Polygonia Design Suite: Inspiration, Design, and Application

David Kaufman

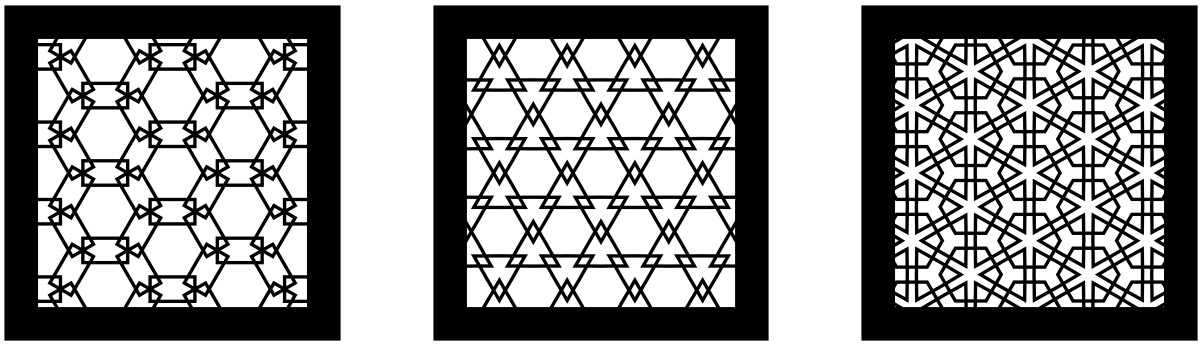
Kensington, Maryland, USA; david@Polygonia.design

Abstract

In this paper I review the development and applications of Polygonia Design Suite. Polygonia is an online tool for creating symmetrical drawings. Drawings can be downloaded as vector files and can be used in various fabrication processes, such as laser cutting, paper cutting, and 3D printing.

Introduction

Repeating geometric patterns are simultaneously engaging and soothing. Engaging because the mind enjoys searching for new combinations of shapes in the repetitions, and soothing because the symmetry provides its own sense of completeness and closure. The Sackler Gallery of the Smithsonian Institution presented an exhibit on the artisans of the Turquoise Mountain region of Afghanistan in the spring of 2017 [7][10]. Included in the exhibit was the work of Ustad Nasser Mansouri [3]. Mr Mansouri creates symmetrical decorative architectural wooden screens. Ten different designs were presented as 30cm square panels, each 3cm thick. A counter in the exhibit held three panels for visitors to handle, allowing for an increased interaction with the designs. Figure 1 shows re-creations of three of his works.



(a) (b) (c)

Figure 1: *Re-creations of the work by Mr. Mansouri in the Turquoise Mountain exhibit at Smithsonian.*

Inspiration and Design Goals

I was captivated by the symmetry of these wooden screens, each beautiful and calming in their own way. I had access to a laser cutter at my local maker space, NovaLabs, in Fairfax, VA [6]. I wanted to recreate these designs, as well as explore variations and create new designs. I wanted a tool to allow complete freedom to explore this domain of symmetrical designs or drawings, one that specifically would minimize the effort in creating files for laser cutting. The vision was for a web app that would provide an effortless means to create symmetrical drawings appropriate for laser cutting. The app would do as much work as possible to reduce the work required of the end user and minimize design errors. The app would apply reflection, rotation, and translation rules to the lines drawn by the user to create the symmetry. The app would support three “template tilings” (triangle, square, and hexagon), but not the complex template tilings that generate Islamic geometries. The app would would export vector files. The output file would consist of non-intersecting polygons that outline the negative space of the drawing.

App Usage

The app is written in Javascript and uses HTML/CSS. Figure 2 shows the two primary panels within the app. The rendered drawing is on the left. The editor and settings are on the right. This drawing was created with two lines, shown in blue in the editor.

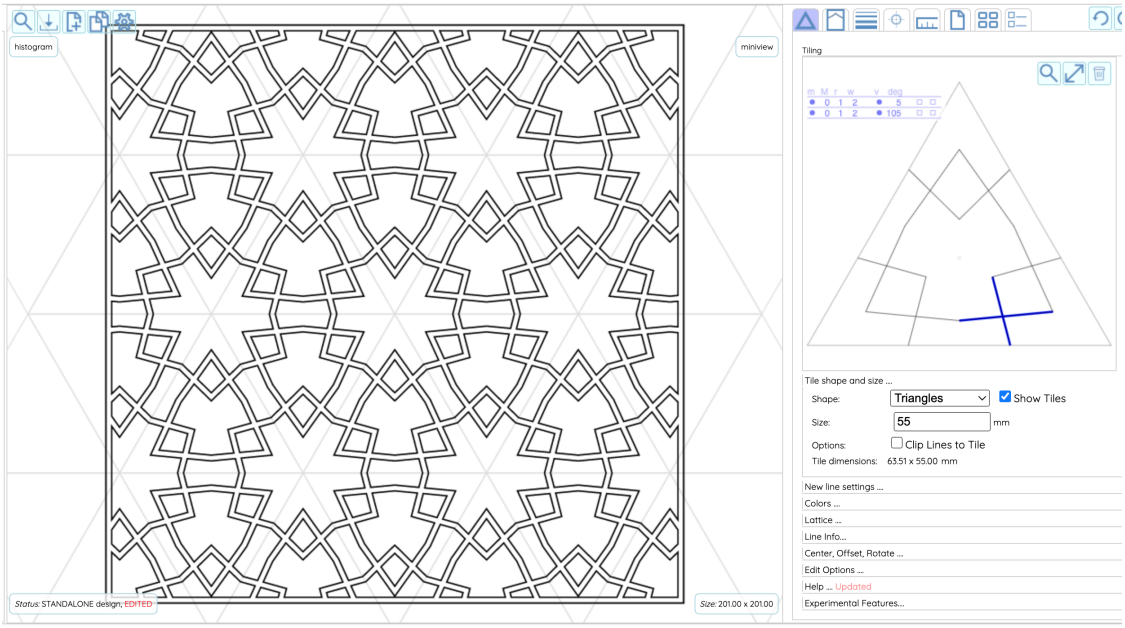
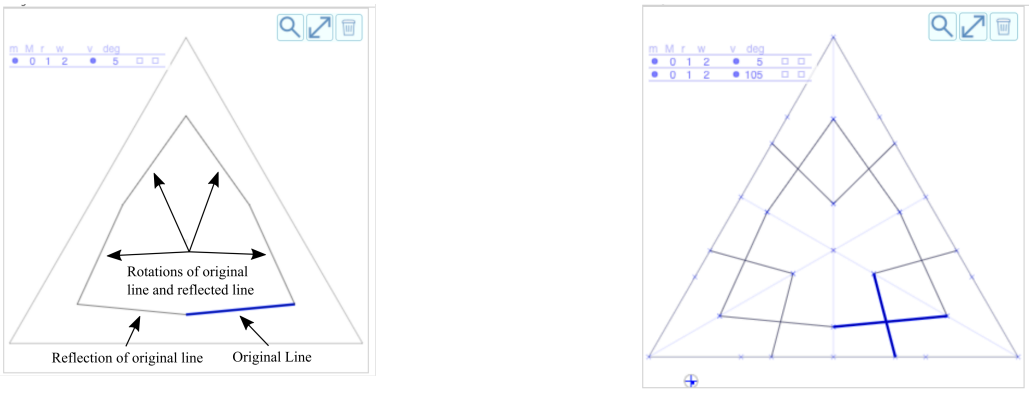


Figure 2: *App showing rendered drawing (left) and editor (right).*

The underlying layout for a drawing is based on a template tiling. The template tiling is shown in light gray behind the rendered drawing in Figure 2. The "frame" of this drawing is a square. The app calculates a template tiling to fill the frame. The size of the template tiling can be changed with the "Size" field.
Users draw lines in the editor. This is shown in blue in the editor panel in Figure 2. Figure 3(a) shows a single line and how the editor will make copies of this line. The editor will make a reflected copy of the line drawn by the user and also two rotated copies of both the original and reflected line (for a triangular template tiling).



(a) (b)

Figure 3: *(a) Editor panel showing the line drawn by the user and its reflected and rotated copies. (b) Editor panel showing two lines drawn by the user, their reflected and rotated copies, and the symmetry lines.*

The app distinguishes between “lines” and “edges.” Lines are shown in the editor. Edges are shown in the rendered drawing. Edges are calculated last, by the polygon offset algorithm, described below.

The app uses the template tiling to fill the frame with copies of the line drawn by the user and its reflected and rotated copies. The lines are clipped to the frame.

Each user line in the editor has a width value (default is 2mm). Once all of the lines have been copied using the template tiling and clipped to the frame, the widths for each line are applied and used to calculate the edges of the drawing. Figure 4 shows the rendered drawing and editor with the single line from Figure 3(a).

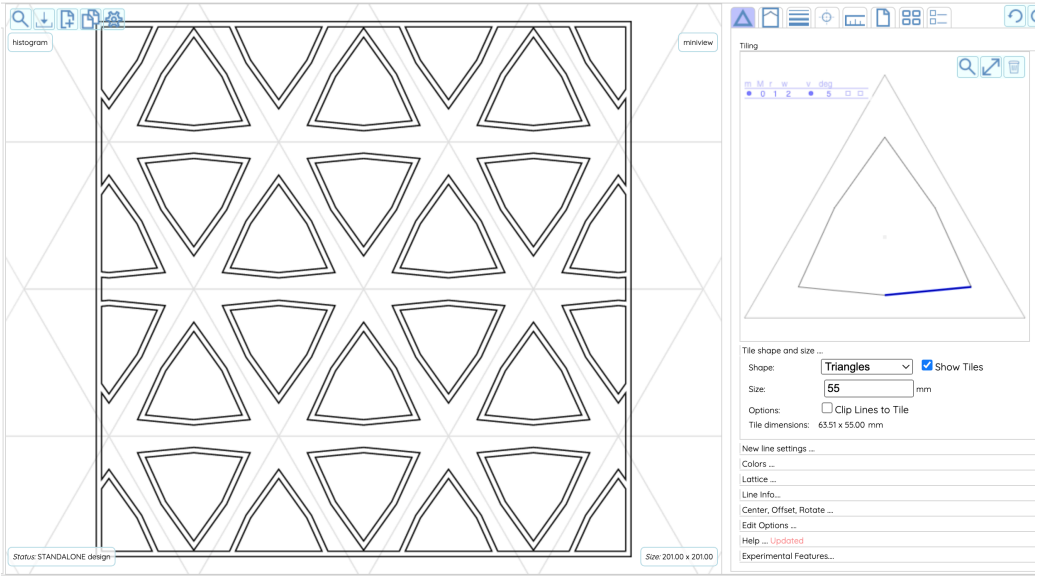
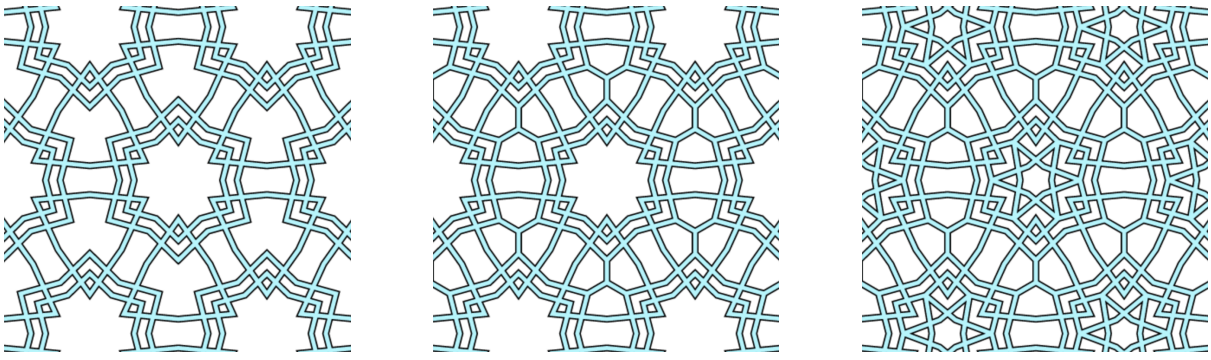


Figure 4: *App showing drawing where one line has been drawn in the editor.*

Figure 3(b) shows the editor panel with two lines drawn by the user. As before, the editor adds the reflected and rotated copies. Also shown are the symmetry lines for the triangle. This triangle corresponds to the triangles in the triangular tiling template. The lines in this figure are the same as in Figure 2 and are used to create the drawing in Figure 2.

Figure 5 shows three variations of the drawing in Figure 2. These were created by adding one, two, and three lines to the editor. Changes made in the editor cause the drawing to be immediately re-rendered, allowing a quick exploration of design variations.



(a) (b) (c)

Figure 5: *Original design from Figure 2 shown with one (a), two (b), and three (c) additional lines drawn by the user. A fill color was added and the template tiling was hidden to better view the drawing.*

Design

The significant challenges in the design of the app were the logic to assemble the intermediate polygons and the polygon offset algorithm.

Polygon Assembly

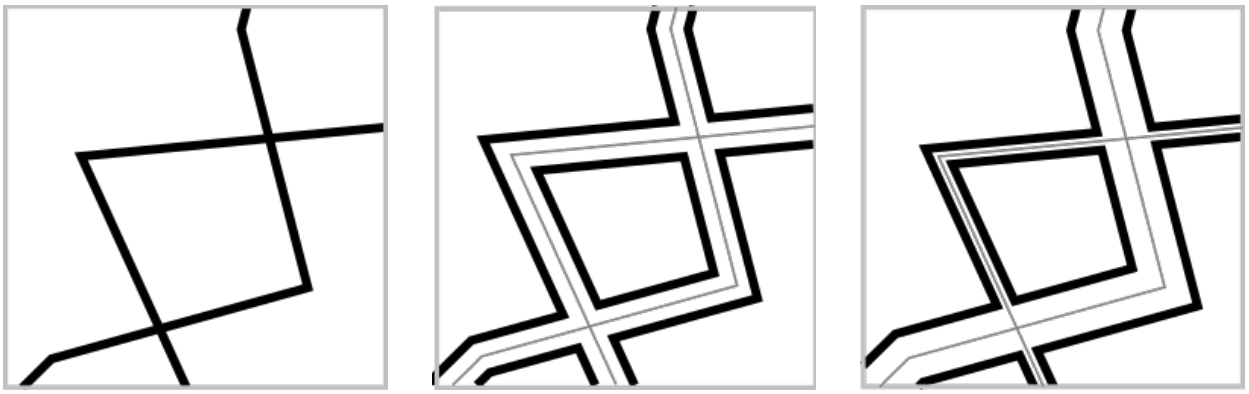
As mentioned earlier, the app creates a list of lines that are the reflected, rotated, and translated copies of the lines drawn in the editor. These lines may intersect each other. The app needs to convert these lines into a list of polygons. The polygons will have common edges, but must not intersect.

I created a polygon construction algorithm that builds a *directed segment graph* from a list of lines and traverses the graph to construct polygons. In this graph, graph nodes are line endpoints and graph edges are line segments. The algorithm handles intersections by adding the intersection point to the graph and subdividing the intersecting lines (graph edges). Once all of the lines are added, the intermediate polygons are assembled by traversing the graph along adjacent edges.

Figure 6(a) shows detail from the output of this stage for the drawing in Figure 2. Each line segment in Figure 6(a) is actually two coincident segments, one for each of the polygons sharing a common polygon edge. Figure 6(b), showing the polygons after the offset is applied, makes this clearer.

Polygon Offset Algorithm

The app creates the final polygons using the intermediate polygons from the previous step. The final polygons represent the negative space of the drawing. These are the white areas in Figures 5(a), 5(b), and 5(c). The final polygons are computed using a polygon offset algorithm. Figure 6(b) shows the result after offsetting the polygons in Figure 6(a) by the same distance for each edge. The width specified in the editor becomes the distance between the edges of neighboring polygons. Each edge is offset by half of this width.



(a) (b) (c)

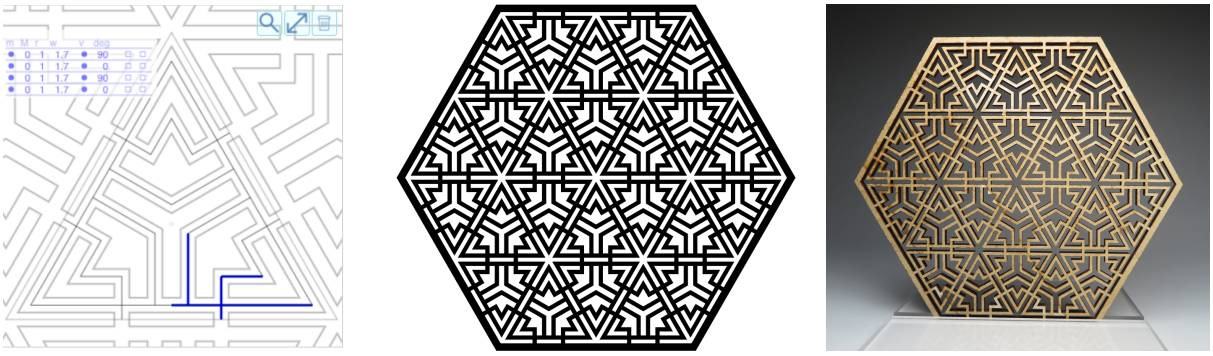
Figure 6: *(a) Detail of intermediate polygons with coincident edges. (b) Final polygons created with constant offset from polygons in (a). (c) Final polygons created with different offsets.*

I created a polygon offset algorithm based on a polygonal skeleton algorithm [1], but with additional capabilities. In the app each polygon edge has its own offset distance. Figure 7(c) shows an example where one line in the editor has a width of 1mm and the other line has a width of 3mm. The app's polygon offset algorithm uses the value associated with each edge of the input polygon to calculate the output polygon.

Application

Initial Laser Cut Output

Figure 7(a) shows the four lines in the editor that create the drawing in Figure 7(b). Figure 7(c) shows the first laser cut output from the app, using the Figure 7(b) drawing. (16cm × 14cm).



(a) (b) (c)

Figure 7: *(a) Editor showing only four lines necessary to create drawing in (b) and used for (c) first laser cut item fully designed in Polygonia.*

Paper, Fabric, Polyhedra, 3D printing

Figure 8(a) shows a layered piece made of four layers of paper (22cm × 19cm). This particular piece was laser cut from card stock. Figure 8(b) shows three fabrics designed in the app and produced by a print-on-demand fabric company (53cm × 50cm). Figure 8(c) shows a triacontahedron with laser-cut faces, where the face was designed in the app with the "diamond" frame (24cm × 24cm × 24cm).

(a) (b) (c)

Figure 8: *(a) Paper cut with a laser cutter. (b) Fabric printed on-demand, (c) Laser cut plywood, Rhombic triacontahedron.*

Additional Capabilities

The app has a large number of features for controlling various aspects of the drawing. Reflections and rotations of the lines can be disabled or enabled per-line. The template tiling size can be changed at any time. The template tiling can be rotated and shifted within the frame. Simple weaves, lattices, and knots can be created by setting the z-value of the lines. A mandala option is available, where the user can set the number of rotations within the polygon of the template tiling. Colors can be set on a per-line basis. Corners in the drawing can be softened with arcs or quadratic bezier curves. Shadows can be added. A background gradient fill can be added.

Related Work

There are a few key differences between the capabilities of the app, Polygonia, as compared to Islamic Star Pattern Generation [4]. Polygonia has simpler template tilings, but provides for complexity by allowing the user to draw multiple lines of differing widths. Islamic Star Pattern Generation provides complexity with a variety of template tilings, but limits the number of lines in the pattern generation.

Polygonia's vector output distinguishes it from many other tessellation and pattern generation apps which generate pixel output, such as the Tilemaker app from the Qatar Foundation International [8], Tessellation Creator [5], and Shodor's Tessellate [9]. Pixel output from these apps does not scale, as vector output can, and can only be used for printing.

Polygonia may not support all 17 wallpaper groups as described in Tessellations [2]. Some glide reflections can be supported by disabling the rotation and reflection settings of the lines drawn in the editor, but others may not be supported.

Summary and Conclusions

Polygonia Design Suite allows rapid and effortless exploration and generation of symmetrical drawings. The vector output allows the drawings to be used with laser cutters, paper cutters, 3D printers, and CNC machines. Symmetries on the screen can be held in the hand. The raster output allows the drawings to be used with fabric printers and manufacture-on-demand services. The online storage allows users to quickly iterate on designs and variations.

Future Directions

I would like to add the ability to have layers in a drawing. Figure 8(a) is composed of four layers of paper. These are four separate drawings. With a layers feature, these would all be part of a single drawing. I would like to add support for bezier curves in the editor. I implemented a prototype of this and saw that it could be very powerful, but it also revealed performance issues in the current architecture and user interface. I would like to add a lattice generation algorithm. The app supports lattices now by post-processing a drawing and adding additional edges. I would like this to be part of the core functionality of the app.

References

[1] F. Cacciola, The Computational Geometry Algorithms Library, *CGAL 5.5.2 - 2D Straight Skeleton and Polygon Offsetting*. https://doc.cgal.org/latest/Straight_skeleton_2/index.html

[2] R. Fathauer. *Tessellation: Mathematics, Art, and Recreation.* A K Peters/CRC Press, 2021

[3] The General Society of Mechanics & Tradesmen of the City of New York. *Turquoise Mountain: Artists Transforming Afghanistan.* 2017. https://generalsociety.org/?p=1613

[4] C. S. Kaplan. "Computer Generated Islamic Star Patterns." *Bridges Conference Proceedings*, Winfield, Kansas, USA, July 28–31, 2000, pp. 6–7. https://archive.bridgesmathart.org/2000/bridges2000-105.pdf

[5] National Council of Teachers Mathematics. *Tessellation Creator*. https://www.nctm.org/Classroom-Resources/Illuminations/Interactives/Tessellation-Creator/

[6] Nova Labs, Inc. *The Nova Labs Makerspace*. 2022. https://www.nova-labs.org/

[7] A. Pigulski. *Turquoise Mountain at the Sackler Gallery*. 2017. https://homefronting.wordpress.com/2017/06/14/turquoise-mountain-at-the-sackler-gallery/

[8] Qatar Foundation International. *TileMaker*. 2023. https://tilemaker.qfi.org/gallery

[9] Shodor Education Foundation, Inc. *Tessellate!*. 2023. http://www.shodor.org/interactivate/activities/Tessellate/

[10] Smithsonian Institution. *Turquoise Mountain: Artists Transforming Afghanistan*. 2017. https://asia.si.edu/exhibition/turquoise-mountain-artists-transforming-afghanistan/