Automatic Origami Crease Pattern Generation from k-uniform Tilings

David Goncharov[1], Marcus Michelen[2], and Uyen Nguyen[3]

[1] University of Illinois, Chicago. davidvgoncharov@gmail.com
[2]University of Illinois, Chicago. michelen.math@gmail.com
[3]New York, NY, USA. win@winwin.fashion

Abstract

We automate the process of constructing an origami crease pattern based on certain planar tilings. In particular, our computer program can take in any k-uniform tiling and output a crease pattern on top of that. This amounts to automating previous work of Nguyen and Fritzson, and allows for the generation of crease patterns that are far too complicated to be generated by hand.

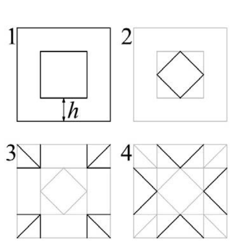
1 Introduction

We investigate the process of generating origami crease patterns based on planar tilings. A previous work of Nguyen and Fritzson [3] describes a process for taking certain planar tilings of convex shapes and making a crease pattern for an origami corrugation based on that tiling. Performing that algorithm manually—even with the use of a computer—is often challenging due to the level of precision necessary as well as the way that errors propagate significantly across large crease patterns. Additionally, many of the tilings the process described by [3] applies to are too large or complicated to perform their algorithm on one segment or one polygon at a time. Our contribution is to automate the process of [3], allowing the efficient construction of crease patterns based on large complex tilings at a high degree of precision. Our Python program performing the automation is included in our supplemental files for use by others.

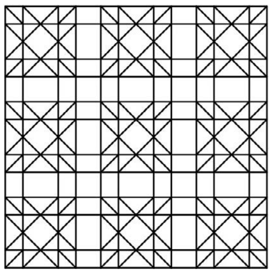
The class of tilings considered are k-uniform tilings, i.e. tilings of the plane by convex polygons connected edge-to-edge with k equivalence classes of vertices. Our program is able to take in an arbitrary k-uniform tiling and build a crease pattern based on it using the algorithm of [3]. There are an infinite number of k-uniform tilings, and enumerating them even for moderately sized k appears to be fairly challenging. To the best of our knowledge, the most extensive library of such tilings is that of Čtrnáct and Kopczyński [1], which lists all k-uniform tilings for $k \leq 12$.

The crease pattern is output in a .dxf format, which can then be scored with a vinyl or laser cutter, or input into Ghassaei's Origami Simulator [2]. In Section 2 we describe our program in detail, using the square tiling as a recurring example. We provide examples of output in the form of crease patterns, simulated folding, and in a paper-folding example in Section 3. Finally, we describe some possibilities for future work on this project in Section 4.

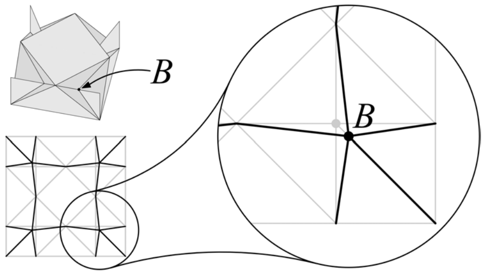
Summary of the Nguyen-Fritzson process The procedure developed by Nguyen and Fritzson [3] is inspired by Ron Resch's *Linear Flower*. Beginning with a k-uniform tiling, an operation is performed on each face, summarized in Figure 1(a). The faces are then connected by a set of lines known as rivers, summarized in Figure 1(b). In order to allow for rigid foldability, certain vertices within each face must be shifted to a precisely calculated location in what Nguyen-Fritzson call a *B-shift*, summarized in Figure 1(c); the value giving the shift required is found by solving a non-linear system of four equations in four unknowns for each type of face that occurs.



(a)



(b)

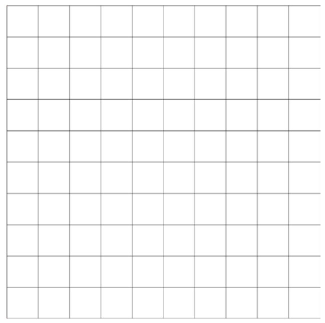


(c)

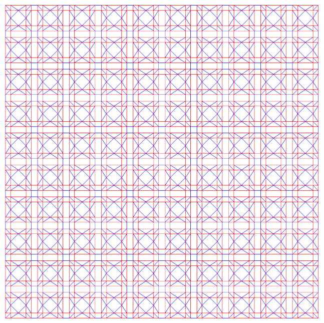
Figure 1: *An overview of the Nguyen-Fritzson process, with diagrams taken from [3]. (a) The facewise operation (b) building rivers between faces (c) the B-shift*

2 Overview of Program

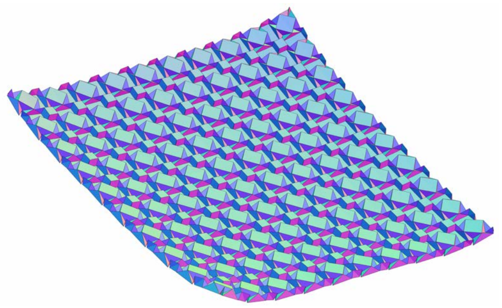
Our program follows the same broad strokes as the Nguyen-Fritzson approach. The main challenge was to write the program in such a way that it has the flexibility to deal with an arbitrary k-uniform tiling.



(a)



(b)



(c)

Figure 2: *An overview of the steps of the program, with (c) showing the output from Origami Simulator[2].*

Generating the tiling To describe a tiling, we consider an input in the form of what we call a *gluing pattern*. A gluing pattern consists of two parts: first, for each vertex type it describes the angles from the horizontal at which each edge occurs. Second it describes which edges are identified—or glued—together, where a pair in the list of the form $(a, i) : (b, j)$ means that the ith edge of vertex a is glued to the jth edge of vertex b. For instance, the gluing pattern for the regular square tiling is given by the relatively simple pair of lists

$$\left[0, \frac{\pi}{2}, \pi, \frac{3\pi}{2}\right] \qquad \{(0,0) : (0,2), (0,1) : (0,3), (0,2) : (0,0), (0,3) : (0,1)\}$$

We treat reflections of vertices to be a different vertex type, and so a gluing pattern for a k-uniform tiling can have more than k vertex types listed. Once a gluing pattern has been given, the tiling is generated recursively. A single vertex is placed down, and for each edge attached to it, we add to a Python dictionary—which we treat like a queue—the location and type of each vertex adjacent to it that has yet to be placed. We then take an element from this dictionary of unplaced vertices, place it along with its edges, and add any new unplaced vertices that would be adjacent to it. We then continue this process, not adding any vertices that would be out of bounds, until there no more unplaced vertices. Figure 2(a) shows the outcome of this for the square lattice.

Identifying the faces Our next step is to identify the faces of the tiling. Up until this step, the only structure stored is the collection of vertices along with the edges. In order to identify the faces, this amounts to finding minimal cycles in the underlying graph that gives the tiling. For a general graph, this may be done by a depth first search. Without any assumptions on the underlying graph, this may be somewhat computationally intensive in practice for graphs at the size and complexity that come up in this application. In our case, since every face in a k-uniform tiling has at most 12 edges, all minimal cycles are of size at most 12. As such, we use a modified depth first search that simply gives up when searching for a minimal cycle of length strictly greater than 12.

Work face-wise to create the tilings within each face With the list of faces given, we now perform the Nguyen-Fritzson process on each face. The majority of this amounts to planar geometry and trigonometry, with the exception of calculating the B-shift. This is accomplished by using the `scipy.optimize` function `fsolve` to numerically solve the system of four equations in four unknowns provided by Nguyen-Fritzson. The B-shift equations are solved only once for each type of face that occurs in a given tiling, and thus the typically computationally-intensive task of approximately solving a non-linear system of equations numerically is only done a handful of times in each given iteration of the program.

Create rivers between the faces The faces are then connected by rivers and the crease pattern is output as a .dxf file, for example in Figure 2(b). Mountain folds are output as blue and valley folds as red so that one may input them to Ghassaei's Origami Simulator [2]. Figure 2(c) shows the outcome of entering the crease pattern into the Origami Simulator.

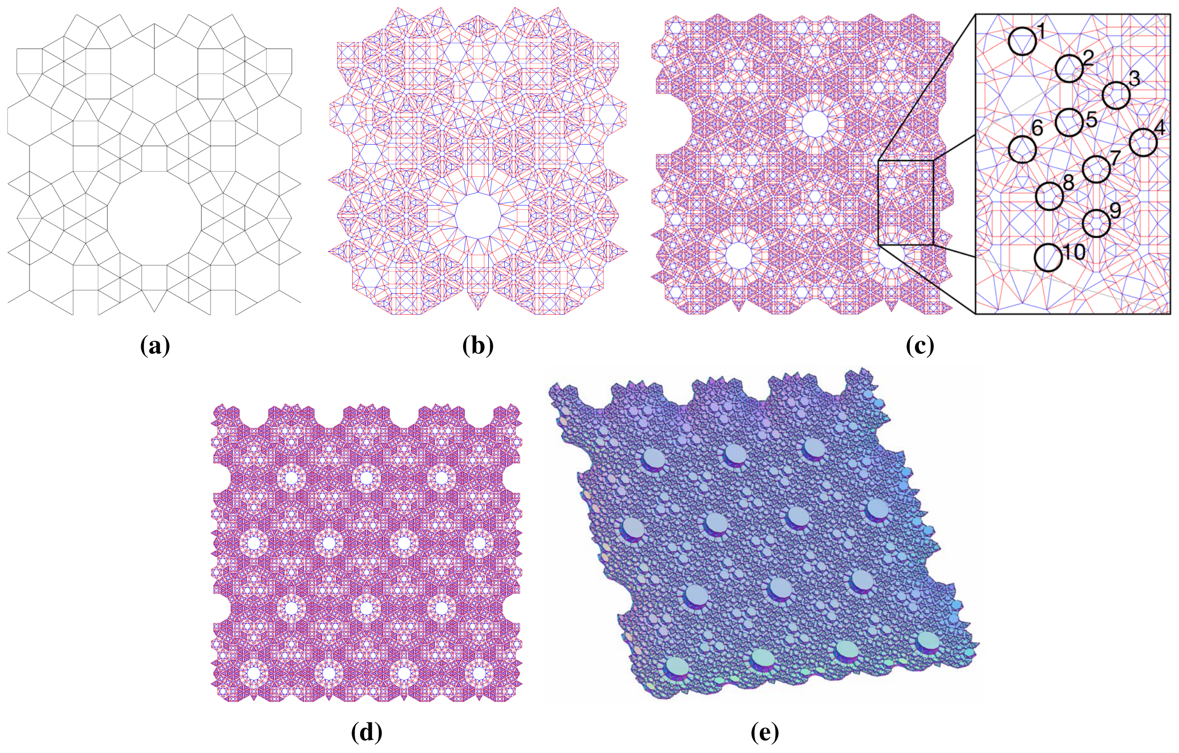


Figure 3: *Sample outputs for the tiling described in equation* (1). *(a) A portion of the tiling; (b) the outputted crease pattern; (c) an identification of the 10 vertices present; (d) a larger crease pattern of the same tiling; (e) the Origami Simulator [2] output of this crease pattern*

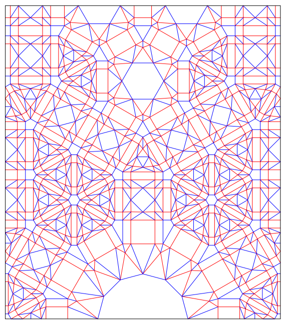
3 Sample Outputs

As examples, we provide the output when applied to a 10-uniform tiling with vertex configuration by

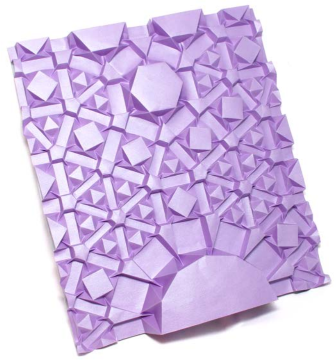
$$\left[3^6, 3^3.4^2, (3^2.4.3.4)3, 3^2.4.12, 3^2.6^2, 3.4^2.6, 3.4.6.4, 6^3\right] \tag{1}$$

Four outputs are provided in Figure 3. In Figure 3(a), we output a piece of the tiling considered, which was generated by our program; Figure 3(b) applies the Nguyen-Fritzson process to this piece of the tiling; Figure 3(d) applies the process to a much larger portion of the tiling; and Figure 3(e) is the result of inputting the crease pattern from Figure 3(d) into Origami Simulator. We also include in Figure 3(c) a detail where we identify the 10 different types of vertices present in the tiling.

In order to test our program on a physical example, we folded a smaller portion of the same tiling, shown in Figure 4; Figure 4(a) shows the crease pattern used and Figure 4(b) shows the result of machine-scoring and hand-folding that crease pattern.



(a)



(b)

Figure 4: *Outputs for the tiling provided by equation* (1)*: (a) A portion of the crease pattern generated; (b) A paper-folding of that crease pattern.*

4 Future Work

One aspect for extending the work is to have different options for describing different tilings. In our gluing patterns, each edge is listed twice. One option would be to adapt the program to use the same format used in the library of k-uniform tilings of Čtrnáct and Kopczyński [1]. Another option is the format is described in the work of Sánchez et al. [4]. Finally, it would be interesting to extend our program to handle convex tilings of irregular polygons provided there are only finitely many equivalence classes of vertices.

Acknowledgments

This work began as a portion of a project at the University of Illinois Chicago's (UIC) Mathematics Undergraduate Computing Laboratory of which the first-named author was a member and the second-named author was the mentor. We thank UIC for facilitating this collaboration as well as group members Matthew Grode and Natalia Sliwinska. DG and MM were both supported in part by NSF grant DMS-2137623. We are grateful to Be Fritzson for comments on a previous draft.

References

[1] M. Čtrnáct and E. Kopczyński. "Tessellation Catalog." 2021. https://github.com/zenorogue/tes-catalog/

[2] A. Ghassaei. "Origami Simulator." http://apps.amandaghassaei.com/OrigamiSimulator/.

[3] U. Nguyen and B. Fritzson. "Origami Explorations of Convex Uniform Tilings Through the Lens of Ron Resch's Linear Flower." *Bridges Conference Proceedings.* pp. 515-518. 2018.

[4] J. Sánchez, T. Weyrich, A. Sá, and L. de Figueiredo. "An Integer Representation for Periodic Tilings of the Plane by Regular Polygons." *Computers & Graphics.* 95. pp. 69 - 80. 2021