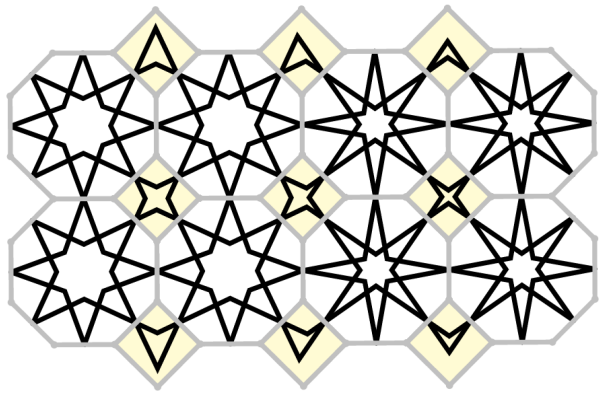


Figure 6: The shaded square tiles do not have a fixed motif, but their inscribed patterns are dynamically deduced from the contact angles of the patterns in the polygons around it.

3.2. Custom patterns

After much experimentation, we also built a custom pattern editor that allows users to specify their own inscriptions and overlay them on tiles. This is most useful for non-regular polygons (e.g. Penrose tiles), but also allows for a high degree of specificity for regular polygons.

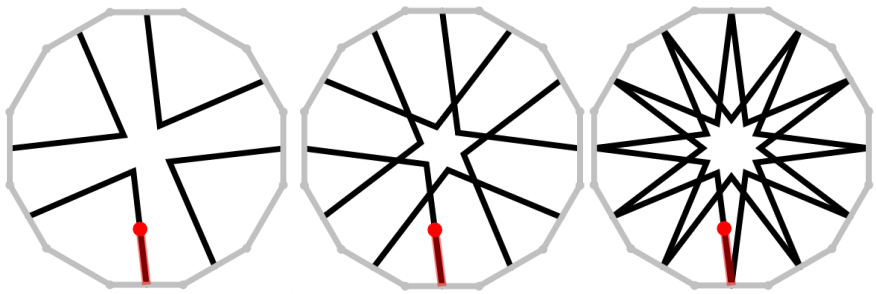
The user begins by specifying:

1. `interval`: the edges the desired pattern will apply to
2. `span`: how many edges a particular pattern spans
3. `offset`: offset of the pattern on the start and end edge
4. `symmetry`: symmetry of the pattern (mirror and/or rotational)
5. `isFlipped`: Boolean for an edge indicating if the pattern should be flipped

The program then allows the user to graphically manipulate the pattern via dragging a handle. The side by side comparisons in Figure 7 and 8 demonstrate the effect of some of the parameters.

The user is also able to designate more than one pattern on each tile, as well as increase the number of degrees of freedom for each pattern (see Figure 9). In this

way this system is fully general and enables the user to easily create a wide variety of inscribed patterns of any desired symmetry group.

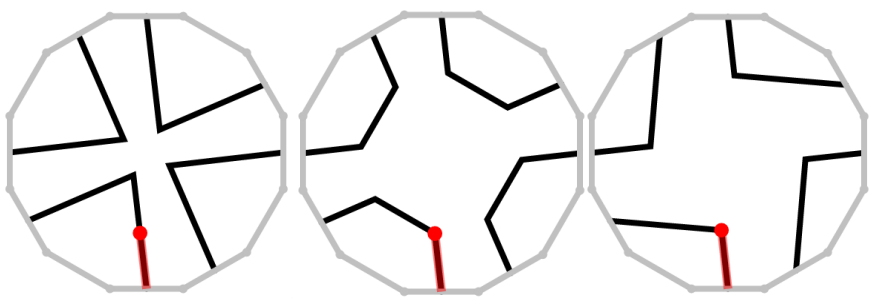


(a) `interval` = 3, `span` = 2

(b) `interval` = 2, `span` = 3

(c) `interval` = 1, `span` = 2

Figure 7: Effect of varying the `interval` and `span` parameters.

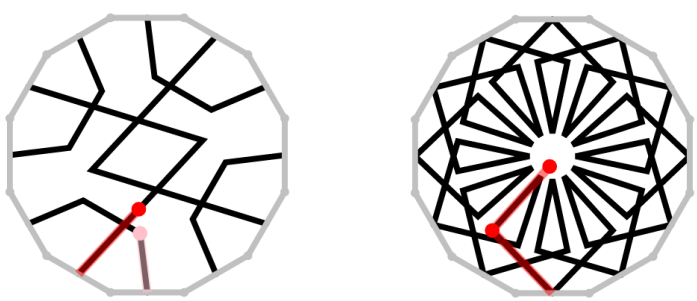


(a) Mirror symmetry uncropped

(b) Mirror symmetry cropped

(c) Without mirror symmetry

Figure 8: The three symmetry operations available



(a) Multiple patterns.

(b) Multiple degrees of freedom.

Figure 9

Note that these tiles do not have any intrinsic notion of their symmetry group; this is imposed by the user when the user specifies the parmeters `interval`, `span` and `symmetry`. We have found that such a paradigm is much more flexible than pre-defining symmetry options when initializing the tile — for example, under this approach, we may inscribe motifs with apparent 5-fold symmetry into the almost regular pentagons seen in Figure 4, even though they only have 1 axis of mirror symmetry.

4. A PATTERN TRACING ALGORITHM

After composing polygonal tiles and inscribing patterns, we need to trace the orbit of these patterns across polygonal boundaries so as to recreate them as physical paper strips. In this section the term *pattern* has a very specific meaning: a piecewise continuous line segment confined to the interior of the polygonal tile that starts and ends on the edge of the tile. Typically, tiles will have more than one pattern arranged in some symmetric manner (i.e. a *motif*).

In our implementation, each pattern stores relevant metadata about its geometry: the edges it starts and ends on, as well as its incident angle and relative position on these edges. It also keeps track of its internal geometric structure as a list of segments, where each segment is in turn a list of lengths between joints or intersections. See Figure 10 for an example.

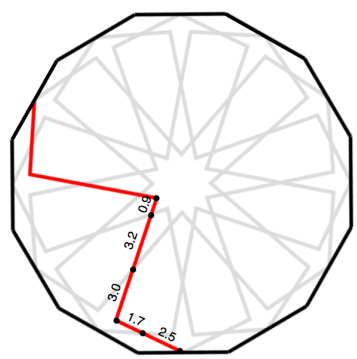


Figure 10: This rosette motif consists of 12 symmetric patterns. The highlighted pattern stores the list of list of lengths corresponding to its internal structure from start to end edge, namely `[[2.5,1.7],[3.0,3.2, 0.9],[0.9,3.2,3.0],[2.5,1.7]]`.

The first step in tracing patterns across polygonal boundaries is deciding which patterns at the boundary are contiguous and should be treated as part of the same strip. Our matching procedure works as follows: for each pattern p incident on an edge, the algorithm filters out the patterns from the other corresponding edge to those with relative position compatible with p. If the filter is empty, the pattern is not matched. If the filter returns a unique pattern, they are matched.

If there are multiple patterns originating from the same position, the algorithm picks the one with incident angle closest to that of p. It also checks that this is mutual, i.e. that p has incident angle closest to the other edge as well. If this is true, the two patterns are matched; otherwise, p is unmatched.

Note that our checks ensure that contiguity is a symmetric relation, such that even when the user joins two edges with incompatible pattern interfaces, the results match our visual intuition (see Figure 11).

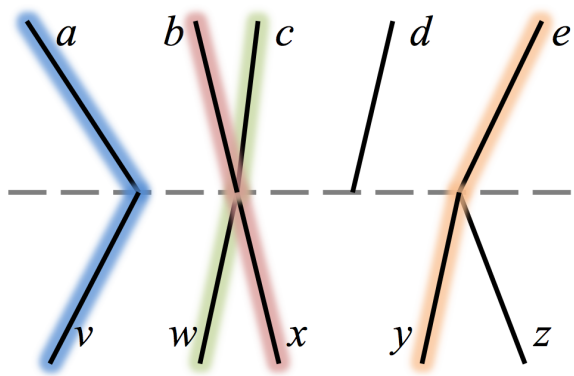


Figure 11: Example of joined edges with incompatible pattern interfaces. Here, consistent with our visual intuition, the matching algorithm returns (a, v), (b, x), (c, w) and (e, y). Patterns d and z are unmatched and their strips terminate at this edge.

Now that patterns are matched up via this criterion, tracing them to construct strips is straightforward. We pick a pattern on some tile arbitrarily, and start tracing across polygonal boundaries first from the entering edge, and then the exiting edge. We can then concatenate the internal list of segments stored by each pattern onto the strip we are building (reversing it if necessary to account for orientation) to build up a full strip. At every step, we also compare against the original pattern that we picked to detect if we enter a loop.

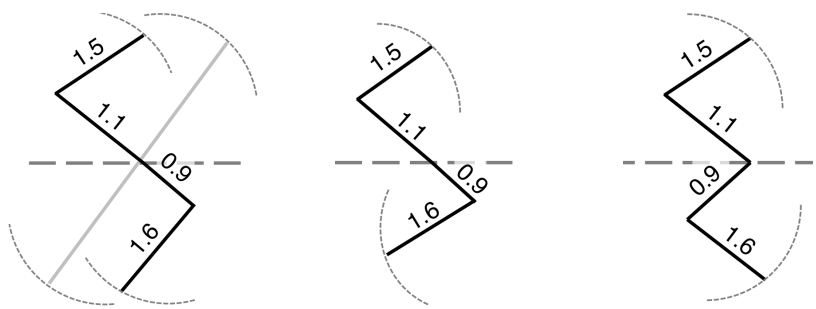
There are some subtleties about the concatenation process. We identify three possibilities and concatenate strips differently based on each case:

(a) if another pattern crosses that edge at the same point, the edge boundary is an intersection and we concatenate the relevant segments together.

(b) if no other pattern originates from that same point and the two incident angles are compatible (within ϵ of each other), the strip visually appears to be a single segment spanning the boundary and we add the lengths together.

(c) if no other pattern originates from that same point but the two incident angles don't match, the strip has a joint at the edge boundary and a simple concatenation of the two lists of segments suffices.

Figure 12 below illustrates each of these cases.



(a) Output:
`[..., [..., 1.6], [0.9, 1.1], [1.5, ...],...]`

(b) Output:
`[..., [..., 1.6], [2.0], [1.5, ...],...]`

(c) Output:
`[..., [..., 1.6], [0.9], [1.1], [1.5, ...],...]`

Figure 12: Concatenation of patterns is performed differently depending on the edge boundary.

Now by repeating this process until every pattern in a polygonal tile has been assigned to a strip, we may convert a tile-based representation of the geometric configuration to a strip-based one.

Note that in these geometric configurations, strips must either terminate at the boundary of a tile or form a closed loop; the second case must be treated specially. When tracing a strip across polygonal boundaries, we record the first segment seen and compare it to each new one that we are appending to the strip. If we

encounter it again, the strip is known to be a closed loop. We append the first segment to the strip again and then break out of the search.

By repeating one segment on both sides of the strip, we get some physical overlap which can be used to glue the closed loop shut (see Figure 13).

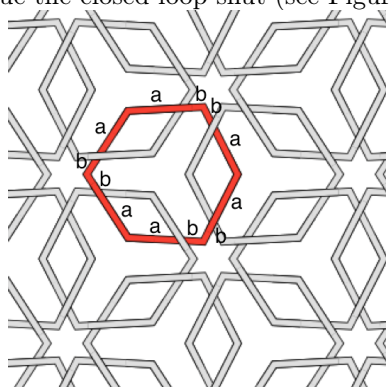


Figure 13: When tracing strips that formed closed loops, one segment is repeated for gluing. Here, the output strip has seven segments and is represented by `[[a,b],[b,a],[a,b],[b,a],[a,b],[b,a],[a,b]]`.

5. STRIP RENDERING IN SVG

Wallpaper also includes a utility to translate between the data representation of a strip (a list of list of numbers) to a Scalable Vector Graphics (SVG) file representing that strip.

Given the dimensions of a strip, we can render it as a rectangle with vertical slits spanning alternately the top or bottom half.[1] Slotting strips along these slits provides a secure interwoven structure. Joints, or creases, are rendered as full vertical lines in a separate color, so that the user can choose to fold them manually or score them by setting the laser cutter to a different power setting. Figure 14 shows an example of one such generated strip.

Figure 14: Strip rendered by our system based on the traced dimensions, ready for die-cutting / laser-cutting and physical assembly.

[1] Under relatively weak conditions – namely, that all strips terminate on the exterior of the tiling – we can assign crossings at intersections such that strips always alternate over and under.

This SVG file can either be printed and cut out by hand, or cut and scored automatically using a die cutter or a laser cutter.

6. CONCLUSION AND FUTURE DIRECTIONS

Conventional Islamic star pattern design techniques are very successful in generating symmetrical and aesthetically pleasing geometric configurations. We have demonstrated Wallpaper, a system that enables us to leverage these techniques to dynamically create these geometric configurations, but also physically recreate these geometric configurations with paper strips, by taking a tile-based data representation of such designs and converting it into a strip-based representation.

Brimstone (see Figure 15), a design by the first author, demonstrates the utility of this system — design and strip generation was heavily automated and completed within an hour. Compare this with the designs in Figure 1, which had all been worked out by hand and traced over the better part of a day.

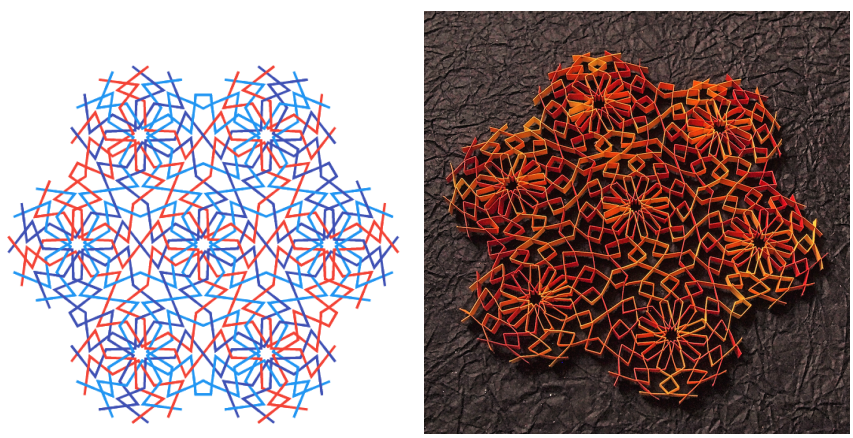


Figure 15: Brimstone (2015). The geometric design (left) was traced to generate paper strips, which was assembled to produce sliceform artwork (right).

We have also seen that Wallpaper also serves a pedagogical purpose — some components of the system, such as the drag-and-drop shape generator and the custom pattern editor, are great hands-on demonstrations of tilings and symmetry.

We hope to continue developing this system to support more features, such as design of non-planar and possibly 3D configurations. Furthermore, we aim to

promote this artform by generating and assembling more pieces that stretch the boundaries of the genre.

7. REFERENCES

Abas S.J., Salman A.S. (1995) Symmetries of Islamic Geometric Patterns. World Scientific.

Bettens, C. (2009) "Zillij: slice form techniques applied to patterns from arabic art.", 2009. `https://www.flickr.com/photos/melisande-origami/sets/72157613125224450/`, accessed 04-22-2015.

Bourgoin, J. (1973). Arabic Geometrical Pattern and Design. Dover Publications.

Hankin, E.H. (1925) Memoirs of the Archaeological Society of India, volume 15. Government of India.

Kaplan, C.S. (2000) Computer generated Islamic star patterns. In: *Proceedings of Bridges 2000*, 105–112.

Lee, A.J. (1995) Islamic star patterns. In: *Muqarnas*, 4:182–197.

Lu, Y., Demaine, E. (2015) Wallpaper. `http://yqlu.me/wallpaper`, accessed 04-22-2015.

Palmer, C. (2009) "Zillij", cardstock, 12". `https://www.youtube.com/watch?v=2TNUxWVgZTs`, accessed 04-22-2015.