A tool for designing with visual algorithms

Andrew I-kang Li*1 and Rudi Stouffs*2

*1 Associate Professor, Department of Design and Architecture, Kyoto Institute of Technology, Ph.D.
*2 Associate Professor, Department of Architecture, National University of Singapore, Ph.D.

Keywords: visual algorithms; generative design; algorithmic design; design tools; shape grammar; implementations.

1. Introduction.

Algorithms are an important aspect of digital design and fabrication; they underlie many technologies like mass customization. Algorithms are often expressed as computer programs. These symbolic representations are suitable for mathematicians, but less so for people who think and work visually, like designers. These visual workers would prefer to draw algorithms.

As an example of a visual algorithm, take the pattern known as the Conway tessellation (Figure 1).

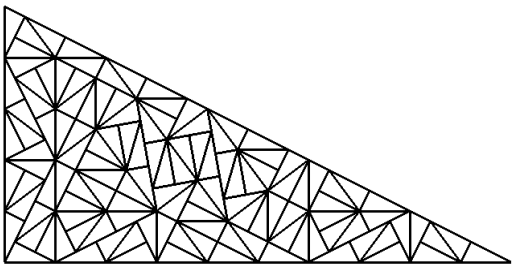

Figure 1. The Conway tessellation

It is a complex figure, but we can construct it easily with this simple algorithm. Expressed in words, the algorithm looks like this:

1. Start with a triangle with sides of lengths 1, 2, and √5.
2. Subdivide the triangle into 5 similar triangles. Repeat optionally.

When the same algorithm is expressed visually, it is easier to understand. The visual version has the same two parts: an initial figure (the triangle) and a transformation consisting of the triangle before subdivision and the same triangle after subdivision (Figure 2). It is easy to see that, with this algorithm, we can transform the triangle indefinitely many times and create many different tessellations. One such sequence might begin with these three figures (Figure 3).

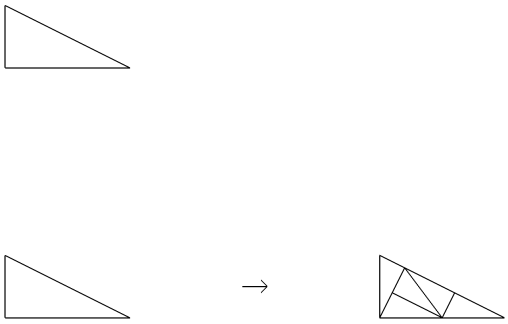

Figure 2. A visual algorithm for creating Conway tessellations. It consists of an initial figure (the triangle above) and a transformation (the before-and-after figures below separated by an arrow).

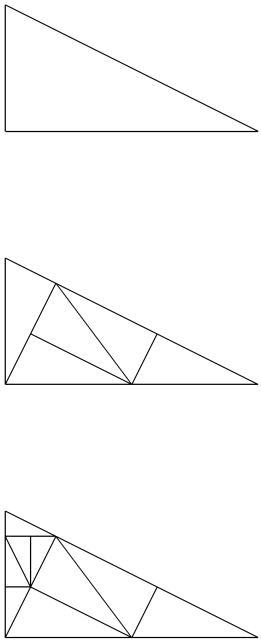

Figure 3. The first three figures in one possible development of a Conway tessellation.

日本建築学会情報システム技術委員会
第42回情報・システム・利用・技術シンポジウム論文集，150-153，2019年12月，東京
Proceedings of the 42nd Symposium on Computer Technology of Information，
Systems and Applications，AIJ，150-153，Dec.，2019，Tokyo

And it tmight end with these three figures (Figure 4):

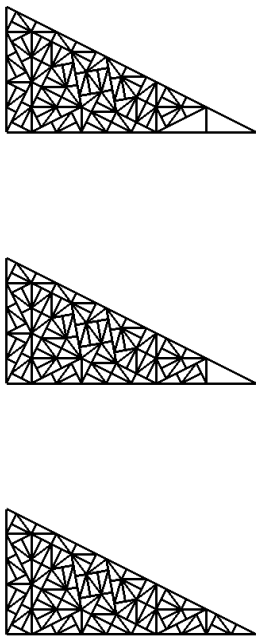

Figure 4. The last three figures in one possible development of a Conway tessellation.

We are developing a tool[1), 2)] that enables designers to use visual algorithms. It is known technically as a shape grammar interpreter. With it, users can draw algorithms, execute them, and thereby create 2d and 3d designs.

2. The interpreter.

The interpreter runs in the widely used modeling application Rhinoceros3d (v5 or later). It consists of the visual interface (the front end) and the engine (the back end). The interface consists of Python scripts which the user invokes like commands or menu items. There are just 3 main scripts: 'initialize', 'create rule', and 'apply rule'.

The user runs the 'initialize' script when beginning work in a Rhino document. If the document is new (empty), the system prepares the document and itself to work with visual algorithms. If the document is old, i.e,. if it contains an algorithm from a previous work session, the system reads the algorithms from the document.

To create a transformation (known technically as a rule), the user draws the left and right figures, using Rhino's native capabilities. She runs the 'create rule' script; the system formats and reads the rule.

The user runs the 'apply rule' script to apply the transformation and generate new figures. The system calculates all the possible new figures and draws them in the Rhino document.

In this way, designers can use algorithms without numbers, coordinates, or other symbols; they work in an entirely visual manner. In addition, they can develop these figures further using the many tools available in the Rhino ecosystem. They can, for example, easily convert their virtual designs to physical objects; these provide additional artifacts for thinking about their algorithms.

The user draws an algorithm for the Conway tessellation that looks just as in Figure 2, and it is ready for use. She can obtain the sequences in Figures 3 and 4 by simply applying the transformation to each new figure in turn. She need draw nothing else; the interpreter calculates and draws the figures.

3. Discussion.

We have been testing the tool in a variety of settings, including a one-semester class for postgraduate design students and workshops ranging in length from two to eight days.

This has led us to some preliminary observations. First, users easily master the visual interface and are soon able to concentrate on developing and testing visual algorithms. It seems reasonable to assume that the visual interface and automatic generation of figures contributes to this.

Second, users benefit from fabricating tangible objects as feedback in developing their algorithms (Figures 5-10). This is easy to do in the Rhino environment, where many tools are available to support digital fabrication.

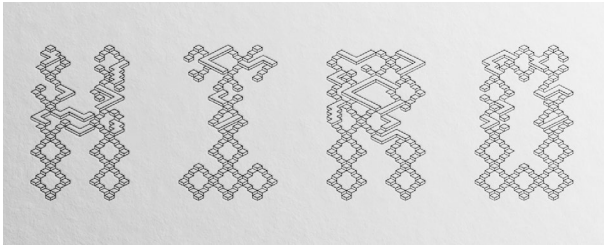

Figure 5. Letters in an algorithmically created font.

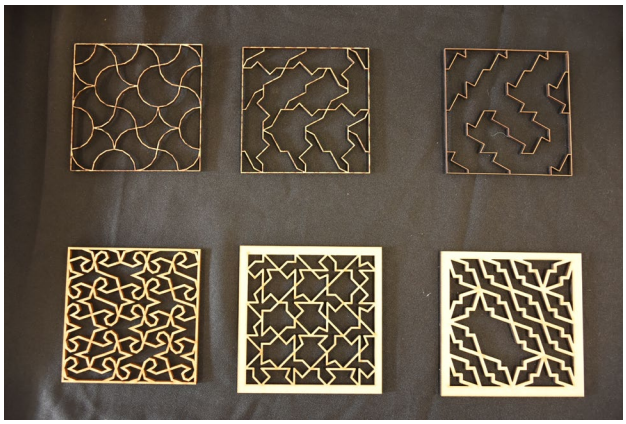

Figure 6. Studies in tessellation using variant algorithms and laser-cut in wood.

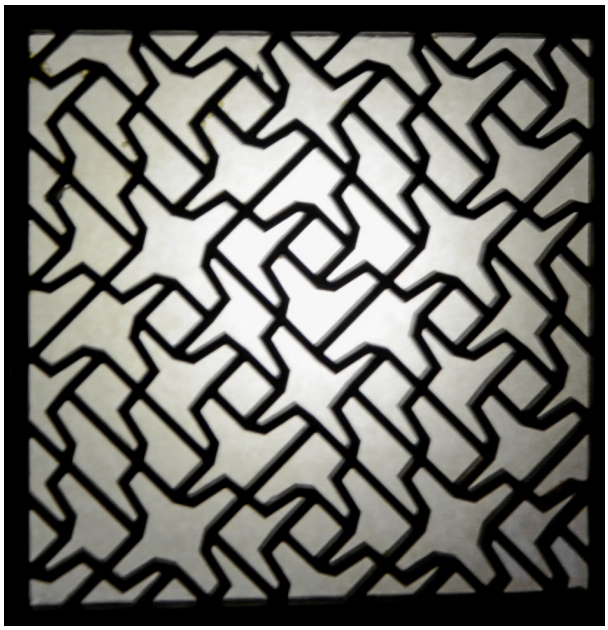

Figure 7. A laser-cut wood lamp made from a tessellation algorithm.

Figure 8. 3d-printed chopstick holders.

Figure 9. A 3d-printed accessory case.

The participants in our classes and workshops have all been new to visual algorithms. As a result, they made few technical demands on the interpreter. In fact, the interpreter has more sophisticated capabilities, but we need more experienced users to evaluate them. For example, in addition to the visual interface, there is a second interface, implemented in the Rhino plug-in Grasshopper, that is less visual and intuitive but technically more powerful (Figure 10). It seems to be worth investigating how to combine these two interfaces. Finally, we still have much to learn about how to support designers who use visual algorithms.

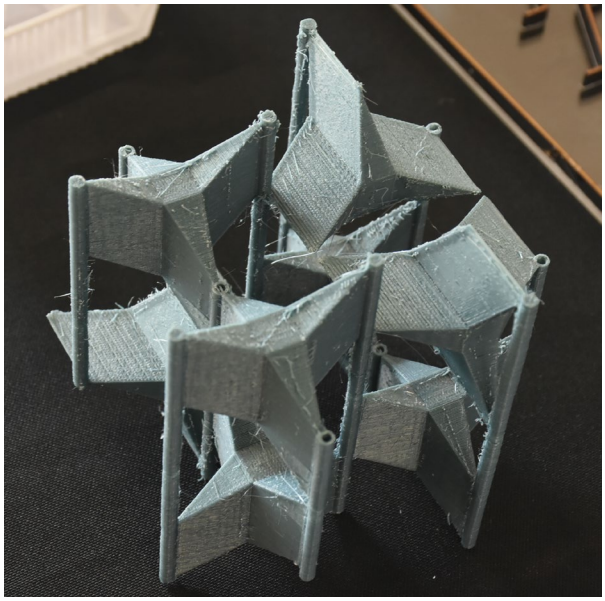

Figure 10. A 3d-printed study. The designer used the less visual Grasshopper-based interface.

The interpreter can be downloaded at http://andrew.li/interpreter/.

4. References.

1) Dy, B. and Stouffs, R.: 2018, Combining geometries and descriptions: a shape grammar plug-in for Grasshopper, in Proceedings of eCAADe 2018, Vol. 2, eCAADe, Brussels, 499-508.
2) Li, A.I.: 2018, A whole-grammar implementation of shape grammars for designers, in Artificial intelligence for engineering design, analysis and manufacturing 32(2), 200-207.
3) Stiny, G.: 1980, Introduction to shape and shape grammars, in Environment and planning B: planning and design 7, 343-351.