

An Algorithm to Generate Repeating Hyperbolic Patterns

Douglas Dunham

Department of Computer Science

University of Minnesota, Duluth

Duluth, MN 55812-3036, USA

E-mail: ddunham@d.umn.edu

Web Site: <http://www.d.umn.edu/~ddunham/>

Abstract

More than 25 years ago combinatorial algorithms were designed that could replicate each of M.C. Escher's four hyperbolic "Circle Limit" patterns. These algorithms were based on tilings of the hyperbolic plane by regular polygons, and were used to create other artistic hyperbolic patterns. This paper describes a more general algorithm that can generate a repeating pattern of the hyperbolic plane based on a tiling by any convex (finite) polygon.

1 Introduction

Figure 1 below shows the Dutch artist M.C. Escher's *Circle Limit III* pattern with the regular tessellation $\{8, 3\}$ superimposed on it. Figure 2 shows an Islamic-style pattern also based on the $\{8, 3\}$ tessellation, the two kinds of 3-fold rotation points lying at the vertices of regular octagons. The purpose of this paper is to

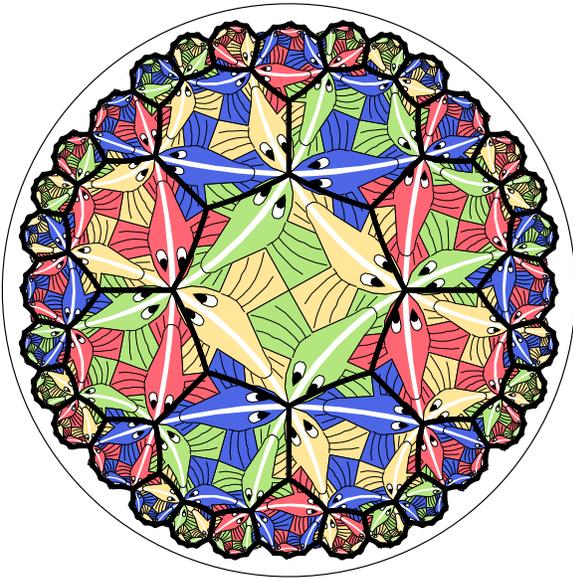


Figure 1: Escher's *Circle Limit III* pattern with the $\{8, 3\}$ tessellation superimposed.

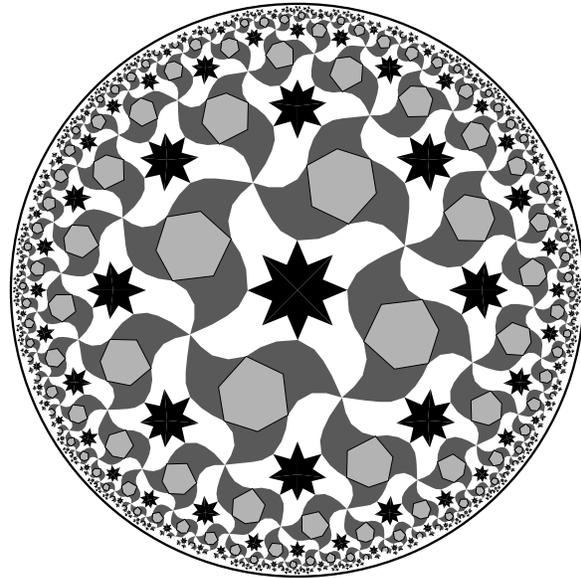


Figure 2: An Islamic-style pattern based on the $\{8, 3\}$ tessellation.

describe an algorithm that can draw patterns based on tilings by a polygon that is not necessarily regular. We can assume that such a polygon is convex, and we will further restrict attention to finite polygons (without points at infinity).

We begin by reviewing some hyperbolic geometry and regular tessellations. Then we describe the new algorithm and show some sample patterns. Finally, we conclude, indicating directions of future research.

2 Repeating Patterns, Tessellations, and Hyperbolic Geometry

The 2-dimensional “classical geometries” are the sphere, the Euclidean plane, and the hyperbolic plane, of constant positive, zero, and negative curvature respectively. A *repeating pattern* in one of the classical geometries is a regular arrangement of copies of a basic subpattern or *motif*. One fish forms the motif in *Circle Limit III* (disregarding color), and half of one white fish plus half of an adjacent black fish forms a motif in Figure 3.

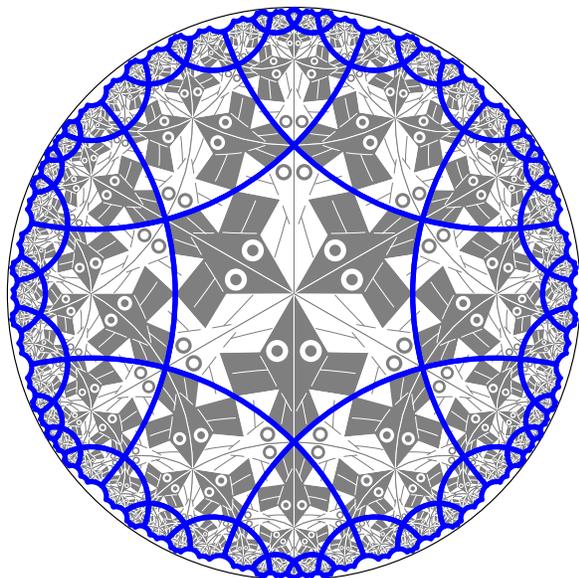


Figure 3: The tessellation $\{6, 4\}$ underlying the *Circle Limit I* pattern and superimposed on it.

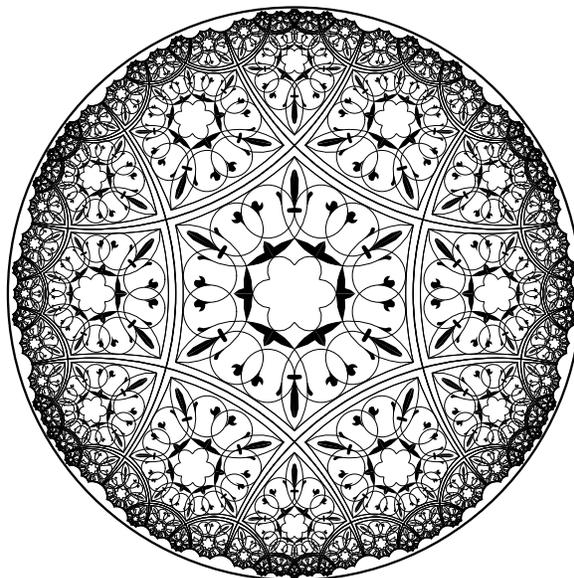


Figure 4: An arbesque pattern based on the tessellation $\{6, 4\}$.

We let $\{p, q\}$ denote the *regular tessellation* by regular p -sided polygons, q of which meet at each vertex. For $\{p, q\}$ to be a tessellation of the hyperbolic plane, it is necessary that $(p - 2)(q - 2) > 4$, which follows from the fact that the sum of the angles of a hyperbolic triangle is less than π . Figures 1 and 3 show the tessellations $\{8, 3\}$ and $\{6, 4\}$ superimposed on *Circle Limit III* and *Circle Limit I* respectively. Similarly, *Circle Limit II* and *Circle Limit IV* are also based on the respective tessellations $\{8, 3\}$ and $\{6, 4\}$. In addition to Escher’s *Circle Limit* patterns, other hyperbolic patterns have been generated by a program based on regular tessellations [3, 4].

The mathematician David Hilbert proved in 1901 that there was no smooth distance-preserving embedding of the whole hyperbolic plane into Euclidean 3-space. Thus we must rely on models of hyperbolic geometry that distort distance. Escher used the *Poincaré disk model* for his “*Circle Limit*” patterns. In this model, hyperbolic points are just the (Euclidean) points within a Euclidean bounding circle. Hyperbolic lines are represented by circular arcs orthogonal to the bounding circle (including diameters). For example, the backbone lines and other features of the fish lie along hyperbolic lines in Figure 3. One might guess that the backbone arcs of the fish in *Circle Limit III* are also hyperbolic lines, but this is not the case. They are *equidistant curves* in hyperbolic geometry: curves at a constant hyperbolic distance from the hyperbolic line with the same endpoints on the bounding circle, and are represented by circular arcs not orthogonal to the bounding circle.

3 The General Replication Algorithm

Given a motif, the process of transforming it around the hyperbolic plane and drawing the transformed copies to create a repeating pattern is called *replication*. To our knowledge, the first combinatorial replication algorithm was invented more than 25 years ago. That algorithm was based on regular tessellations and was used to replicate each of Escher’s four “Circle Limit” patterns. This was discussed in the paper *Creating Repeating Hyperbolic Patterns* [1]. Unfortunately there is an error in the Pascal pseudocode describing the algorithm. There is a correct algorithm given in the later paper *Hyperbolic Symmetry* [2]. We call these algorithms *combinatorial* since they only depend on the values of p and q in the underlying tessellation $\{p, q\}$, and how the edges of the polygon are matched up.

A *fundamental region* for the symmetry group of a repeating pattern is a closed topological disk such that transformed copies of it cover the plane without gaps or overlaps. The motif of an Escher pattern can usually be taken to be a fundamental region for its symmetry group — for instance a fish for *Circle Limit III*. However, for ease of understanding, we use a convex polygon as a fundamental region instead, as can always be done [6]. Of course such a polygonal fundamental region will contain exactly those pieces of the motif needed to reconstruct it. Thus every repeating pattern has an underlying tessellation by convex polygons. Figure 5 shows a natural polygon tessellation underlying a transformed version of the *Circle Limit III* pattern. Figure 6 shows the polygon tessellation by itself, with a fundamental polygon in heavy lines.

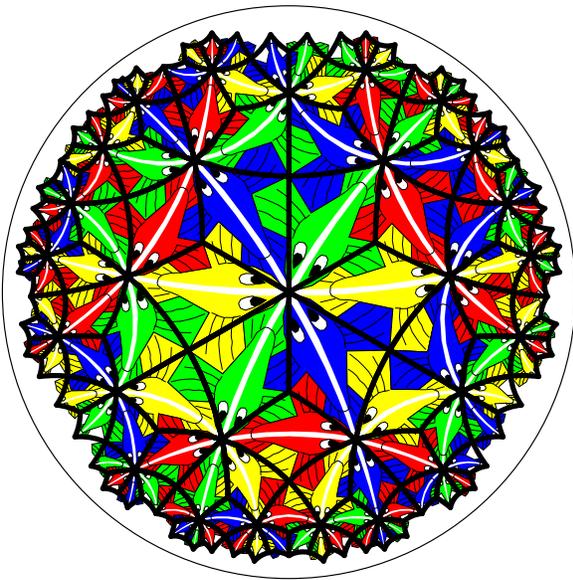


Figure 5: A polygon tessellation superimposed on the transformed *Circle Limit III* pattern.

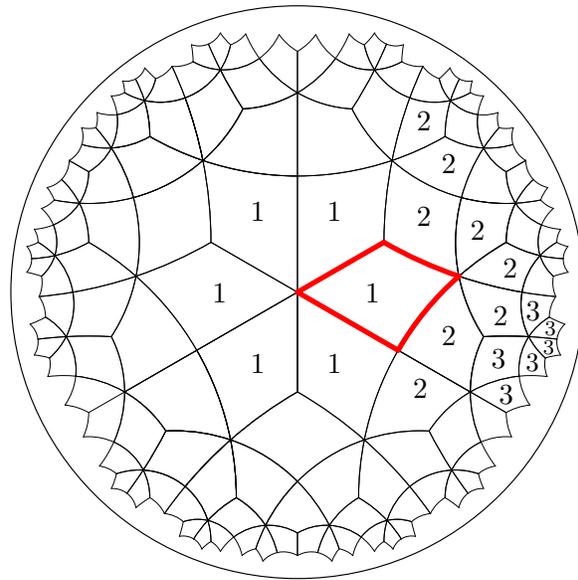


Figure 6: The polygon tessellation, with a fundamental polygon emphasized and parts of layers 1, 2, and 3 labeled.

We can see in Figure 6 that the polygons are arranged in *layers* (often called *coronas* in tiling literature). The layers are defined inductively. The first layer consists of all polygons with a vertex at the center of the bounding circle. The $k + 1^{st}$ layer consists of all polygons sharing an edge or a vertex with a polygon in the k^{th} layer (and not in any previous layers). In Figure 6, all the first layer polygons are labeled “1”, and some of the second and third layer polygons are labeled “2” and “3” respectively. The algorithm we describe will replicate all the layers up to a cutoff limit (of course theoretically there are an infinite number of layers).

As in the p, q notation for regular tessellations, we specify the fundamental polygon by the number of edges (= number of vertices), p , and the number of polygons (the valence) at each vertex, q_1, q_2, \dots, q_p .

We denote this tessellation by $\{p; q_1, q_2, \dots, q_p\}$. Thus the interior angle at the i^{th} vertex is $2\pi/q_i$. The condition that $\{p; q_1, q_2, \dots, q_p\}$ be hyperbolic is that $\sum_{i=1}^p \frac{1}{q_i} < \frac{p}{2} - 1$ (which generalizes the condition $(p-2)(q-2) > 4$ for regular tessellations). Figure 7 below shows a $\{4; 6, 4, 5, 4\}$ tessellation with corresponding valence array $q[\] = (6, 4, 5, 4)$ (that is, $q[1] = q_1 = 6, q[2] = q_2 = 4$, etc.). The edges of the fundamental polygon are labeled 1, 2, 3, and 4. In general, the i^{th} vertex is the right-hand vertex of the i^{th} edge when looking outward from the center of the polygon.

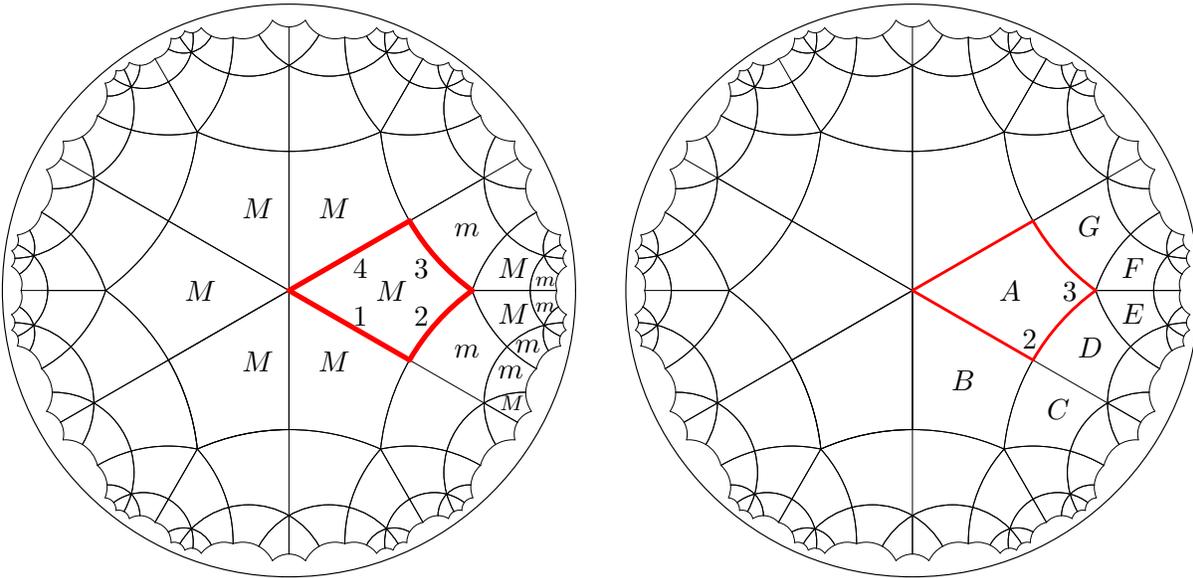


Figure 7: The $(6, 4, 5, 4)$ tessellation showing a fundamental polygon, and with some minimally exposed and maximally exposed polygons marked with m and M respectively. **Figure 8:** A first-layer polygon A , two of its vertices labeled 2 and 3, and some polygons adjacent to A .

To describe the replication algorithm, we define the *exposure* of a polygon by its relation to the next layer. In Figure 7, we say the polygons labeled m , sharing only one edge with the next layer, have *minimum exposure*, and the polygons labeled M , sharing two edges with the next layer, have *maximum exposure*. In particular all the layer 1 polygons have maximum exposure.

The algorithm will draw the first layer at the top level and then recursively draw the other layers by drawing all the polygons adjacent to the exposed edges and vertices of the current polygon. The idea of the recursive part is to draw all the polygons in the next layer that are adjacent to the right vertices (looking outward) of exposed edges of the polygon except that some of the first polygons are skipped since they will be drawn by the recursive call to an adjacent polygon in the current layer. Also, about an exposed vertex, some of the first polygons are skipped since they would have been drawn by the previous vertex (if there was one). Figure 8 shows how the recursive calls work for the first-layer polygon A . Polygon D is the only one drawn adjacent to the vertex 2, because B is in the same layer as A and C will be drawn by a recursive call from B . Thus B and C will be skipped over and D will be the only polygon adjacent to vertex 2 to be drawn by the recursive call. Similarly, about vertex 3, D will be skipped (it was just drawn), and then E, F , and G will be drawn. It seems easier to skip polygons at the beginning rather than at the end (it is necessary to skip some polygons to avoid drawing them more than once).

We break up the algorithm as follows: an initial “driver” procedure `replicate()` calls the recursive procedure `replicateMotif()`. For each edge of the fundamental polygon there is a transformation

of the polygon across that edge, and thus an array of p such transformations, `edgeTran[]`. We assume `edgeTran[]`, `p`, `q[]`, and `maxLayers` to be global for simplicity. Also, we will use a function `addToTran()` (discussed below) that computes an extension of a given transformation from the center by one of the `edgeTran[]`'s. Here is the C-like pseudocode for `replicate()`:

```
replicate ( motif )
{
  for ( j = 1 to q[1] )
  {
    qTran = edgeTran[1] ; // qTran = transformation about the origin

    replicateMotif ( motif, qTran, 2, MAX_EXP )

    qTran = addToTran ( qTran, -1 ) ; // -1 ==> anticlockwise order
  }
}
```

The first operation in `replicateMotif()` is to draw a transformed copy of the motif, so this guarantees that the first layer gets drawn.

In order to understand the code for `addToTran()`, we note that our transformations contain (in addition to a matrix) the orientation and an index, `pPosition`, of the edge across which the last transformation was made (`edgeTran[i].pPosition` is the edge that is matched with edge i). Here is the code for `addToTran()`:

```
addToTran ( tran, shift )
{
  if ( shift % p == 0 ) return tran ;
  else return computeTran ( tran, shift ) ;
}
```

where `computeTran()` is:

```
computeTran ( tran, shift )
{
  newEdge = ( tran.pPosition + tran.orientation * shift ) % p ;
  return tranMult ( tran, edgeTran[newEdge] ) ;
}
```

“%” is the mod operator, and `tranMult(t1, t2)` multiplies the matrices and orientations, and sets the `pPosition` to `t2.pPosition`, and returns the result.

The basic idea of `replicateMotif()` is to first draw the motif, then iterate over the exposed vertices (except the last one which will be handled by an adjacent polygon in the current layer), and for each exposed vertex to iterate about it, calling `replicateMotif()` to draw the exposed polygons/motifs. There are five global 2-element arrays that are used in `replicateMotif()`:

```
pShiftArray[] = { 1, 0 } ;
verticesToSkipArray[] = { 3, 2 } ;
qShiftArray[] = { 0, -1 } ;
polygonsToSkipArray[] = { 2, 3 } ;
exposureArray[] = { MAX_EXP, MIN_EXP } ;
```

The first two depend on the exposure (one of the values MIN_EXP, MAX_EXP), and give the initial shift and number of vertices to skip respectively. The next two depend on whether this is the first vertex to be done or not, and give the initial shift about a vertex and the number of polygons to skip respectively. The last one, exposureArray[], gives the exposure of the new polygon, and depends on whether this is the first polygon about the current vertex to be done or not.

Here is the code for replicateMotif():

```

replicateMotif ( motif, initialTran, layer, exposure )
{
  drawMotif ( motif, initialTran ) ; // Draw the transformed motif
  if ( layer < maxLayers )
  {
    pShift = pShiftArray[exposure] ; // Which vertex to start at
    verticesToDo = p - verticesToSkipArray[exposure] ;

    for ( i = 1 to verticesToDo ) // Iterate over vertices
    {
      pTran = computeTran ( initialTran, pShift ) ;
      first_i = ( i == 1 ) ;
      qTran = addToTran ( pTran, qShiftArray[first_i] ) ;
      if ( pTran.orientation > 0 ) vertex = (pTran.pposition-1) % p ;
      else                          vertex = pTran.pposition ;
      polygonsToDo = q[vertex] - polygonsToSkipArray[first_i] ;

      for ( j = 1 to polygonsToDo ) // Iterate about a vertex
      {
        first_j = ( j == 1 ) ;
        newExposure = exposureArray[first_j] ;

        replicateMotif ( motif, qTran, layer + 1, newExposure ) ;
        qTran = addToTran ( qTran, -1 ) ;
      }
      pShift = (pShift + 1) % p ; // Advance to next vertex
    }
  }
}

```

where drawMotif() simply multiplies each of the point vectors of the motif by the transformation and draws the transformed motif. We note that the algorithm above is a simplified version that draws multiple copies of the motif for the special cases $p = 3$ or $q_i = 3$.

Since the algorithm above only depends on $\{p; q_1, q_2, \dots, q_p\}$ and which edge of the fundamental polygon is matched by transformation across that edge, we consider it to be a combinatorial algorithm. There are other kinds of replication algorithms. Two algorithms work by creating a list of transformations without duplicates, thus avoiding drawing the same copy of the motif twice. The first algorithm uses the theory of automatic groups to eliminate words of “generator” transformations that reduce to the same transformation that has already been used [5]. The second algorithm to eliminate duplicate transformations uses the fact that they are often represented by real $n \times n$ matrices. The matrices used so far are kept in a list. Possible new transformations are created by composing each matrix in the list with the generator matrices (a small set). Each new transformation is compared with every transformation in the list. If the new matrix and some

“list matrix” are within some small distance in \mathbf{R}^{n^2} , the new matrix is a duplicate and is discarded, otherwise it is added to the list. The same technique works for complex matrices.

4 Sample Patterns

Figure 9 shows a hyperbolic version of Escher’s notebook drawing number 85 based on the “three elements”: *air* represented by a bat, *earth* represented by a lizard, and *water* represented by a fish (page 184 of [7]). Figure 9 is based on the regular tessellation $\{6, 4\}$ (superimposed on it), and thus can be generated by the simpler algorithm described in [2]. In this pattern, three bats meet head-to-head, four lizards meet nose-to-nose, and four fish meet nose-to-nose. The simpler algorithm suffices because both the lizards and fish meet four at a time. In Escher’s notebook drawing, three of each animal meet at their heads. Figure 10 shows the “three elements” pattern based on the $\{3; 3, 4, 5\}$ tessellation, with different numbers of animals meeting head-to-head, and is drawn by the more general algorithm.

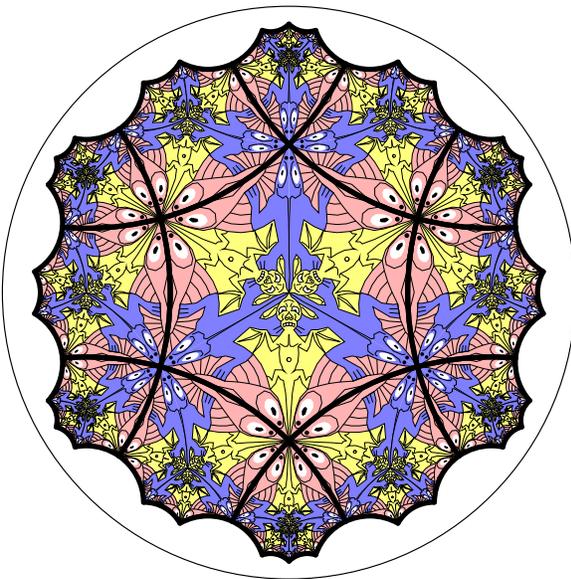


Figure 9: A “three elements” pattern based on the regular tessellation $\{6, 4\}$.

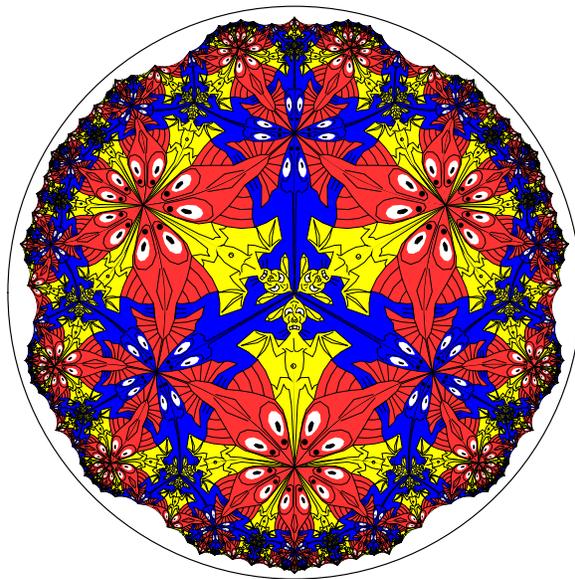


Figure 10: A “three elements” pattern with different numbers of animals meeting at their heads.

Figures 11 and 12 below show two other variations on the “three element” pattern, with different numbers of animals meeting head-to-head, that can only be drawn by the new algorithm. Figure 11 is based on the $\{3; 3, 5, 4\}$ tessellation, and Figure 12 is based on the $\{3; 4, 3, 5\}$ tessellation.

5 Conclusions and Directions for Future Research

We have described an algorithm that can draw repeating patterns of the hyperbolic plane in which the fundamental region is a finite polygon, and we have shown some patterns drawn with the use of this algorithm. It would be interesting to extend this algorithm to allow some of the vertices of the fundamental polygon to be points at infinity. A special program has been written that can transform a motif from one triangular fundamental region to another with different vertex angles. It would be useful to generalize this capability at least to transform between fundamental polygons with the same number of sides but different angles. Transforming between polygons with different numbers of sides would be desirable, but seems more difficult. Another difficult problem is that of automatically generating patterns with color symmetry.

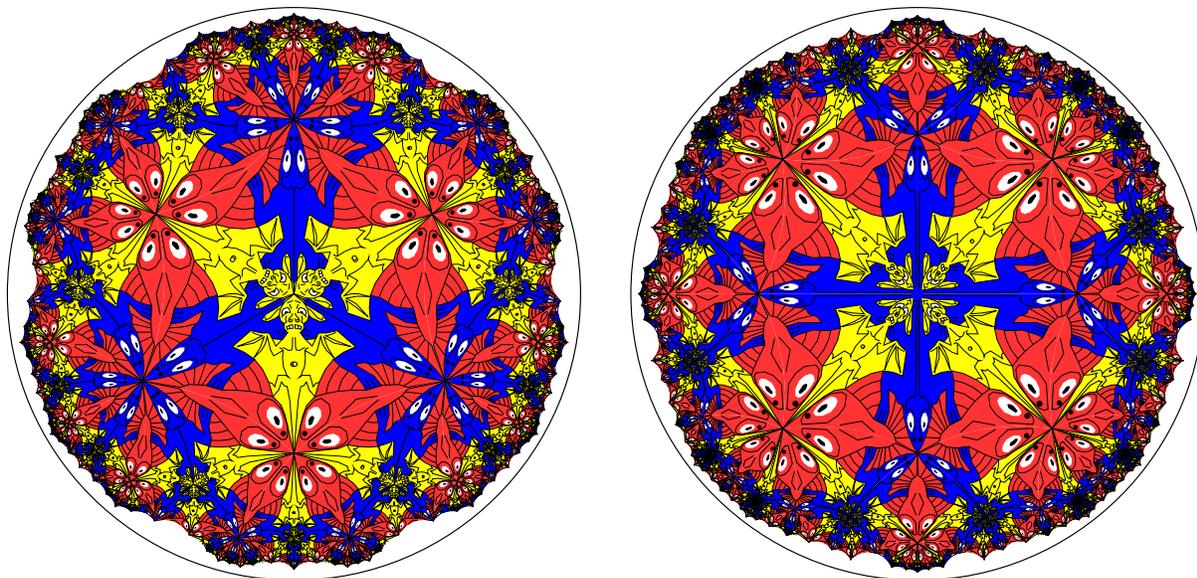


Figure 11: A “three elements” pattern with 3 bats, 5 lizards, and 4 fish meeting at their heads. **Figure 12:** A “three elements” pattern with 4 bats, 3 lizards, and 5 fish meeting at their heads.

Acknowledgments

I would like to thank Lisa Fitzpatrick and the staff of the Visualization and Digital Imaging Lab (VDIL) at the University of Minnesota Duluth. I would also like to thank the reviewers for their helpful suggestions.

References

- [1] D. Dunham, J. Lindgren, and D. Witte, Creating repeating hyperbolic patterns, *Computer Graphics*, Vol. 15, No. 3, August, 1981 (Proceedings of SIGGRAPH '81), pp. 215–223.
- [2] D. Dunham, Hyperbolic symmetry, *Computers & Mathematics with Applications*, Vol. 12B, Nos. 1/2, 1986, pp. 139–153. Also appears in the book *Symmetry* edited by István Hargittai, Pergamon Press, New York, 1986. ISBN 0-08-033986-7
- [3] D. Dunham, Artistic Patterns in Hyperbolic Geometry, *Bridges 1999 Conference Proceedings*, pp. 239–249, 1999.
- [4] D. Dunham, Hyperbolic Islamic Patterns – A Beginning *Bridges 2001 Conference Proceedings*, pp. 247–254, 2001.
- [5] D.B.A. Epstein, with J.W. Cannon, *Word Processing in Groups*, A. K. Peters, Ltd., Natick, MA, USA, 1992. ISBN 0867202440
- [6] A.M. Macbeath, The classification of non-Euclidean plane crystallographic groups, *Canadian J. Math*, **19** (1957), pp. 1192–1205.
- [7] D. Schattschneider, *M.C. Escher: Visions of Symmetry*, 2nd Ed., Harry N. Abrams, Inc., New York, 2004. ISBN 0-8109-4308-5