Escher: An Engine for Exploring Hierarchical Combinatorial Tilings

John C. Bowers[1] and Dakota Lawson[2]

Department of Computer Science, James Madison University, Harrisonburg, Virginia, USA
[1]bowersjc@jmu.edu [2]lawsonde@dukes.jmu.edu

Abstract

In this paper we present a tool for creating combinatorial tilings defined by finite subdivision rules. Like their better known geometric counterparts (for example Penrose tiles), combinatorial tilings are formed by a finite tile set glued together along their sides. Unlike geometric tilings, however, combinatorial tilings have no inherent geometry and are endowed with geometry after the fact using graph drawing methods. Subdivision tilings are a type of combinatorial tiling that are defined recursively. Such tilings have interesting mathematical properties. Our open source tool, Escher, allows a user to define new tilings, compute properties of the tilings, and produce drawings of them. The Escher engine has both a Python library for exploring tilings using code as well as a simplified language for defining tilings and producing drawings of them that is designed to be accessible to non-programmers.

Introduction

Geometric tilings like the famous Penrose tilings, or the girih tiles found in Turkish-Islamic art, are well known to many interested in the intersection of art and mathematics. Some tilings are *periodic*, meaning they admit some global symmetry, like the square grid or the hexagonal honeycomb; others, like those of Penrose, are *aperiodic*, meaning they admit no global symmetries. Aperiodic tilings are rare enough that their discovery is noteworthy. Recent attention has been given to aperiodic conformal tilings generated by a finite set of subdivision rules [4, 5]. *Escher* is our tool for exploring and creating such tilings.

Start with a finite set of tiles, called *prototiles*, and for each prototile a *subivision rule* that shows how to subdivide the prototile into smaller copies (all scaled by the same scale factor λ). Consider the subdivision tiling called the *chair tiling*, shown in Figure 1. By recursively applying the subdivision rules for each prototile, we obtain a *heirarchical tiling*.

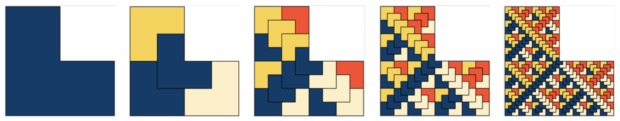
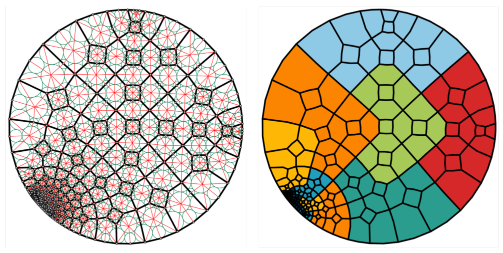
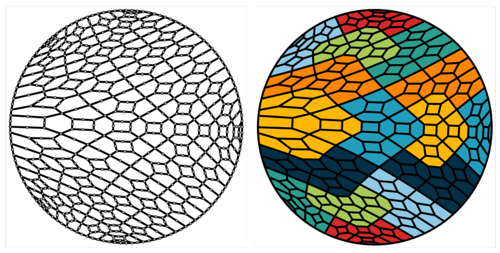


Figure 1: *An initial prototile (left) followed by four levels of recursive subdivision. The left-most two figures display the subdivision rule, which is recursively applied.*

What if we completely forget the geometry of a tiling and instead remember only its combinatorics? A tile becomes an n-cycle of vertices surrounding an abstract face in a planar graph and the subdivision rules define how to refine an n-cycle of some tile type by subdividing combinatorially into further vertices, edges, and faces. This is purely combinatorial–there are no geometric objects present. Note that there is a bit of a subtlety here. In a geometric tiling, all tiles of the same type are geometrically similar. So too in a combinatorial tiling, all tiles must be combinatorially similar. Looking closely at the graph of the edges of the first subdivision of the chair tiling we see that the six cornered chair is subdivided into three chairs each of which (considered as a planar graph) have 6 vertices, but one which must have midpoint vertices introduced along two of its edges creating an 8-cycle. In order to make this subdivision rule purely combinatorial, we must either have two tile types, one a 6-cycle and one an 8-cycle, or simply start by imagining each of the long sides of the chair as subdivided by a midpoint vertex. Combinatorially this makes all of our chairs 8-cycles, and thus we can define the combinatorial version of the rule with only one tile type.



(a)



(b)

Figure 2: *Chair tilings where we "forgot" the geometry before subdividing. (a) A star circle packing of the level-4 tiling (left) and its highlighted aggregate tiles two levels up (right). The black edges are the edges of the tiling. The red edges are the additional edges introduced to triangulate each face before computing a circle packing to obtain a drawing. The circles are shown lightly in green. (b) A Tutte embedding of the level-3 tiling (left) and its highlighted aggregate tiles two levels up.*

We now apply the combinatorial subdivision rules recursively to obtain a purely combinatorial object in the form of a planar graph (which has abstract vertices, edges, and faces, but no geometry). We now want to obtain a geometric tiling from our combinatorial object. Two techniques are implemented in Escher. The first is a *star-subdivision circle packing drawing* of the tiling. Given the planar graph $G = (V, E, F)$ whose faces are all triangles (except the outer face), a *circle packing* is a pattern of interior disjoint circles indexed by the vertex set (C_v for $v \in V$) such that whenever $uv \in E$, the circles C_u and C_v are tangent. According to a variant of the Koebe-Andre'ev-Thurston theorem, there always exists a circle packing for G in which all circles lie on the interior of the unit disk and the boundary vertices of G (those that border the outer face) are internally tangent to the boundary of the unit disk (and this packing is unique up to Möbius transformations that fix the unit disk)[1]. Escher uses the algorithm from [3] to compute this packing. The first step is to subdivide each tile into triangles by introducing a new "star" vertex on the tile's interior and adding edges between this vertex and each of the tile's boundary vertices. The resulting graph is a triangulation and Escher then computes a circle packing on the interior of the unit disk for this triangulation. The second graph drawing technique implemented in Escher is a Tutte embedding (as described in [7]). Here we place all of the vertices of the outer face of the tiling equidistant along the unit circle. Since this places the vertices of the outer face into convex position, there is a *Tutte embedding* of the tiling, which is a planar embedding of the tiling in which each interior vertex is the average (barycenter) of its neighbors. Examples of each are shown in Figure 2.

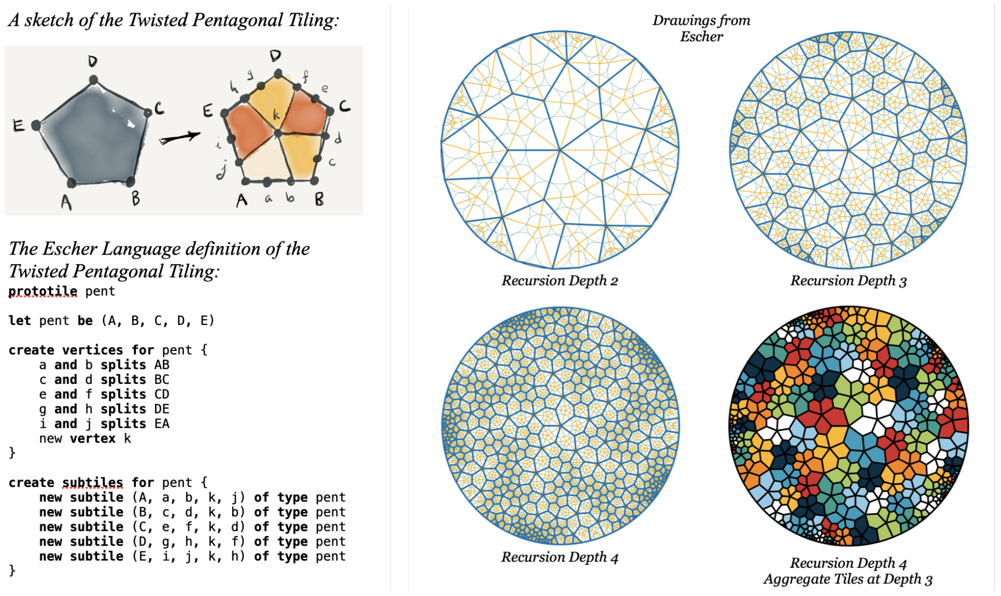
Now for a surprise! Let us select a particular level of the recursive process and for each tile at that level choose a unique tile color at that recursive level. From there we continue to recursively apply subdivisions, except that each tile produced by a subdivision rule now inherits its color from the parent tile whose subdivision created it. This allows us to view *aggregate tiles*, or all the tiles who have a common ancestor. The union of these tiles with a common ancestor at level n is an n-level aggregate tile. Figure 2 show aggregate tiles for both the star circle packing drawings and Tutte drawings of our chair tiling. The chairs have returned! Here the reader may wish to reflect on exactly how surprising this is. In both cases we have forgotten the geometry and retained only the combinatorics of the chair tiling. A "chair" is just an 8-cycle. When we draw it with a graph drawing technique, we should not expect to see chairs at all, and in fact we do not. However,

[1]This is actually a circle packing of the hyperbolic plane and drawn in the Poincaré model, which is why the star vertices in figure 2a appear to be off center–they are at the hyperbolic centers of the circles.

when we inspect the aggregate tiles, somehow the chair-ness of the figure reappears–the combinatorics of the tiling are coding for its geometry! It turns out that this is no fluke of the chair tiling. In fact, Kenyon and Stephenson recently showed in [6] that this surprising result, at least in the case of star circle packing drawings, generalizes–combinatorial subdivision tilings somehow encode geometry into their combinatorics and the aggregate tiles approach their Euclidean tiles geometrically. (The reader may wish to note here that why the same showed up for the Tutte embedding is an intriguing open problem.)

Using Escher

We designed *Escher* to aid in the discovery of new tilings and the exploration of their mathematical properties. Escher has two components–a Python library for defining and computing subdivisions that returns the subdivision in a structure in which additional properties may be explored programmatically, and a simple language that can be used to generate tilings at different recursive levels and produce drawings of them.



The Escher Language definition of the Twisted Pentagonal Tiling:

```
prototile pent

let pent be (A, B, C, D, E)

create vertices for pent {
    a and b splits AB
    c and d splits BC
    e and f splits CD
    g and h splits DE
    i and j splits EA
    new vertex k
}

create subtiles for pent {
    new subtile (A, a, b, k, j) of type pent
    new subtile (B, c, d, k, b) of type pent
    new subtile (C, e, f, k, d) of type pent
    new subtile (D, g, h, k, f) of type pent
    new subtile (E, i, j, k, h) of type pent
}
```

Figure 3: *An example of taking a tiling description of the* twisted pentagonal tiling *from drawing to tiling.*

The goal of the Escher engine is to match closely the way a mathematician explains a subdivision rule using a chalkboard. Consider the chair tiling. To explain the tiling rule for the chair tiling, one would first draw the prototile, and then draw its subdivision into smaller tiles. Escher begins with the same drawing of a subdivision of each prototile. In the subdivision there are three distinguishable types of vertices–vertices that are part of the original prototile, vertices that are obtained by splitting an edge of the original prototile into two or more edges (thus introducing new vertices along the original edge), or vertices that are introduced on the interior of the tile. The user provides names for each of the vertices, both in the initial tile definition and in the subdivision rule.

A prototile object is created for each prototile, which is simply a list of its vertex names. These may optionally be associated with geometric points if the tiling is a geometric tiling, but may be omitted if defining a purely combinatorial tiling. The subdivision rule is specified by: (1) defining which original edges should be split (and into how many sections) and what names to give the new vertices;)(2) giving names to any additional vertices needed on the interior of the prototile; and (3) telling the system which tiles to subdivide into, which requires specifying both type and vertex cycle. An example is shown in Figure 3. Escher then applies the tiling rules recursively to the desired depth given by the user. The output is a modification of a *doubly connected edge list (DCEL)* data structure that additionally stores the recursive tile hierarchy and allows for efficient computation of a variety of tiling properties–for instance, aggregating tiles into their parent super-tiles is very efficient, as is computing properties of interest to the tiling community like collared tiles (tiles labeled by the types of their neighbors). The Escher engine is available open source as part of the `KoebePy` geometry processing library [1].

In addition to the the Python library we have developed a simple language for defining tilings that mimics the programmatic creation of a tiling in Python, but allows the user to define and view it without Python. The user writes a text file describing a tiling and runs the tiler program on it to "compile" a drawing of the tiling as an SVG file. Various parts of the figure are included in different groups so that a standard SVG editor can be used to quickly modify the look of the tiling. This part of the engine is currently at the testing phase and is due to be released publicly in the summer of 2024.

Summary and Conclusions

In this paper we have introduced *Escher*, our engine for programmatically exploring combinatorial hierarchical subdivision tilings. A longer form explanation of the motivations and mathematics involved can be found at [2]. Currently the Escher language only has provisions for producing drawings of tilings, while more sophisticated computation must be done programmatically in Python. In the future we plan to provide a variety of additional tools for tiling researchers, including things like computing and visualizing the entire collared tile set of a tiling within the Escher language itself. The system is also currently limited by the exponential nature of the recursive subdivision rules and we're realistically only able to produce tilings down to a depth of about seven. As future work we plan to explore approximation schemes to allow us to compute parts of a tiling and its drawing at much greater depth (without having to globally compute the entire tiling).

Acknowledgements

Special thanks to Angelo Luna and Nicole Maguire who developed an earlier proof-of-concept for Escher as part of an independent study at James Madison University.

References

[1] *KoebePy Geometry Processing Library*. https://github.com/johncbowers/koebepy. Accessed: 2024-04-29.

[2] J. C. Bowers. *Combinatorial tilings from finite subdivision rules*. https://theobtusegeometer.com/2020/04/08/tilings-from-finite-subdivision-rules-in-koebepy/. Accessed: 2024-04-29.

[3] P. Bowers and K. Stephenson. "A "regular" pentagonal tiling of the plane." *Conformal Geometry and Dynamics of the American Mathematical Society*, vol. 1, no. 5, 1997, pp. 58–86.

[4] P. L. Bowers and K. Stephenson. "Conformal tilings I: foundations, theory, and practice." *Conform. Geom. Dyn.*, vol. 21, no. 1, 2017, pp. 1–63.

[5] P. L. Bowers and K. Stephenson. "Conformal tilings II: Local isomorphism, hierarchy, and conformal type." *Conform. Geom. Dyn*, vol. 23, no. 4, 2019, pp. 52–104.

[6] R. Kenyon and K. Stephenson. "Shape convergence for aggregate tiles in conformal tilings." *Proceedings of the American Mathematical Society*, vol. 147, no. 10, 2019, pp. 4275–4287.

[7] W. T. Tutte. "How to draw a graph." *Proceedings of the London Mathematical Society*, vol. 3, no. 1, 1963, pp. 743–767.