Chapter 4

ESCHER'S TILINGS

4.1 Introduction

M. C. Escher, more than any other individual throughout history, brought forth the great beauty that lies at at the intersection of art and mathematics. The immense popularity of his graphic art shows no signs of waning, and his imagery continues to appear regularly everywhere people casually place eye-catching art. Although he downplayed his abilities as artist and as mathematician, his work has always been a source of profound inspiration to both. It drives the artist to seek out the aesthetic possibilities of geometry and pushes the mathematician to invent new machinery to explain the designs he created through what he considered non-technical means.

Figure 4.1 M.C. Escher in a self-portrait. *M.C. Escher's "Self-portrait" ©2000 Cordon Art B.V. – Baarn – Holland. All rights reserved.*

Bruno Ernst provides a classification of the many ways that mathematics played a role in Escher's work. Any one of these suggests a fruitful avenue for exploration of the geometric aesthetic. I have chosen to focus on the regular division of the plane, on the many tesselations he created.

Escher had a lifelong obsession with the regular division of the plane, an obsession that in part can trace its lineage back through the Islamic star patterns of the previous chapter. Escher made several visits to the Alhambra and other monuments in Spain. He was deeply inspired by the interlocking geometric forms of the Moors but felt it a pity that they were forbidden by their

religion from depicting real-world objects in their art [50].[1] He undertook as a personal quest the reinvention of geometric art, substituting easily-recognized motifs such as animal forms for the purity of the Moorish rosettes and polygons.

By the time Escher began studying the regular division of the plane in the first half of the twentieth century, tiling as an art form had passed mostly into history, to be replaced by the growing development of a systematic mathematical theory. Escher came along at the right time. He was born just as physicists and mathematicians were beginning to categorize the sorts of symmetries he used. One of his first exposures to the mathematics of symmetry was when his half brother, a geologist, pointed him at a paper by Polyà.

Escher arrived at each of his interlocking animal forms after a great deal of tinkering and manipulation. Over the years, he became more proficient at inventing new arrangements of motifs, developing his own "layman's theory" of tilings to track the ground he had covered and suggest new directions for exploration. He managed over his career to produce a notebook with more than a hundred of these ingenious, playful designs. His notebooks are lovingly reproduced by Schattschneider [124].

Most of Escher's work is not ornamental in the traditional sense. The regular division drawings in his notebook were either intended purely as exercises, or as studies for future prints. Later in his life, when Escher attained a certain measure of celebrity, he did receive several commissions for ornamental installations around Holland, and he covered architectural surfaces with patterns in paint, inlaid wood, and ceramic tile (see the notes in Schattschneider [124] for photographs of some of these installations). Often he took a periodic pattern and wrapped one of the directions of periodicity into a circle, resulting in a cylindrical pattern that could be used to cover a column. Schattschneider also points out that even if Escher did not focus his efforts on ornamental design, his work has had a great influence in the decorative arts [124, Page 281].

In this chapter, I develop a suite of mathematical and software tools that enable the construction and manipulation of Escher-style tilings of the plane. The mathematical tools are based on the theory of isohedral tilings, as developed by Grünbaum and Shephard and presented in Section 2.4.

---

[1] As Section 3.1 points out, the prevalence of geometric patterns in Islamic art is not fully explained by a simple religious prohibition. However, this misunderstanding was common in Europe at the time that Escher was exposed to Islamic art.

The main focus of the chapter is *Escherization*: an algorithm that, given an arbitrary polygonal shape, attempts to find a tiling of the plane using a prototile that resembles the shape. Escherization takes over the task of searching through the complex space of tilings, providing an outline that an artist can manipulate and decorate. I then adapt Escherization to handle more complex cases, namely where there are two goal shapes and the objective is to create a dihedral tiling of the plane. I also consider the problem of creating Escher-like tilings using Penrose's aperiodic tile sets, tiles that Escher might very well have enjoyed working with had they been discovered during his lifetime.

4.2 Related work

Software specifically geared towards the construction of tilings of the plane has been around for at least twenty years. Chow had a very successful FORTRAN program [21] that let the user input the portion of a tile's boundary that is "independent," *i.e.*, not constrained to some other part of the boundary through a symmetry of the tiling. The program then filled in the remaining part of the tile and replicated it in the plane.

For many years, Kevin Lee has offered a commercial software package called TesselMania! that makes it easy to draw and decorate Escher-like tilings. His system is geared towards the use of tilings as a tool for mathematics education, and the most recent version of TesselMania! includes tutorials, games and puzzles designed for teaching concepts of geometry.

Tupper's Tess [87] has traditionally allowed the user to create drawings belonging to the frieze and wallpaper groups, in a manner similar to the Geometry Center's Kali [5]. Recently, he modified Tess to support a set of tilings directly. Like TesselMania!, Tess is geared towards pedagogical use.

The three programs above are all based on the 28 *Heesch types* [80], which are in some sense precursors to the isohedral tilings that do not take into account the additional internal symmetries that a prototile may be forced to have. Roughly speaking, each Heesch tiling represents a set of isohedral types related through a hierarchy of increasingly symmetric tile shapes, and so the Heesch tilings are just a coarse-grained view of the isohedral tilings. Schattschneider reproduces a table of the 28 Heesch types in her book [124, Page 326]. In principle, much of the work in this chapter could have been based on the Heesch types rather than the isohedral types. However, the availability, thoroughness, and clarity of Grünbaum and Shephard's presentation of the isohedral tilings make

them the preferable choice for this research.

Escher created a small number of carved wooden sculptures featuring spherical interpretations of his tesselations. Using these as a starting point, Yen and Séquin [140] created an "Escher Sphere Construction Kit," a system that allows the user to design ornamental spherical tilings much as one could create Euclidean tilings in the systems above. As an added feature, the tilings they create can be exported to rapid prototyping hardware and constructed as real artifacts, as was done with spherical Islamic star patterns in Section 3.9. They have created many attractive spherical patterns. Some are fabricated in one piece, others tile by tile, assembled by gluing each tile to the surface of a ball.

4.3 Parameterizing the isohedral tilings

We now return to the subject of the isohedral tilings first described in Section 2.4.1. The isohedral tilings turn out to have a balance of mathematical, algorithmic, and aesthetic properties that make them an excellent choice in the search to create new designs in the spirit of Escher. We use them as the basis for almost all of the work in this chapter.

Escher rarely strayed from the isohedral tilings when developing his tesselations. Mathematically, every monohedral tiling produced by his "layman's theory" is isohedral. Only one of his roughly 140 notebook drawings is anisohedral, and that drawing was based on a tiling that Penrose showed to Escher specifically to demonstrate the existence of an anisohedral prototile [118]. Even his periodic tesselations with multiple motifs are ultimately based on the isohedral tilings, for he constructed such tilings by splitting an isohedral prototile into multiple pieces. As a further justification for our decision to use IH, we may appeal to Grünbaum's argument that in building a pattern, artists and designers focus on the relationship between a motif and its neighbours, rather than that between a motif and the whole [64]. The isohedral tilings are precisely those monohedral tilings in which a tile's relationship to its neighbours completely determines the complete structure of the tiling.

On the practical side, the combinatorial structure of the isohedral tilings allows us to construct simple data structures to represent them and efficient algorithms to manipulate them. Building upon the mathematical treatment of the isohedral tilings given in Section 2.4.1, this section and the next

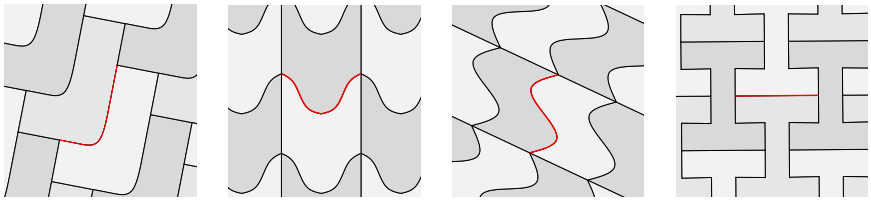


Figure 4.2 Examples (from left to right) of **J**, **U**, **S** and **I** edges. In each case, the tiling edge with the given shape is highlighted in red.

one develop the computational tools needed to work with IH, the space of isohedral tilings. I begin here by providing a parameterization of the space of shapes of isohedral prototiles. In the next section, I will use that parameterization to construct a software library.

Within the tilings of a single isohedral type, prototiles are distinguished from each other by their shapes, determined by the positions of the tiling vertices and the shapes of the curves that join them. In order to move from the combinatorial description of isohedral tilings to a geometric one, we must understand how incidence symbols dictate the range of possible prototile shapes for a given isohedral type. We parameterize the space of isohedral tilings by giving, for each type, an *edge shape parameterization* and a *tiling vertex parameterization*. The former encodes the minimal non-redundant geometric information sufficient to reconstruct the tiling edges. The latter determines the legal configurations of tiling vertices.

4.3.1 Edge shape parameterization

The constraints on the shapes of tiling edges in an isohedral tiling are simple to describe. Although the underlying choice of how to represent a curve is left open, the tiling's symmetries imply a great reduction in the tiling edges' degrees of freedom. These constraints can be extracted directly from the tiling's incidence symbol. We enumerate the four cases for the structure of a tiling edge. For each case, Figure 4.2 shows a tiling with such an edge.

If some directed edge is adjacent to itself without a flip, then a tile's neighbour across that edge is adjacent through a half-turn. This rotation forces the edge shape to itself be symmetric through a half-turn about its centre. We call such an edge an **S** edge as a visual mnemonic. Only half of an **S**

edge is free; the other half must complete the rotational symmetry.

An undirected edge must look the same starting from either end, meaning it must have a line of mirror symmetry through its midpoint. If the edge is adjacent to any edge other than itself, it is free to take on any curve with this bilateral symmetry. We call it a **U** edge. Again, only half of a **U** edge is free.

If an undirected edge is adjacent to itself, or if a directed edge is adjacent to itself with a change in sign, that edge must have both **S** symmetry and **U** symmetry. The only shape that has both is a straight line, leading us to call such an edge an **I** edge.

The remaining case is when a directed edge is adjacent to some other directed edge. Such an edge is free to take on any shape, and we call it a **J** edge.

Note also that if an edge x is adjacent to an edge y, then x and y have the same shape (even though they have different names). In this case, we need only represent one tiling edge, since the other is entirely constrained to it. Thus, referring back to the derivation presented in Figure 2.16, the tiling edges of IH16 can be summarized by one curve: the shape of the edge labeled b. Edges labeled a are **I** edges and have no degrees of freedom, and edges labeled c are constrained to b.

4.3.2 Tiling vertex parameterization

Like the shape vertices, tiling vertices cannot move entirely independently of each other. Moving one tiling vertex forces the others to move to preserve the tiling. The exact nature of this movement depends on the tiling type in question. The incidence symbol for a tiling type implies a set of constraints on the tiling polygon's edge lengths and interior angles. Any tile of that type will have a tiling polygon that obeys those constraints.

If we hope to build a generative model of isohedral tilings, it is not sufficient merely to recognize the constraints on the shape vertices: we need a way to explicitly navigate the space of legal tiling polygons. For each isohedral type we need a parameterization of the tiling vertices for tilings of that type. The parameterization should be *complete*, in the sense that for every legal configuration of tiling vertices, there is a set of parameters that generates that configuration. We also require it to be *consistent*, in the sense that every set of parameters generates legal tiling vertices.

We have developed a set of parameterizations for the isohedral types that we believe to be

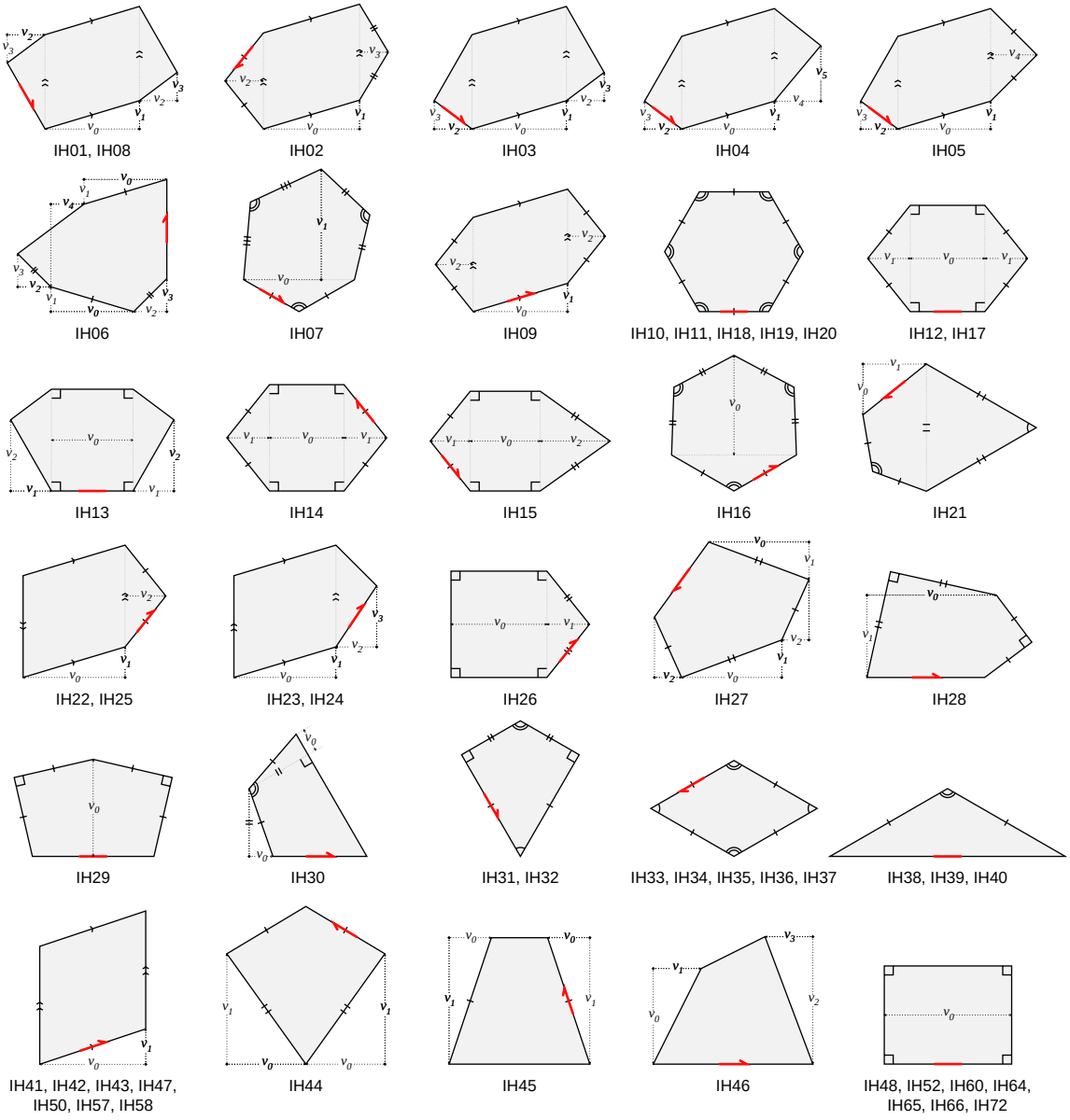


Figure 4.3 The complete set of tiling vertex parameterizations for the isohedral tilings. In each tile, the edge marked with a red line is the first edge in the tiling type's incidence symbol. When that first edge is directed, the red line has an arrowhead. Labelled dotted lines represent parameter values, and are horizontal or vertical (with the exception of one guide line in the diagram for IH30). Since the diagrams are scale independent, distances that do not depend on parameters can be taken to have unit length. Tile edges cut with the same number of short lines have the same length, and edges cut with chevrons are additionally parallel. A single arc, a small square, and a double arc at vertices represent $60°$, $90°$, and $120°$ angles, respectively.

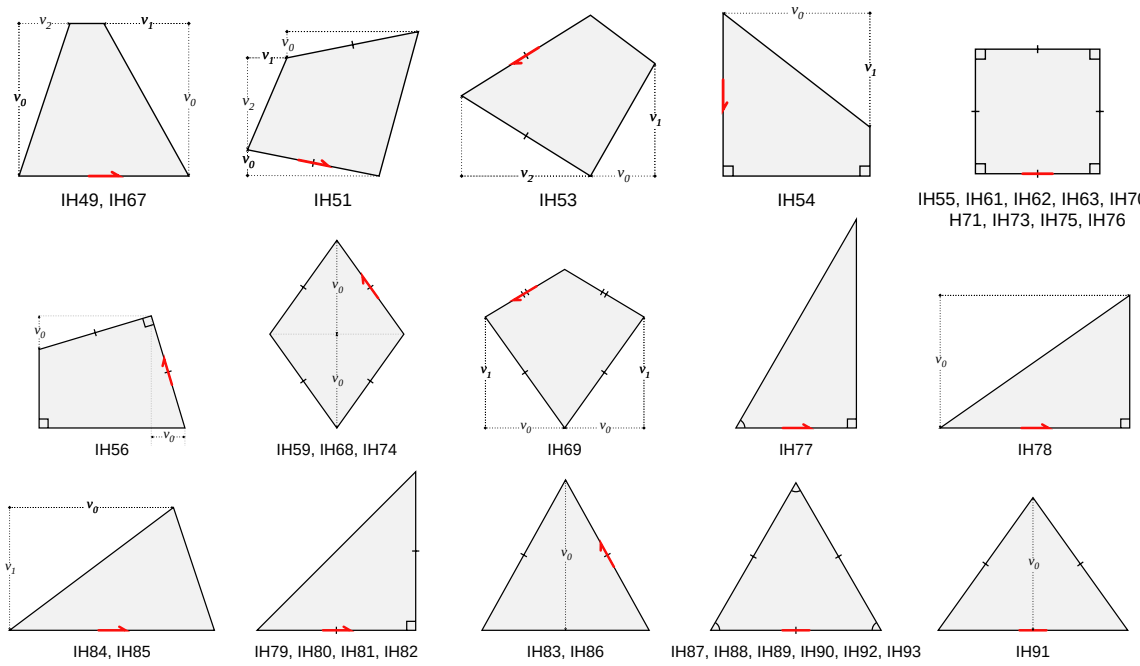


Figure 4.3 The complete set of tiling vertex parameterizations for the isohedral tilings (continued).

consistent and complete, though the history of IH has certainly experienced its share of flawed analyses [68, Section 6.6]. The parameterizations were derived by determining angle and length constraints from the incidence symbols and parameterizing the unconstrained degrees of freedom. In some cases, parameterizations are shared between tiling types: nine tiling types have squares as tiling polygons (implying a parameterization with zero parameters), and seven have parallelograms (implying two parameters). These easy parameterizations are balanced by tiling types with one-of-a-kind structure that can take some thought to derive. In all, the 93 isohedral types require 45 different parameterizations. Diagrams of the parameterizations appear in full in Figure 4.3.

To our knowledge, no tiling vertex parameterizations have ever been given specifically for IH. They represent a nontrivial extension to the table of information about IH found in Grünbaum and Shephard. Previously, Heesch and Kienzle provided tiling vertex parameterizations for the 28 Heesch tiling types [80]. Our parameterizations can be seen as an elaboration of theirs; each Heesch type is identical to one of the isohedral types, and for those types the parameterizations coincide.

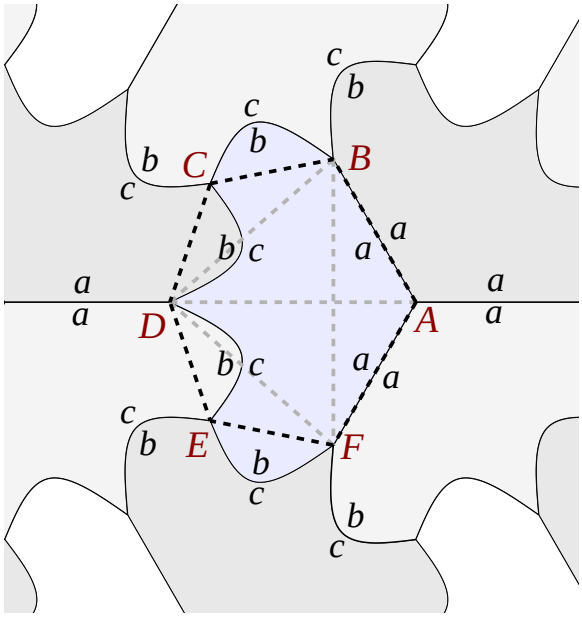


Figure 4.4 The diagram used to establish a tiling vertex parameterization for IH16. For simplicity, the arrows indicating edge direction have been left out of the diagram.

The remaining isohedral types have parameterizations where degrees of freedom are coalesced to yield more symmetric tiling polygons. Because our parameterizations provide information directly tied to each individual isohedral type, they are still useful in applications of tiling theory.

Often, determining the tiling vertex parameterization for an isohedral type is easy. Take, for example, IH41. The table provided by Grünbaum and Shephard tells us that every tiling of this type will have symmetry group $p1$, consisting simply of translations in two directions. Furthermore, each translational unit contains a single aspect. These two facts imply that the tiling polygon must be able to function as a translational unit of the tiling, meaning that it must be a parallelogram.

To give the flavour of a more complicated parameterization, here is a sketch of the derivation for IH16 (see Figure 4.4). We begin by placing at least enough tiles to completely surround one central tile, and marking up the tiles with the labels from the tiling's incidence symbol. Now consider the situation at tiling vertex A. This vertex is surrounded by three copies of the same angle from three different tiles, namely $\angle FAB$, the angle between the a edges. It follows that the tiling polygon must have a 120° angle at that vertex. The same observation applies to vertices C and E. Thus,

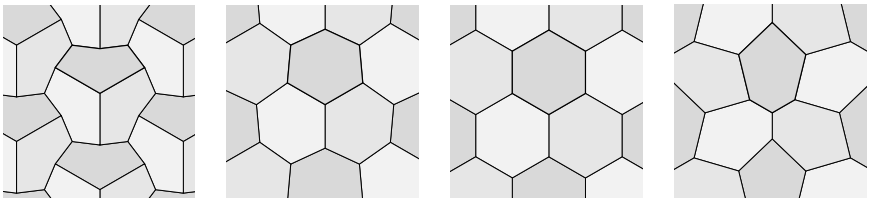


Figure 4.5 Some examples of IH16 with different values for the single parameter in its tiling vertex parameterization.

$\triangle FAB$, $\triangle BCD$, and $\triangle DEF$ are all 120° isosceles triangles. Because these isosceles triangles can be constructed given only the edge opposite the 120° angle, the tiling polygon depends entirely on the "skeleton" triangle $\triangle BDF$. Furthermore, the incidence symbol reveals a line of bilateral symmetry in the tile across $\overline{AD}$, forcing $\triangle BDF$ to be isosceles. The only degrees of freedom left in the tiling polygon are the lengths of $\overline{AD}$ and $\overline{BF}$. However, we are not interested in capturing the absolute size of the tiling polygon, merely its shape up to similarity. We can factor out the dependence on scale by fixing $BF = 1$ and keeping just a single parameter: $v_0 \equiv AD$. Figure 4.5 shows tilings of type IH16 that can result from different values of this single parameter.

4.4 Data structures and algorithms for IH

Using the parameterization given in the previous Section, I have developed a computer representation of isohedral tilings. The representation is expressed as a self-contained C++ library. I have written a number of large applications and small utilities that use this library to generate tilings, edit them, render them in various ways, and extract information from them.

At the top level, the library provides two classes: `IsohedralTemplate`, an abstraction of an isohedral tiling type, and `IsohedralTile`, an abstraction of a specific prototile. The template contains information about a tiling type in general, information that doesn't change from instance to instance. The prototile refers to a template and contains the information needed to determine the locations, shapes, and colours of tiles. I describe each of these components in detail, and then show how they can be used to support efficient editing and viewing.

```
    template IH16 {
1       topology 3^6
2       symbol [a+b+c+c-b-a-;a-c+b+]
3       colouring (1 2 3) (1 2 3) (1 2 3)
4       aspects 3
5       rules
6           aspect 2 1
7           aspect 3 6
8           translate T1 1,5
9           translate T2 1,3
    }
```

Figure 4.6 The tiling type information stored for IH16

4.4.1 Isohedral templates

The templates are computed once ahead of time, and stored in a master file (`isohedral.ih`) designed to be computer-readable. This file has been publicly available [92] on the World Wide Web for about two years, and has been extensively debugged in that time. When the tile library is initialized, the file is parsed and an `IsohedralTemplate` instance is created to hold each template. Other programmers using `isohedral.ih` have written software to generate code directly from the file, potentially leading to better performance, at the expense of the ease of debugging that was necessary as the templates were first being developed.

The template file contains one record for each isohedral type. A sample of such a record appears in Figure 4.6. It reproduces some of the information tabulated by Grünbaum and Shephard, such as the topological type (line 1), the incidence symbol (line 2), and the number of aspects (line 4). It also gives a default colouring (line 3). The remaining information, the `rules` section (lines 5–9), is symbolic description of how to compute the tiling's translation vectors and transform matrices for its aspects. We execute the symbolic description and cache the resulting transforms in an `IsohedralTile` to permit efficient rendering of tilings. In what follows, I provide more details about the `colouring` field and `rules` section.

The `colouring` field provides a default rule for assigning colours to tiles (colourings of tilings are described in Section 2.5). An `IsohedralTile` may override this default with its own colour-

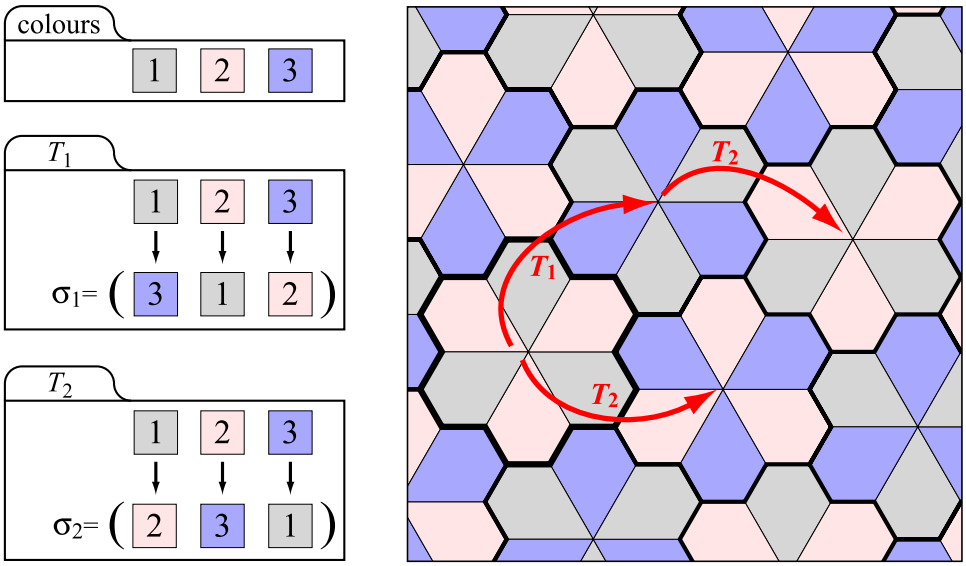


Figure 4.7 A demonstration of how the colouring information in the isohedral template (for IH21 in this case) is used to apply colours to tiles. The translational units (each containing six aspects) are outlined in bold. There are three symbolic colours, $\{1, 2, 3\}$, and they are associated respectively with gray, pink, and blue. On the left, the permutations for the two translation vectors are indicated by showing with arrows the mapping from original to permuted colours; the permutation's textual desciption can be read off of the bottom row of this mapping. On the right, the permutations are applied when moving between translational units. The colouring for this tiling can be read from the diagram as `colouring 3 (1 2 1 2 1 2) (3 1 2) (2 3 1).`

ing. Here we follow Escher's lead and aim to provide perfect colourings. Recall that in a perfect colouring, every symmetry of the tiling is a permutation of the set of colours.

The actions of all the symmetries can be summarized by giving the permutations associated with the two translation vectors of the tiling and a default assignment of colours to the aspects in a single translational unit. Successive translations will permute this default assignment appropriately. The colouring field in the template gives, in order, the number of colours, the assignment of colours to aspects in a translational unit, and the permutations of the assignment associated with the two translation vectors. A permutation ρ of the numbers $\{1, \ldots, n\}$ is very simply represented as a sequence $(s_1 \ldots s_n)$, with $\rho(k) = s_k$.

In particular, consider a tiling with translation vectors $\overrightarrow{T_1}$ and $\overrightarrow{T_2}$ and their associated colour

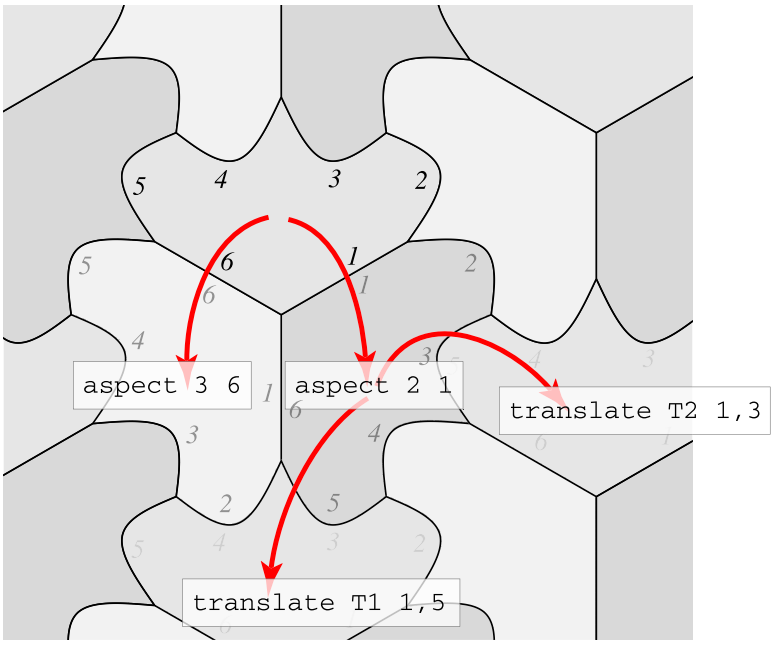


Figure 4.8 A visualization of how aspect transforms and translation vectors are computed for IH16, using the information in the `rules` section of the isohedral template (see Figure 4.6). In the order that they are referenced in the template, the aspects are coloured blue, pink, and gray. The edges are numbered as they are given in the incidence symbol. Each red arrow represents a single hop, a rigid motion that brings a tile into coincidence with one of its neighbours. The end of every sequence of hops is labeled with the corresponding rule from the template for IH16 (see Figure 4.6).

permutations ρ_1 and ρ_2. Let the tiling have n aspects, with the default colours in a translational unit given as $(c_1, \ldots, c_n)$. Then aspect k in the translational unit located at $a\overrightarrow{T_1} + b\overrightarrow{T_2}$ will have the colour $\rho_2^b(\rho_1^a(c_k))$. This encoding can in fact express a superset of the perfect colourings, but it is easy to check empirically whether a given colouring is perfect.

The `rules` section gives a collection of rules that, when applied to a tiling polygon, yield rigid motions (in the form of transformation matrices) for all the aspects of a translational unit, as well as for the two translation vectors. These transforms cannot be computed ahead of time, as they depend on the tiling polygon. We speed up the drawing of the tiling by storing these transform matrices in the `IsohedralTile` instance, and recomputing them only when the tiling vertices move.

Every tile in an isohedral tiling is surrounded in a consistent way by its neighbours, and so for

every tiling edge there is a well-defined rigid motion that carries the tile on one side of that edge to the tile on the other side. The motion will either be a half-turn around the edge's center (in the case of an **S** edge), a reflection across the edge (in the case of an **I** edge), or a translation (in the cases of **J** and **U** edges). The kind of transform that applies can be determined from the tiling type's incidence symbol, and the numeric values in the transform matrix depend on the positions of the tiling vertices that delimit the edge. We call such a rigid motion a "hop" across a tile edge. In a tile with n edges, we can label the hops unambiguously as $H_1, \ldots, H_n$. Each `rule` encodes a sequence of hops that, when chained together, transform a tile to a new aspect or to the same aspect in a neighbouring translational unit.

Aspect 1 is always given the identity matrix as its transform, and the other aspect transforms are computed from it. In the example, the first rule (line 6) says that the transform for creating aspect 2 from the first aspect is the hop across edge 1 of the first aspect — that is, a reflection across the first edge, labelled `a+`, in the incidence symbol. We will store H_1 as the aspect's transform matrix. Similarly, the second rule (line 7) says that the transform for creating aspect 3 from the first aspect is H_6, a reflection about the edge labelled `a-`.

The two translation vectors are specified in the same way. Here, we can obtain translation vector $\overrightarrow{T_1}$ in two hops, first from the first aspect across edge 1 into some neighbouring tile, and from *that* tile across edge 5. The resulting transform matrix would be $H_1 H_5$. Note, however, that this matrix does not necessarily represent a translation, and so we cannot just take $\overrightarrow{T_1}$ to be the translational component of that matrix. The problem is that the matrix may contain internal symmetries of the tile shape, which were accumulated when composing the hops together. Fortunately, we can still extract the translation in a simple way as the vector joining the centroids of the transformed and untransformed tiling vertices. This calculation works because the centroid is independent of internal tile symmetries, operations that merely permute the vertices.

In general, a `rule` may specify any number of hops to get from the first aspect to another aspect or a translation. Each step in the `rule` names an edge of the tile, and the transform is computed by composing together the associated hops.

One piece of per-tiling-type information missing from the template file is the set of tiling vertex parameterizations. The parameterizations are more easily described in code than in a table-driven format, and are embedded in the source code, each as a C++ class. A Python file `params.py` that

```
def ih16_params( v0 ):
    m = 0.5 / math.sqrt( 3.0 )
    T1 = match( Point( 0.5, v0 ), Point( 1, 0 ) )
    T2 = match( Point( 0, 0 ), Point( 0.5, v0 ) )

    return (
        Point( 0.5, -m ),
        Point( 1, 0 ),
        T1 * Point( 0.5, m ),
        Point( 0.5, v0 ),
        T2 * Point( 0.5, m ),
        Point( 0, 0 ) )
```

Figure 4.9 Sample Python code implementing the tiling vertex parameterization for IH16. When called with a single real parameter `v0`, the function returns a tiling polygon. The function `match` takes two points as arguments and returns a direct rigid motion that maps the unit interval onto the line segment given by the two points.

implements the parameterizations is available on the World Wide Web along with `isohedral.ih`. An example from `params.py` is given in Figure 4.9.

This representation of isohedral tilings suffers from a flaw related to degenerate edges in the tiling polygon. If two consecutive tiling vertices are made to coincide, then the hop across their shared edge is undefined, and any `rules` that use the degenerate edge give invalid transforms. In a purely mathematical treatment of the subject there is no problem, because there is no such thing as a degenerate edge in the tiling polygon. As two adjacent tiling vertices merge, they fuse into a single vertex and the tiling as a whole slips into a different (but related) isohedral type, as shown in Figure 4.10. The representation given here can manipulate non-degenerate tiles without any difficulty, but it cannot handle these discontinuous transitions. We will return to the subject of these discontinuities in Section 5.4. An interesting extension to the library would be to classify these isohedral types that are related through degeneracy, and transparently slip from type to type as tiling vertices move through each other.

Note also that in the tiling vertex parameterizations given, some adjacent tiling vertices simply cannot be brought into coincidence. These correspond to the edges of the tiling polygon that are assumed to have unit length. Any enhancement allowing degenerate edges would have to provide

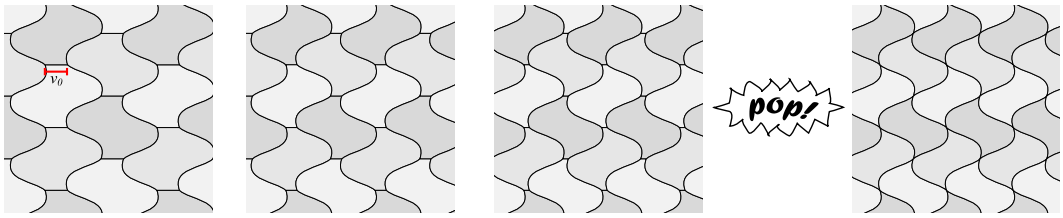


Figure 4.10 An example of how a degenerate tile edge leads to a related tiling of a different isohedral type. As parameter v_0 goes to zero in this IH9 tiling, the tiles deform continuously. But at the instant that v_0 becomes zero, pairs of tiling vertices fuse and the tiling passes through a topological discontinuity to type IH59.

alternate parameterizations in which any edge of the tiling polygon not otherwise constrained could become degenerate.

4.4.2 Isohedral prototiles

All the information related to a specific isohedral prototile is stored in the `IsohedralTile` class. A great deal of data is stored in every `IsohedralTile`:

- **Geometry information** includes the parameters for the tiling vertex parameterization, a cached tiling polygon, and the cached aspect transforms and translation vectors derived from the `rules` section of the `IsohedralTemplate`.

- **Shape information** contains polygonal paths that make up the non-redundant portion of the tile's outline (called the "fundamental edge shapes"). The shape information also includes a cached copy of the tile's outline for fast drawing.

- **Colouring information** contains a colouring (like the one that appears in `isohedral.ih`) and actual RGB triples for each symbolic colour.

A callback mechanism ensures that when part of the tile's description changes (for example, when a vertex parameter is adjusted), all cached information that depends on it is automatically updated.

The `IsohedralTile` class is designed to be extensible, flexibly storing additional information supplied by client code. One example of such an extension is a prototype system for associating vector-based artwork with the tile. The artwork is made up of a set of "markings," shapes that have familiar properties such as line width, line colour, and fill colour.

An instance of `IsohedralTile` can be written out to a "tile file," an XML document that contains the non-cached information in the tile object. Each extension is also given the opportunity to serialize itself into the tile file.

Each fundamental edge shape is an array of points representing a path starting at $(0, 0)$ and ending at $(1, 0)$. By default, the points are interpreted as a sequence of line segments, but to increase the aesthetic appeal of our tilings we have implemented the ability to treat them as control points for a subdivision curve. As a further enhancement, each control point has an associated weight. The higher the weight, the more subdivision steps will go by before that point is averaged with its neighbours. In effect, the weight controls the sharpness of the curve near the control point, with maximum weight yielding a sharp corner that interpolates the control point. (This approach is a curve-based analogue of the "hybrid subdivision" technique of DeRose *et al.* [35].) The subdivision weights are not built in to `IsohedralTile`, but implemented using the extension mechanism described above.

The shape information in the prototile contains a hierarchical model of rigid motions whose leaves are the fundamental edge shapes. The model makes multiple references to fundamental edges to express the redundancy inherent in the tile's outline. To rebuild the tile shape, we apply the tiling vertex parameterization to obtain the positions of the tiling vertices and use the hierarchical model to construct edge shapes between them.

There are at most three levels in the hierarchical model between a fundamental edge shape and a point on the outline of the tile. The first level takes into account the symmetries of **U** and **S** edges. Half of the **U** or **S** edge comes directly from the fundamental edge. The other half is derived from the first half as needed through rotation or reflection. **J** edges are passed unmodified through this level, and since **I** edges are immutable, all tiles share a single system-wide copy of an **I** edge.

At the next level up, we recognize that edges with different names in the incidence symbol may still have related shapes. In IH16, for example, the edge named `b+` is adjacent to `c+`, forcing the two edge shapes to be congruent. In this case, the two edges share the same shape passed up from

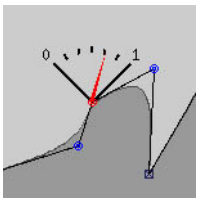
the level below.

Finally, the topmost level maps the unit interval to an edge of the tiling polygon; this mapping will move an edge shape from its normalized coordinate system into a portion of the tile's outline. At this level, all edges with the same label in the incidence symbol share a lower-level shape object.

4.4.3 Interactive viewing and editing

Using the library described above, we have implemented a number of programs that read, edit, and write isohedral tilings. For simple editing of tilings, the most important of these programs is **Tactile**, an interactive editor and viewer for tile files. Tactile is highly responsive, running at interactive rates on an off-the-shelf Linux system with no graphics acceleration. A screenshot of **Tactile** is given in Figure 4.11. Because of the deep sharing of information in the tile representation, when a part of the tile is edited, the system provides immediate feedback by showing all parts of the tile (and tiling) that are affected by the change.

When subdivision is enabled, we provide a novel gauge-based interface for editing weights on control points. The gauge pops up at the vertex location and is set with a radial motion. Setting weights integrates very comfortably with the general process of editing the vertices.



The viewer portion of the interactive system relies on an algorithm for filling a region of the plane with copies of the prototile. Given a tile shape in its local coordinate system and a viewing region, we need to find the set of rigid motions that replicate the tile to cover the region.

To find these motions, we compute the coordinates of the viewing region's corners in the basis formed by the tiling's translation vectors. In that coordinate system, the translational units become lattice squares; we draw the translational units corresponding to the lattice squares that intersect the projection of the viewing region. For each needed translation, we iterate over the tiling's aspects, placing a tile relative to the rigid motion formed by composing the translation with the aspect's transform. This algorithm is demonstrated in Figure 4.12. The figure shows that the algorithm is only approximate — it can leave a border of the viewing region unfilled. However, it is adequate for the purposes of the work presented here. When it is vital that the viewing region be completely

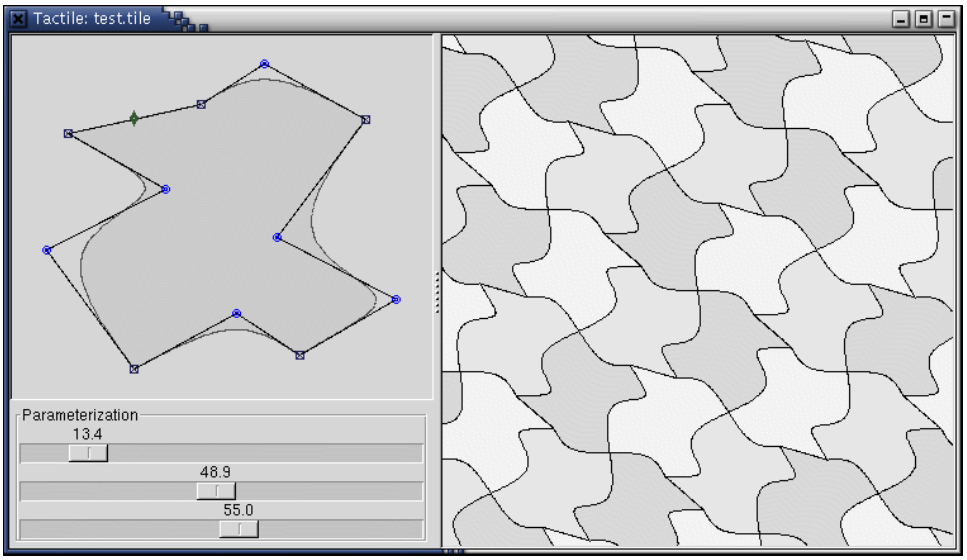


Figure 4.11 A screen shot from Tactile, the interactive viewer and editor for isohedral tilings.

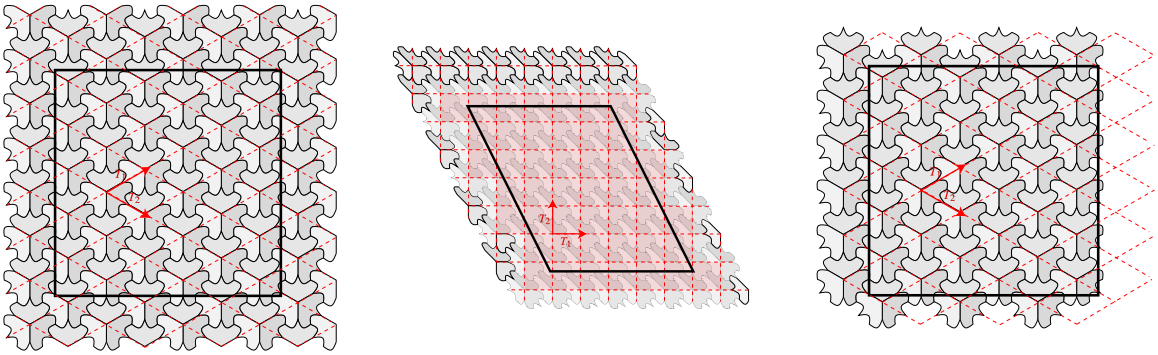


Figure 4.12 The replication algorithm for periodic Euclidean tilings. The left image shows the tiling to be replicated, with a superimposed black square representing the desired viewing region. The red parallelograms delineate translational units of the tiling, based on vectors T_1 and T_2. In the middle image, the whole diagram is shown in a coordinate system where T_1 and T_2 are an orthonormal basis. In this coordinate system, translational units are lattice squares, and it is easy to choose the squares that overlap the viewing region. The chosen units are drawn in the untransformed image on the right. Note that the algorithm is imperfect (it leaves part of the viewing region unfilled) but it is suitable for the purposes of interactive design.

filled with tiles, a simple solution is to inflate the region into a larger rectangle, and apply the filling algorithm on it instead. The filling rectangle can be made large enough that any unfilled part of its border will still lie outside the viewing region.

As was mentioned above, there is a prototype implementation of vector-based artwork for tiles, but it is rarely used because of the engineering work that would be needed to create an interface for designing and editing the artwork effectively. Doing so would be tantamount to re-implementing great drawing software like Adobe Illustrator. One future solution would be to implement a marking system as an Illustrator plug-in that takes advantage of their easy-to-use interface and exports artwork compatible with tile files.

In the meantime, I make much more frequent use of image-based tile artwork. I have implemented an image-based tiling renderer based on **libart**, a free image manipulation library that provides a sophisticated imaging model [100]. The renderer takes a tile file and a set of images (one for each colour in the tiling's colouring) to serve as backdrops. For each tile in a region, it starts with the image backdrop for that tile's colour, optionally applies a semi-transparent wash of the tile colour, rasterizes any markings that are present, draws the tile's outline, and transforms the composited tile into its position in the final rendering. Note that in this model, the user can still incorporate vector-based artwork from an external tool by rasterizing the artwork and feeding the resulting image to the renderer. This approach is not ideal, but it produces satisfactory renderings in most cases (see, for example, Figures 4.15(d) and 4.26(b)).

4.5 Escherization

Our goal in this chapter is to understand Escher's tesselations with the ultimate aim of creating new imagery in the same style. Based on the tools described so far, this goal is already possible. We can use **Tactile** just as we would use commercial products like TesselMania! or Tess to design ornamental tilings and render them with decorations. These tools are a boon to artists because the computer can absorb all the tedium of replication and the difficulty of accounting for the constraints on a tile's shape. The artist is free to explore and to develop an intuition for the aesthetic ranges of different tiling types.

This design process might be called a "forward tiling process": we start from a simple shape

that is known to tile and evolve it until the outline sparks the imagination, evoking some real-world form. At that point we can tweak the shape to better convey the form, and paint the tiles. The Escher-like tesselations by contemporary artists are for the most part developed using a forward process. One must imagine that Escher himself worked this way, though he was certainly gifted with an uncommon intuition. His son recalls Escher's singular ability to tease animal forms and faces from the random patterns of clouds, wood grain, and random swirls of paint [49, Page 7].

The suggestive use of the word "forward" above implies that there ought to be an "inverse" tiling design problem. Whereas in the forward process we start with a mathematically simple shape and evolve it to create an artistic result, here we would start with a desired form and "devolve" it, imposing the mathematical constraints necessary to make that shape tile. The constraints will force some deformation upon the original shape, but with luck the change will be small enough that the result will still be recognizable.

We formalize the inverse tiling design problem as the "Escherization problem":

> **Problem** ("ESCHERIZATION"): Given a closed plane figure S (the "goal shape"), find a new closed figure T such that:
>
> 1. T is as close as possible to S; and
> 2. T admits a monohedral tiling of the plane.

As Section 2.3 points out, little is understood about monohedral tilings in general. To make Escherization tractable, we immediately retreat from the fully general problem statement and restrict our attention to IH (later, we will consider Escherization over some other families of tilings). The previous sections have shown how to parameterize the space of isohedral tilings and implement that parameterization as a software library. We also know (as was mentioned at the start of Section 4.3) that the isohedral tilings are a good match for the sorts of tesselations Escher created.

The nature of a solution to the Escherization problem hinges on the interpretation of the word "find." We may imagine a space $\mathcal{S}$ formed by the set of all possible shapes in the plane, together with some sort of metric that measures the "closeness" between two shapes. The prototiles of isohedral tilings form a subspace $\mathcal{T}$ of shape space. A given goal shape S is a point somewhere in $\mathcal{S}$. Optimistically, an analytic solution to Escherization would project shape space onto its tiling subspace, moving S to a closest point $T \in \mathcal{T}$.

Though appealing to the mathematical aesthetic, such an approach is infeasible. However, the mental model provided by $\mathcal{S}$ and $\mathcal{T}$, and the idea of closeness as a metric, provide the intuition needed to formulate an attack on Escherization. We envision starting from some known prototile and trying to move through $\mathcal{T}$ in a way that brings us ever closer to S. In other words, our proposed solution will be based upon continuous optimization, a common technique in computer science for solving problems that are not easily invertible. Using optimization, we may never reach a prototile that is globally optimal with respect to S, but perhaps we can get close enough to satisfy the artistic intent of S.

A continuous optimization problem consists of a *configuration space* and an *evaluation function* that maps configurations to real numbers. The goal is to find the configuration that minimizes (or equivalently, maximizes) the value of the evaluation function. In the optimization procedure for Escherization, the configuration space is $\mathbb{R}^n$; the parameters that describe the shape of an isohedral prototile can readily be expressed as a tuple of real numbers, as will be shown below. Other optimizers (see, for example, the work of Agrawala on route maps [4]) maintain a more abstract configuration space and rely on a user-defined function to suggest perturbations to configurations.

The evaluation function in Escherization is precisely the closeness metric in shape space — by decreasing the value of that metric (for some formal definition of closeness), we find tiles that are more like the goal shape.

The remainder of this section formalizes the Escherization algorithm by providing an efficient closeness metric and a framework in which the continuous optimization takes place. The implementation is able to find reasonable-looking tiles for many real-world shapes (see, for example, the "Escherized" version of Escher's own self-portrait, shown in Figure 4.15(a)). Subsequent sections will then build upon the ideas presented here by showing other forms of Escherization.

4.5.1 The shape metric

The Escherization problem raises the difficult question of how to compare two shapes. An answer should be in the form of a metric that would take two outlines and return a nonnegative real number; zero would mean that the outlines are identical, and higher positive values would denote shapes that are increasingly dissimilar. We would also like the metric to be insensitive to rigid motions or

uniform scaling of either of the shapes.

Fortunately, such metrics have been developed by computer vision researchers. We use the metric devised by Arkin *et al.* for comparing polygons [6]. Their metric represents the input polygons as *turning functions*, functions that map fraction of arc length in a polygon to the angle of the polygon at that point (with respect to the positive x axis). Turning functions are naturally translation- and scale-independent. Vertical and horizontal translation of a turning function correspond respectively to rotation of the polygon and movement of the point where the measurement of arc length begins. They show that the minimal L^2 distance between all possible translations of the two turning functions satisfies the mathematical definition of a metric and corresponds to an intuitive measure of closeness. It turns out that this minimum is achieved at one of a relatively small number of translations of the turning functions, allowing for an efficient algorithm that searches these configurations for that minimum. They provide an implementation that compares two polygons with m and n vertices in time $\mathrm{O}(mn \log n)$.

This algorithm has the drawback that detail is assumed to be directly proportional to arc length. In other words, two pieces of a polygon's boundary with the same length carry the same amount of detail even if one piece packs that detail into a much smaller space. It helps to keep this fact in mind and aim for a consistent level of detail when creating a goal shape to be Escherized. I have also had some success running a low-pass filter over the goal shape to smooth out local areas of high detail.

I use the polygon comparison metric for both polygons and subdivision curves. A subdivision curve is simply approximated by a polygon with a large number of vertices.

4.5.2 Optimizing over the space of tilings

Armed with a set of tilings, parameterizations over those tilings, and a good shape metric, we are now ready to address the problem of building an optimizer that can search over the space of those tilings to find an instance whose prototile is close to the goal shape.

Our optimizer is based on simulated annealing. It works roughly as follows:

```
function FINDOPTIMALTILING(GOALSHAPE, FAMILIES):
    INSTANCES ← CREATEINSTANCES(FAMILIES)
    while ||INSTANCES|| > 1 do
        for each i in INSTANCES do
            ANNEAL(i, GOALSHAPE)
        end for
        INSTANCES ← PRUNE(INSTANCES)
    end while
    return CONTENTS(INSTANCES)
end function
```

The optimizer takes as input a goal shape and a set of isohedral families in which to search for an optimal tiling. The optimizer begins by creating a set of multiple `IsohedralTile` instances for each isohedral type. The "default" tiling for each isohedral type is chosen to be the type's underlying Laves tiling. The Laves tiling can always be expressed within the parameterization space of any isohedral type of the same topology. So each new instance starts with the Laves tiling and applies a small random perturbation to its shape.

The optimizer then calls a re-entrant simulated annealing procedure to improve each one of these instances. (This ANNEAL() procedure is discussed in more detail below.) After each of the instances has been optimized to some degree, the instances are evaluated according to the shape metric, and the worst ones are removed. The annealing is continued on the remaining instances. This iterative process of alternately pruning the search space and then improving the remaining instances is repeated until just a single `IsohedralTile` is left. This tiling is returned as the output of the Escherization algorithm.

The set of tiling types to be passed to the optimizer is at the user's discretion, but some guidelines can be used to drastically reduce the number of types that need to be checked. As has been mentioned before, some isohedral types "subsume" others, in the sense that a tiling type with a symmetric prototile can be seen as a special case of a related tiling type with no prototile symmetry. The more symmetric type, being more constrained, can get no closer to the goal shape than its asymmetric counterpart, and so it need not be included in the optimization. The best possible solution can be found by optimizing only over the 28 types with asymmetric prototiles (correspond-

ing to the 28 Heesh types). Once a solution is found, it is always possible to re-run the optimizer with the symmetric children of the solution's tiling type, in search of a solution with more prototile symmetry.

The annealer is a re-entrant procedure, which works roughly like this:

```
procedure ANNEAL(TILING, GOALSHAPE):
    loop
        while T > T_min do
            OPTIMIZETILING(TILING, GOALSHAPE, T)
            T ← REDUCE(T)
            suspend
        end while
        SMOOTHEDGESHAPES(TILING)
        SPLITEDGESHAPES(TILING)
        (T, T_min) ← UPDATESCHEDULE(T, T_min)
    end loop
end procedure
```

The annealer takes a given tiling instance and a goal shape as input. It acts like a coroutine, periodically suspending itself while maintaining its state, so that it can be resumed seamlessly later. FINDOPTIMALTILING runs each annealer for a few hundred iterations at a time, so that they can be compared against each other to track progress. The ANNEAL procedure takes a number of cooling steps, reducing the "temperature" at each step. Within this inner loop, it makes a call to a procedure called OPTIMIZETILING(). This procedure implements the continuous multidimensional simulated annealing algorithm described by Press *et al.* [120, Section 10.9]. Their algorithm is based on a fuzzy version of the "downhill simplex method," where a simplex of proposed solutions evolves towards a global minimum by pushing the worst solution through the hyperplane formed by the remaining ones.[2] The procedure attempts to improve all of the parameters of the tiling, comprised of the parameterization of the tiling vertices and the positions of the shape vertices. The procedure always accepts a downhill step (one that brings the tile shape closer to the goal shape) and sometimes accepts an uphill step, with probability depending on the temperature T. Once the temperature has

[2]This technique should not be confused with the simplex method of linear programming.

cooled to some minimum temperature T_{min}, the inner loop terminates. At this stage, the algorithm runs through the vertices of the tile and removes any vertices that are nearly collinear with their neighbors; these vertices are not being used by the optimizer to improve the tile shape, and are needlessly slowing down the optimization. Next, the edges of the tile are subdivided, reintroducing vertices where detail might get added as the optimization continues. (Note that during this process, the dimensionality of the annealing problem may change, and we must ensure that OPTIMIZETILING can cope with the change.) The cycle of smoothing and subdividing edges is highly effective in keeping degrees of freedom only where they are needed. Finally, the cooling schedule restarts, generally with slightly lower values of T and T_{min}.

One additional part of the optimization, which is not shown in the pseudocode and which is optional, is to automatically convert the vertices of the tiles into control points for subdivison curves after a certain stage in the optimization. The annealer can then incorporate the weights on the subdivision control points as additional degrees of freedom.

The use of simulated annealing is subject to the usual practicalities. First, the success of the optimization for a single instance of a single tiling type depends to some extent on the initial shape of the tiling polygon and the initial positions of the shape vertices. I therefore generally start with multiple, randomly perturbed instances for each tiling type. An interesting alternative would be to seed the initial tiles with segments from the goal shape's outline. As with any simulating annealing algorithm, the choice of cooling schedule can also make a difference. I use a very simple approach where the temperature T is multiplied by a factor of ϕ after every N iterations, with $T = 0.1$, $N = 250$, and $\phi = 0.9$ to start. When the temperature reaches 5% of its initial value ($T_{\min} = 0.05T$), the optimization resets, lowering the starting and minimum temperatures by a factor of 0.6, increasing the number of iterations N by a factor of 1.2, and reducing the temperature multiplier ϕ by a factor of 0.1. I did not spend a lot of time "optimizing" this cooling schedule, so other reasonable choices would probably work equally well or better.

4.5.3 Results

Figure 4.13 shows snapshots from two sample runs of the Escherizer. The goal shape in the first run is a simple test polygon, part of a series used to verify and tune the optimizer. The second goal

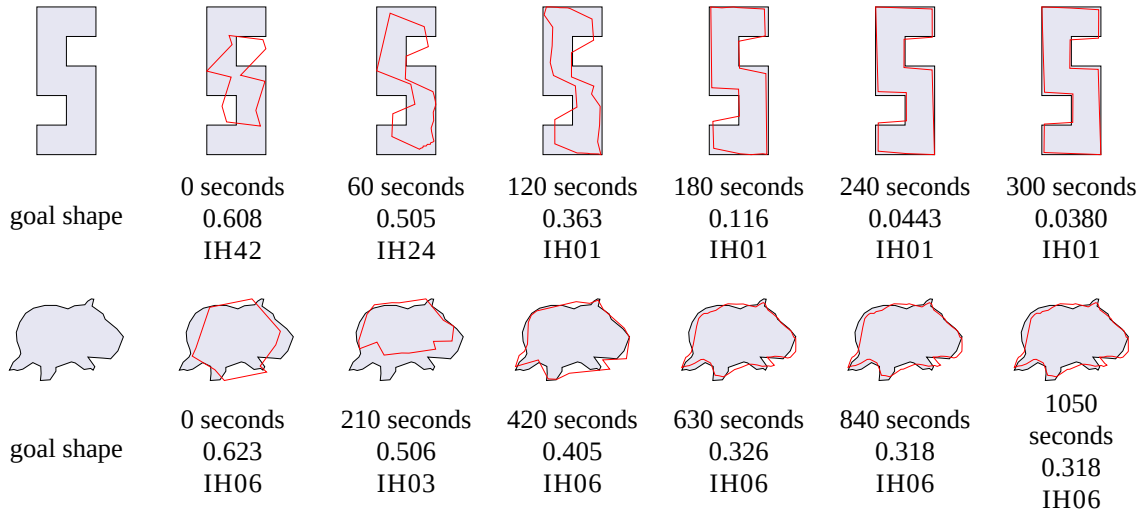


Figure 4.13 Timelines for two sample Escherization runs. Each step shows the current best tile in the system (in red) overlaid on the goal shape. The caption indicates the elapsed time, the score for that tile, and its isohedral type.

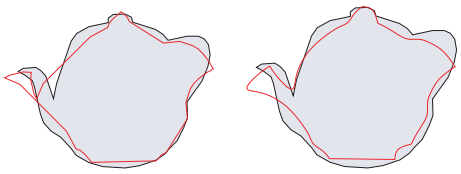


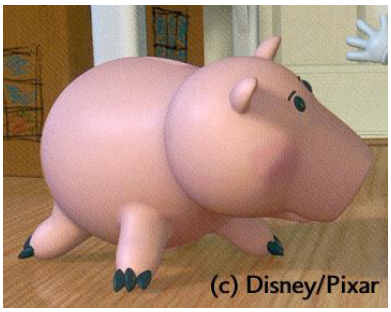
Figure 4.14 A comparison between the tile returned by the optimizer and the same tile with user modifications. Note also that the second tile has subdivision enabled.

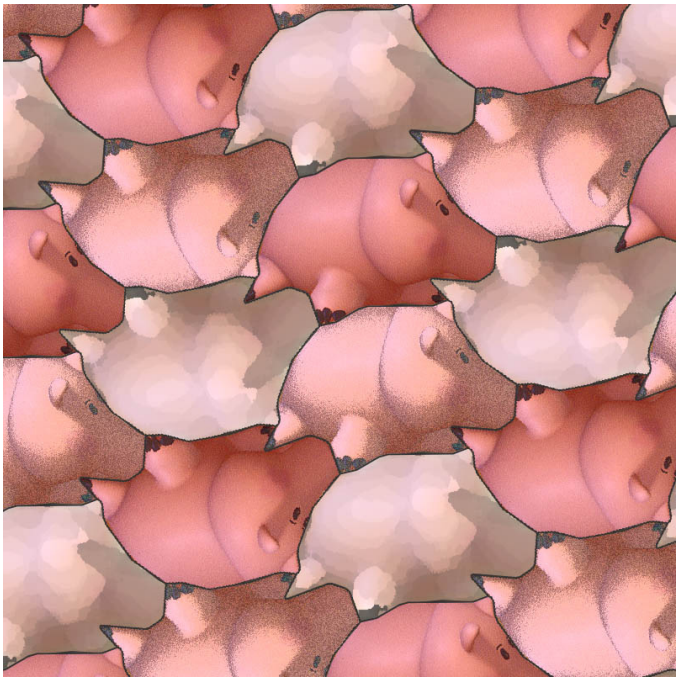
shape is a more typical real-world outline. The more complicated shape takes longer to run, and the convergence is not quite as complete (as should be expected from a real-world outline).

The optimizer does not require user intervention, but it does run interactively so that its progress may be watched. In practice, some constructive intervention is possible while watching the running Escherizer. If it is clear that the goal shape will simply not work as a tiling, the process can be interrupted. On the other hand, if one particular tiling type seems to be performing very well on the goal shape, the program can be stopped and restarted with many instances of that type, resulting in a narrower and deeper search.

Figure 4.14 shows the tile result produced by the optimizer for a teapot image, followed by the

(a) *Escher's Escher Escherized* (IH1)





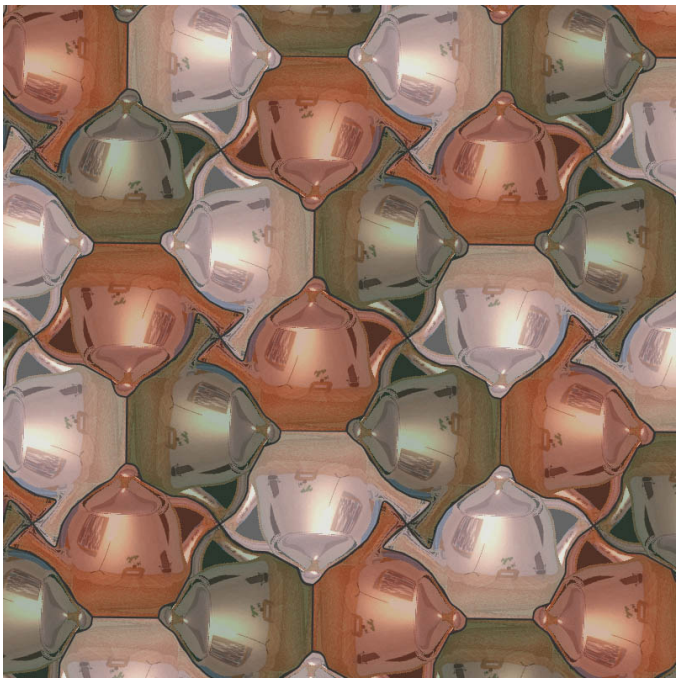
(b) *Pigs in 2-Space* (IH3)

Figure 4.15 Some examples of Escherized images and the tilings they generate. Hamm the pig appears courtesy of Disney/Pixar.

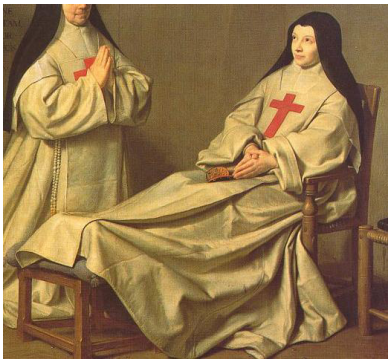
(c) *Dogs; Dogs Everywhere* (IH4)

(d) *Weiner Dog Art* (IH5)

Figure 4.15 (continued)

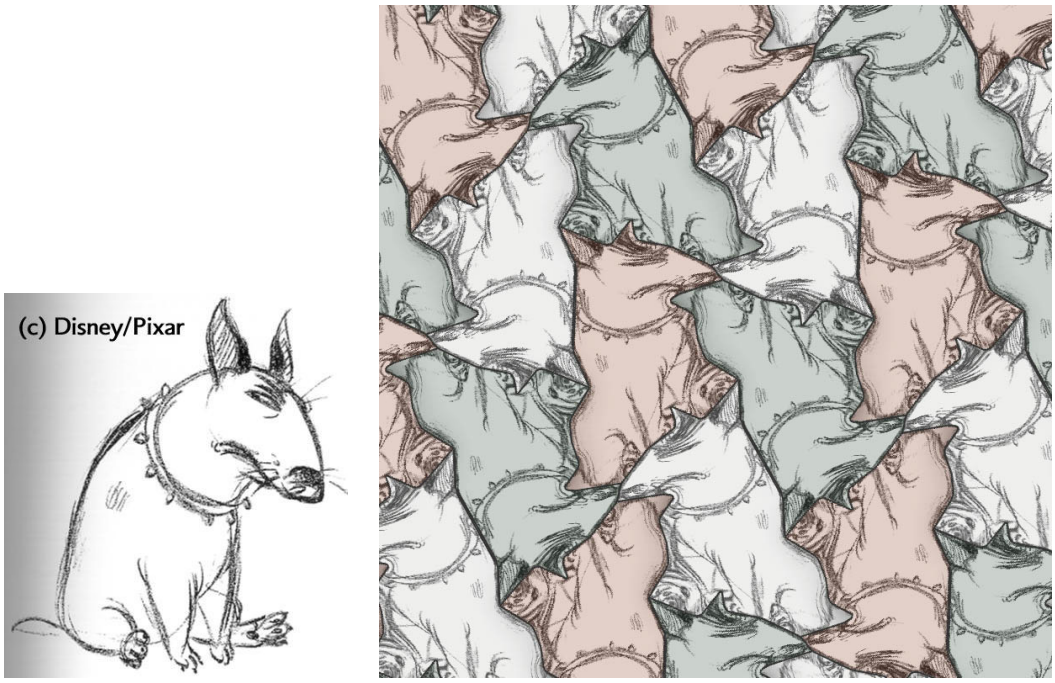


(e) *Tea-sselation* (IH28)

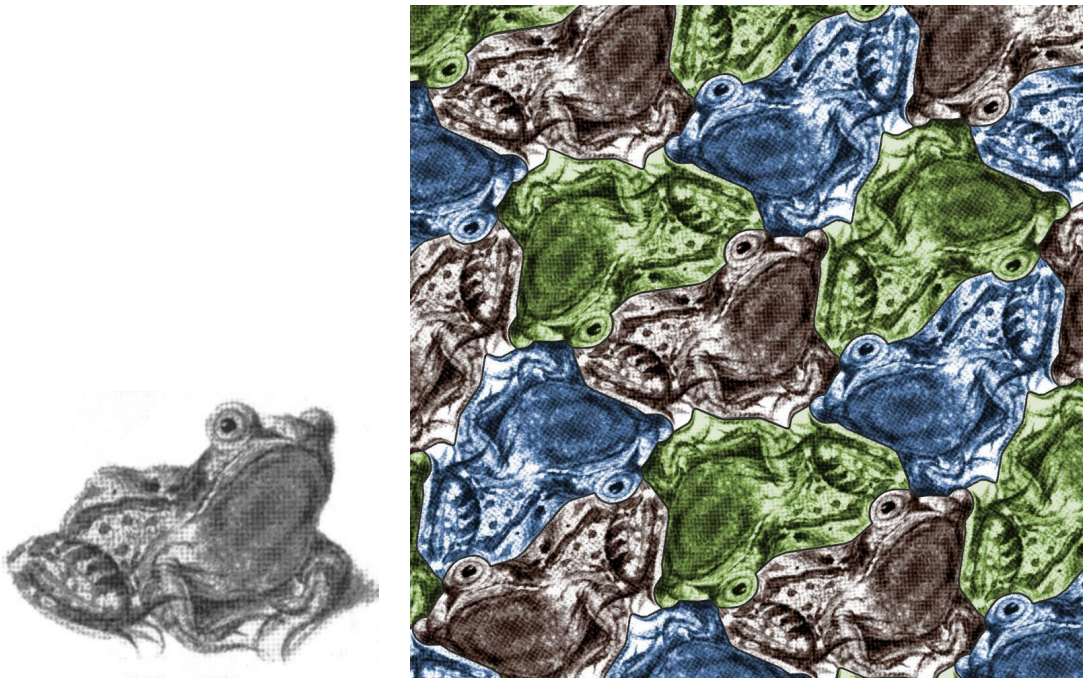


(f) *Twisted Sisters* (IH86)

Figure 4.15 (continued).

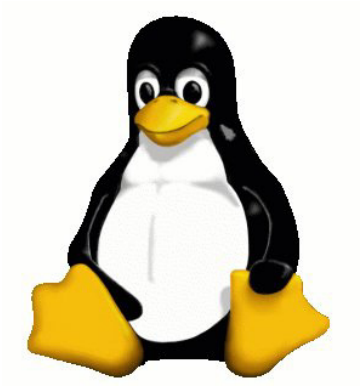


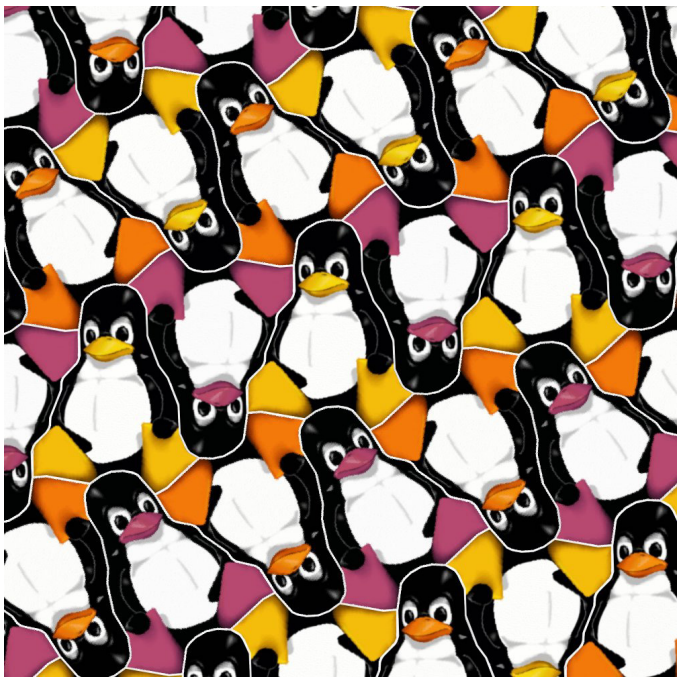
(g) *Sketchy Dogs* (IH6)



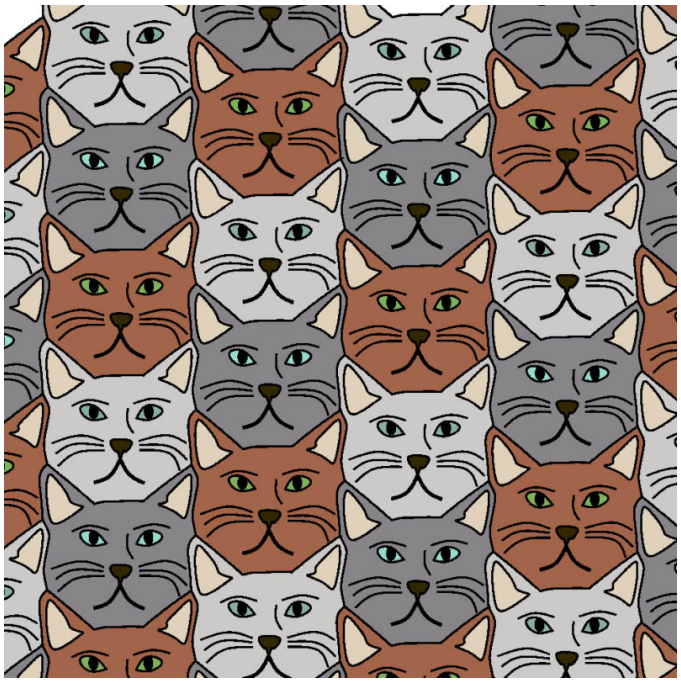
(h) *A Plague of Frogs* (IH6)

Figure 4.15 (continued). Sketchy dog appears courtesy of Disney and Pixar.





(i) *Tux-ture mapping* (IH6)



(j) *Bubbles the Cat* (IH1)

Figure 4.15 (continued). Tux the penguin appears courtesy of Larry Ewing.

tile after a small amount of hand-tweaking in the interactive editor. For the results shown here, the edits took a minute or two to perform and were fairly typical of the experience of creating tilings using Escherization. In general, hand editing of Escherized outlines should be expected and even welcomed. The success of Escherization does not lie in its ability to replace an artist with a black box tool that produces tilings from thin air. An artist will have a specific intent in mind, expressed roughly (but perhaps not precisely) with a goal shape. The Escherization algorithm achieves a "quantum leap of aesthetics" by proposing a tiling with a prototile that roughly resembles the goal shape, a feat that exceeds the grasp of most human intuition. Still, a result that is perfectly sound mathematically might lack in artistic merit. All Escherization sees of the artist's goals is the imperfect communication of those goals as a shape to approximate. As part of the process of turning the Escherized tiling into a finished drawing, it is only natural that the artist would want to start by examining the tile shapes themselves.

The Escherization workflow starts with the user defining a goal shape. Typically, this goal shape is specified by tracing a feature in an image (say, the outline of a favourite pet). An obvious enhancement would be to incorporate computer-assisted tools such as intelligent scissors to extract contours in an image [114].

The natural choice for decorating an Escherized tile is to use the interior of the goal shape in the image that was originally traced. Using the correspondence provided by the comparison metric, I do a Beier-Neely style image warp [8] to deform the interior of the goal shape in the source image into the interior of the Escherized tile shape. When the deformation is not too great, the result is an attractive tiling out of motifs that resemble the original image. When the automatically-determined correspondence produces too much distortion (which can happen when the goal shape and tile shape differ in level of detail), it can be edited by hand to create a better match. This use of image warping to decorate tiles is valuable in a somewhat sneaky way as well: it can help to cover up deficiencies in the tile shapes. The eye latches onto features of the decoration, and is willing to forgive a certain lack of believability in the shapes of tiles. Escher himself was no stranger to this technique. His deft addition of features as simple as circles for eyes could turn even the most abstract form into a whimsical creature. (In *Understanding Comics*, McCloud eloquently illustrates this point in the context of comics [110, Page 32].)

To further increase the appeal of an image-based rendering, I apply various filters and effects to

the warped tile image before replication. This post-processing step gives the artist creative control over the appearance of the final tiling. Alternatively, as was discussed in Section 4.4.3, the original image can be discarded entirely in favour of vector-based art created using a commercial drawing package. The artwork is then rasterized and re-inserted into the rendering pipeline. The user can also create multiple versions of the decorated tile, which are placed according to the tiling's colouring.

I have used the Escherization algorithm and decoration tools to produce a number of ornamental tesselations from various sources of imagery. These results can be seen in Figure 4.15. The vector-based art for *Bubbles the Cat* (j) was created using the prototype marking system. The decorations for *Weiner Dog Art* (d), on the other hand, are a combination of line art created in Adobe Illustrator and textures and colouring created in the GIMP [53]. *Pigs in 2-Space* (b) uses various artistic filters from Adobe Photoshop.

4.6 Dihedral Escherization

The Escherization algorithm described in the previous section is limited to producing monohedral tilings, and while an artist can use the system to produce a variety of designs in the style of M. C. Escher, we have only scratched the surface in terms of the complete set of tesselations he created. In addition to the monohedral tesselations that make up his collected symmetry drawings [124], we also find many tesselations with two or (less frequently) more motifs.

Although these multihedral tilings make up the minority of his drawings, they are a very important aspect of his work. Some of his most famous prints (for example, *Sky and Water*, *Verbum*, and *Metamorphosis II*) make use of one or more dihedral tilings. Furthermore, the use of multiple motifs agrees with Escher's predisposition to imbue his work with narrative structure (most clearly expressed in the *Metamorphosis* prints [49, Page 48]). A single motif, unchanging forever except for colour, is largely incapable of telling a story. With two or more motifs, suddenly there is the opportunity for contrasts, for harmony or discord, for interaction and drama. In symmetry drawing 45 [124, Page 150], better known as "Heaven and Hell," or "Angels and Devils," Escher plays with the balance that exists between good and evil, light and dark. Drawing 63 [124, Page 165], later immortalized in the print *Encounter*, depicts smiling optimists and frowning pessimists attempting to work out their differences.

Naturally, we would like to rework the Escherization algorithm to handle two goal shapes simultaneously and produce dihedral tilings as output. We can immediately formulate a revised version of the original Escherization problem:

> **Problem** ("DIHEDRAL ESCHERIZATION"): Given closed plane figures S_1 and S_2 (the "goal shapes"), find new closed figures T_1 and T_2 such that:
>
> 1. T_1 and T_2 are as close as possible to S_1 and S_2, respectively; and
> 2. T_1 and T_2 admit a dihedral tiling of the plane.

The success of the monohedral Escherization algorithm suggests that we should take a similar approach here. I structure the dihedral Escherization algorithm as a continuous optimization over some space of dihedral tilings. At every step in the optimization, the parameters will somehow be converted into a pair of tile shapes, whereupon I can use the original polygon comparison metric (twice) to see how similar these tile shapes are to the two goal shapes. Some combination of the two comparisons will act as an evaluation function for the dihedral optimization.

The majority of the work is therefore limited to the definition of an appropriate space of tilings to plug in to the optimization. In this section, I present two spaces of dihedral tilings that yield satisfactory results. The first consists of the *split isohedral tilings*, which naturally capture the sorts of two-motif tesselations Escher created. The second set of tilings is made up of generalizations of Penrose's aperiodic tile sets $P2$ (kites and darts) and $P3$ (rhombs).

4.6.1 Split isohedral Escherization

Our search for a space of useful dihedral tilings begins with Escher himself. A meticulous note-taker, he documented his exploration of two-motif systems [123, 124]. In every case, he starts with one of his monohedral systems and draws a path through the prototile to break it into two shapes. When that division is copied symmetrically to all other tiles, the result is a dihedral tiling.

Escher's splitting process was carried out consistently for every tile, yielding a dihedral tiling with two prototiles A and B. It follows that the resulting dihedral tiling will have exactly two transitivity classes, one containing all the A tiles and the other containing all the B tiles. In other

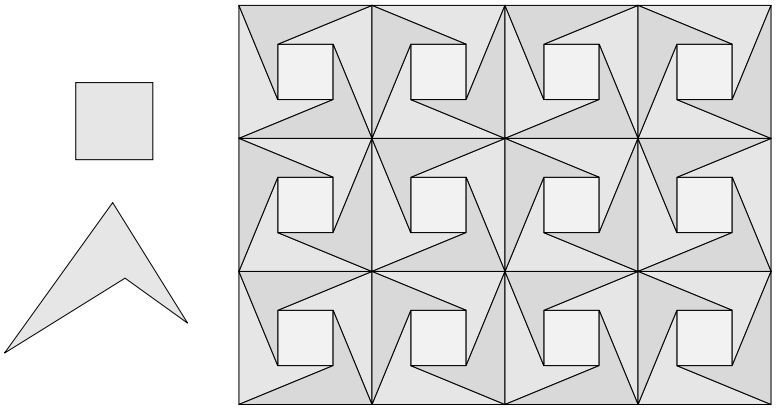


Figure 4.16 A 2-isohedral tiling with different numbers of A and B tiles. The two prototiles, a square and an irregular dart, are shown on the left. The tiling on the right has four times as many darts as squares in every translational unit.

words, Escher's two-motif systems are just 2-isohedral tilings. We can conclude that to create dihedral tesselations in the spirit of Escher, we need to parameterize at most the 2-isohedral tilings.

On the other hand, Escher focused on only a small part of all possible 2-isohedral tilings. His tilings always have equal numbers of A and B tiles (in each translational unit), whereas there exist many 2-isohedral tilings that do not (an example is given in Figure 4.16). Moreover, his base tilings and splitting paths were always chosen to give a tiling that could be coloured using only two colours. As Schattschneider points out, this decision was motivated partly by the practicality of working in the medium of printmaking, which in its simplest form produces pictures with two colours: the ink colour and paper colour. Our space of tilings should be at least broad enough to contain Escher's understanding of two-motif designs.

Delgado-Friedrichs *et al.* carry out a complete enumeration of the 2-isohedral tilings [33]. They prove a general result that every $(k+1)$-isohedral tiling can be constructed from a k-isohedral tiling through a combination of two operations: SPLIT and GLUE. The SPLIT operation corresponds with Escher's use of a splitting curve, and they show that any *fundamental* (asymmetric) prototile of a $(k+1)$-isohedral tiling can be derived from a fundamental prototile of a k-isohedral tiling through a single SPLIT. The GLUE operation erases the edge between two adjacent tiles, producing tilings with symmetric prototiles, some of which have different numbers of A and B tiles. Using SPLIT

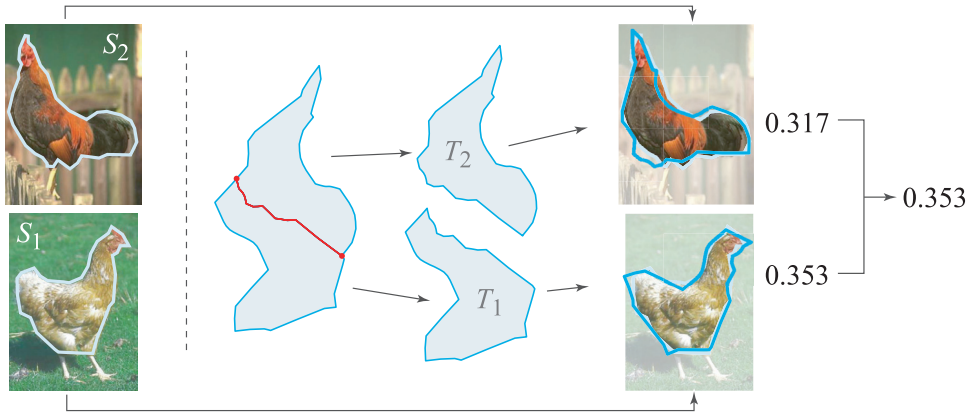


Figure 4.17 A summary of our process for split isohedral Escherization. On the left, two goal shapes S_1 and S_2 are traced from images. Next, the isohedral tile and splitting path are shown at a late stage in the optimization. The quality of this configuration is judged by breaking the tile into two shapes T_1 and T_2, which are then compared with S_1 and S_2. The optimization attempts to minimize the value at the right, the maximum of the two comparisons.

and GLUE, they show that there are over a thousand 2-isohedral tiling types, making a dihedral Escherization algorithm based on this full classification impractical. Instead, I look for an approach that exploits the work already done on the isohedral tilings.

For the purposes of dihedral Escherization, I concentrate on the *split isohedral tilings*, tilings that can be derived from an isohedral tiling through a single application of SPLIT. This family of tilings contains all of Escher's two-motif systems. It fails to distinguish those tilings that can be coloured with only two colours, though in some sense this fact is not critical because results can easily be printed in multiple colours. (Alternatively, from Escher's notes it would be possible to derive the necessary restrictions on tiling type and splitting path that would produce exactly his two-motif systems.) Later, in Section 4.6.2, we will encounter an even stronger restriction that yields a special family of dihedral Escher tilings dubbed "Heaven and Hell" patterns [38].

A split isohedral tile is represented by a class `SlicedTile`, a subclass of `IsohedralTile`. I augment the isohedral prototile with a splitting path, which is embodied by a new set of parameters: two parameters that control the path's start and end positions along the boundary of the isohedral prototile, and the degrees of freedom that control the path's shape.

The optimization process works in exactly the same way as the monohedral case, except that the evaluation function must be modified to handle two goal shapes instead of just one. The splitting path in a `SlicedTile` divides the isohedral prototile into two shapes T_1 and T_2. The evaluation function uses the shape metric defined in Section 4.5.1 to compare T_1 with S_1 and T_2 with S_2, and returns the maximum of the two comparisons as the optimization's evaluation function. By returning the maximum (a kind of L_∞ norm), we ask both tile shapes to approximate their goal shapes as closely as possible. This process is illustrated in Figure 4.17.

Let S_1' and S_2' denote reflections of S_1 and S_2. To find the best split isohedral tiling corresponding to two goal shapes S_1 and S_2, two instances of the Escherization algorithm are required: with S_1 and S_2, and with S_1 and S_2' (or S_1' and S_2). Although the shape comparison metric is insensitive to translation and rotation, it does distinguish between a polygon and its reflection. It might happen that S_1 and S_2 interact more favourably if one is reflected.[3]

Figure 4.18 shows some examples of Escherization using split isohedral tiles. One might guess that because of the need to match two goal shapes simultaneously, dihedral Escherization would have a lower success rate than monohedral Escherization. I have found that the additional degrees of freedom offered by the splitting edge more than compensate for the added complexity of the problem, and that the dihedral objective function performs comparably to or better than the monohedral one.

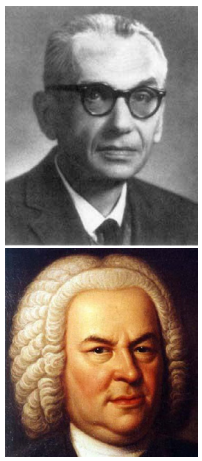
4.6.2 Heaven and Hell Escherization

Some of Escher's dihedral tesselations, such as *Heaven and Hell* [124, Page 150], have additional geometric structure that sets them apart from the rest. Not only is the tiling colourable using only two colours, but each colour is the exclusive domain of one of the classes of tiles; in Heaven and Hell, every angel is white and every devil is black. This sort of colouring is possible when every tiling vertex is surrounded by an alternating sequence of A and B tiles, or equivalently, when every A tile shares edges only with B tiles (and vice versa).

Aesthetically, such tilings are particularly effective because each transitivity class of tiles plays

[3]Note that only the relative parity matters; the flexibility of the isohedral tilings guarantees that (S_1, S_2) is equivalent to (S_1', S_2'), and that (S_1', S_2) is equivalent to (S_1, S_2').

(a) *Strange 'Tractors* (IH28)



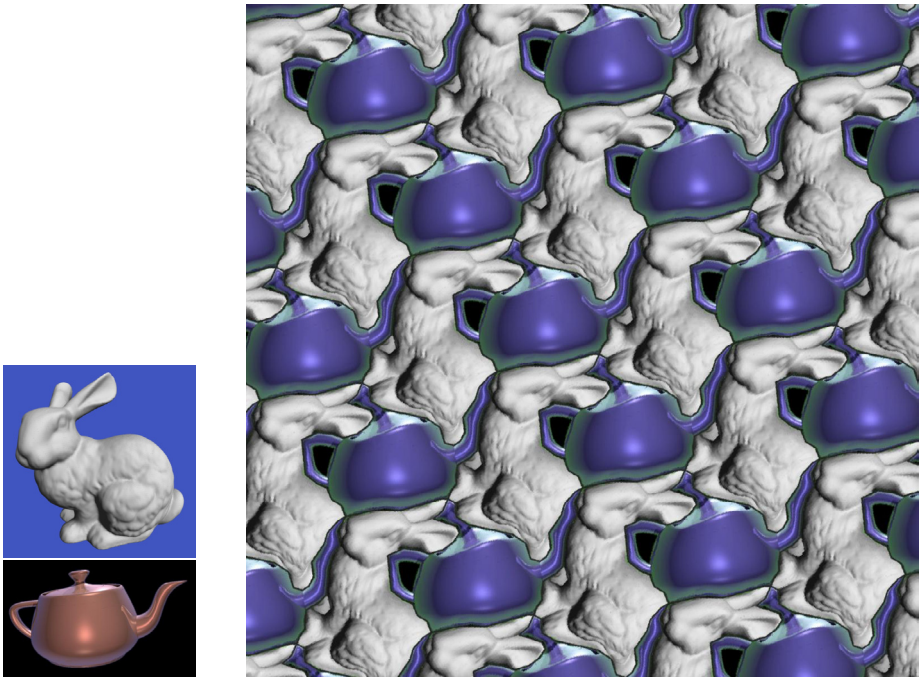
(b) *Gödel, Bach (Braided): an Eternal Escherization* (IH2)

Figure 4.18 Examples of dihedral Escherization using the split isohedral tile method.

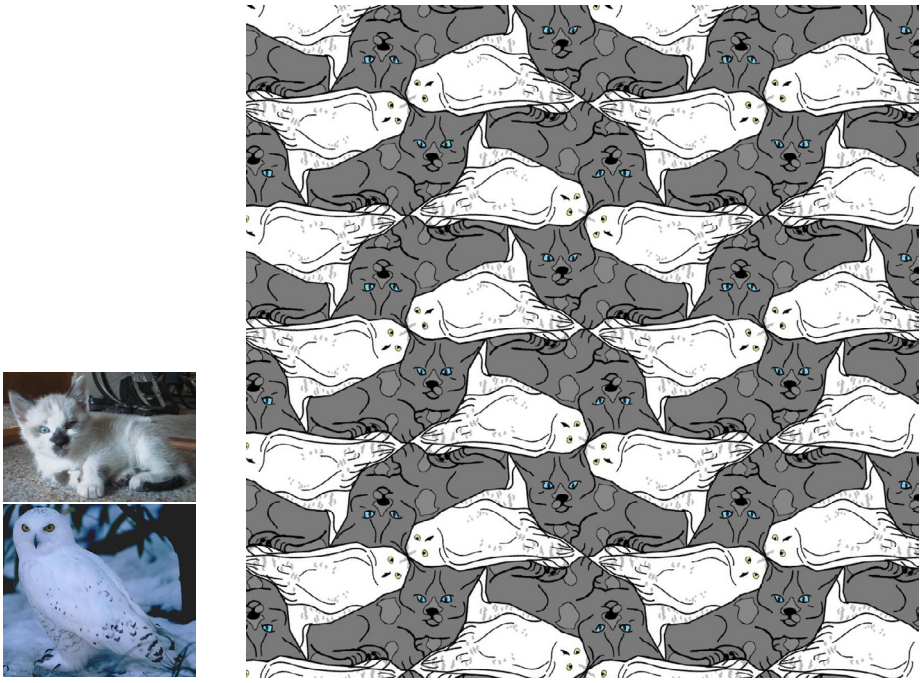
(c) *Pen/Rose Tiling* (IH1)

(d) *Rembrandt and Mrs. van Rijn* (IH1)

Figure 4.18 (continued)



(a) *The complete history of computer graphics* $(\mathrm{IH27}; 1\frac{1}{2}, 4)$



(b) *The Owl and the Pussycat* $(\mathrm{IH27}; 1\frac{1}{2}, 4)$

Figure 4.19 Examples of Heaven and Hell Escherization.

the role of ground to the other's figure: the A tiles exactly fill the negative space created by the B tiles. Moreover, the fact that the colours can be unambiguously associated with tile shapes allows them to become part of the identities of those shapes, as in the white angels and black devils of *Heaven and Hell*. Escher used this particular space of tilings to produce some of his best-known prints, several of which are mentioned at the beginning of this section.

The class of 2-isohedral tilings with this additional structure were enumerated by Dress [38], who dubbed them "Heaven and Hell" patterns. Based on an analysis using Delaney symbols, he classified the Heaven and Hell patterns into 37 distinct types. As always, we can develop an Escherization algorithm tailored to this particular classification, allowing us to construct these especially satisfying tesselations.

Twenty-nine of Dress's types can be constructed by applying the SPLIT operation to an isohedral tile and are therefore special cases of split isohedral tilings. The additional structure comes from a careful choice of locations for the endpoints of the splitting path. Analysis of Dress's classification (and examination of the diagrams in his paper) reveals that for these twenty-nine types, the path's endpoints are limited to a small number of possible locations. An endpoint will always be either one of the tiling vertices of the underlying isohedral prototile, or the midpoint of one of its tiling edges. If the isohedral prototile has n tiling vertices, we can enumerate this set of locations as $L = \{1, 1\frac{1}{2}, 2, 2\frac{1}{2}, \ldots, n, n+\frac{1}{2}\}$, where a whole number k refers to a tiling vertex and $k+\frac{1}{2}$ refers to the midpoint of the edge from k to $k+1$. The numbering of the tiling vertices can be taken from the ordering of the edges in the tiling type's incidence symbol, as given in `isohedral.ih` (see Section 4.4). Each of the 29 types based on splitting can then be given the notation $(\mathrm{IH}m; a, b)$, where $\mathrm{IH}m$ denotes one of the 93 isohedral types, and where $a, b \in L$.

I represent the Heaven and Hell tiling $(\mathrm{IH}m; a, b)$ using a slightly modified version of the class `SlicedTile`. Under this modification, the endpoints of the splitting path are not treated as degrees of freedom in the optimization but fixed according to the locations a and b. Once the two endpoints are fixed in this way, the remainder of the dihedral Escherization algorithm can be applied as is.

As has been mentioned before, tiling types can often be placed in a hierarchy, where types with asymmetric prototiles subsume their more symmetric children. Dress discusses this ordering, and his diagram shows the 37 Heaven and Hell types laid out according to their heirarchy. His explicit ordering makes it easy to recognize that of the 29 types based on splitting, we need only optimize

over 12, as the remaining 17 are subsumed under them by prototile symmetry. Using the notation given above, the 12 types are as follows:

$$(\mathrm{IH1}; 1, 4),\ (\mathrm{IH2}; 2, 5),\ (\mathrm{IH3}; 2, 5),\ (\mathrm{IH5}; 1, 4),\ (\mathrm{IH27}; 1\tfrac{1}{2}, 4),\ (\mathrm{IH31}; 1, 3)$$
$$(\mathrm{IH33}; 1, 3),\ (\mathrm{IH41}; 1, 3),\ (\mathrm{IH43}; 1, 3),\ (\mathrm{IH47}; 2\tfrac{1}{2}, 4\tfrac{1}{2}),\ (\mathrm{IH52}; 1, 3),\ (\mathrm{IH55}; 2, 4)$$

The remaining eight types in Dress's classification require the use of the GLUE operation and produce tilings with different numbers of A and B tiles. Escher did not create any two-motif patterns with this property. In principle, it would be simple to add these types to the Escherization process, although it would require the implementation of a new class of tiling. I choose not to consider these types in the present work.

Figure 4.19 gives two examples of Heaven and Hell Escherization. The special figure-and-ground relationship maintained by the two prototiles is best depicted by using only two colours: one colour each for the A tiles and B tiles.

4.6.3 Sky and Water designs

Escher's print *Sky and Water* is a very special application of Heaven and Hell tilings. What starts out in the center of the print as a dihedral tiling of stylized fish and birds evolves towards the top and bottom into realistic drawings: birds above, and fish below. Escher used this device in many prints and sometimes multiple times in a single print (as in *Verbum* and *Metamorphose II*).

It is critical that the central tiling where the birds and fish interface be a Heaven and Hell tiling, and not just a split isohedral tiling. The stylized birds evolve into the background for the realistic fish (and vice versa), and so the tiling needs to be colourable with one colour for each tile shape.

Heaven and Hell Escherization seems very well suited to the construction of Sky and Water designs because the realistic goal shapes are already part of the process that leads to the stylized tile shapes. To turn a Heaven and Hell tiling into a Sky and Water design, it suffices to gradually blend the tile shape into the goal shape as tiles are placed successively farther from a given "interface line."

I have embedded the basic Heaven and Hell Escherization algorithm into a suite of interactive tools for constructing Sky and Water designs. One tool lets the user specify a set of tiles to draw and an interface line. Another tool lets the user add decorations to tiles with monochromatic vector-based strokes. Each stroke is a sequence of Bézier curves with user-specified widths; the curves are

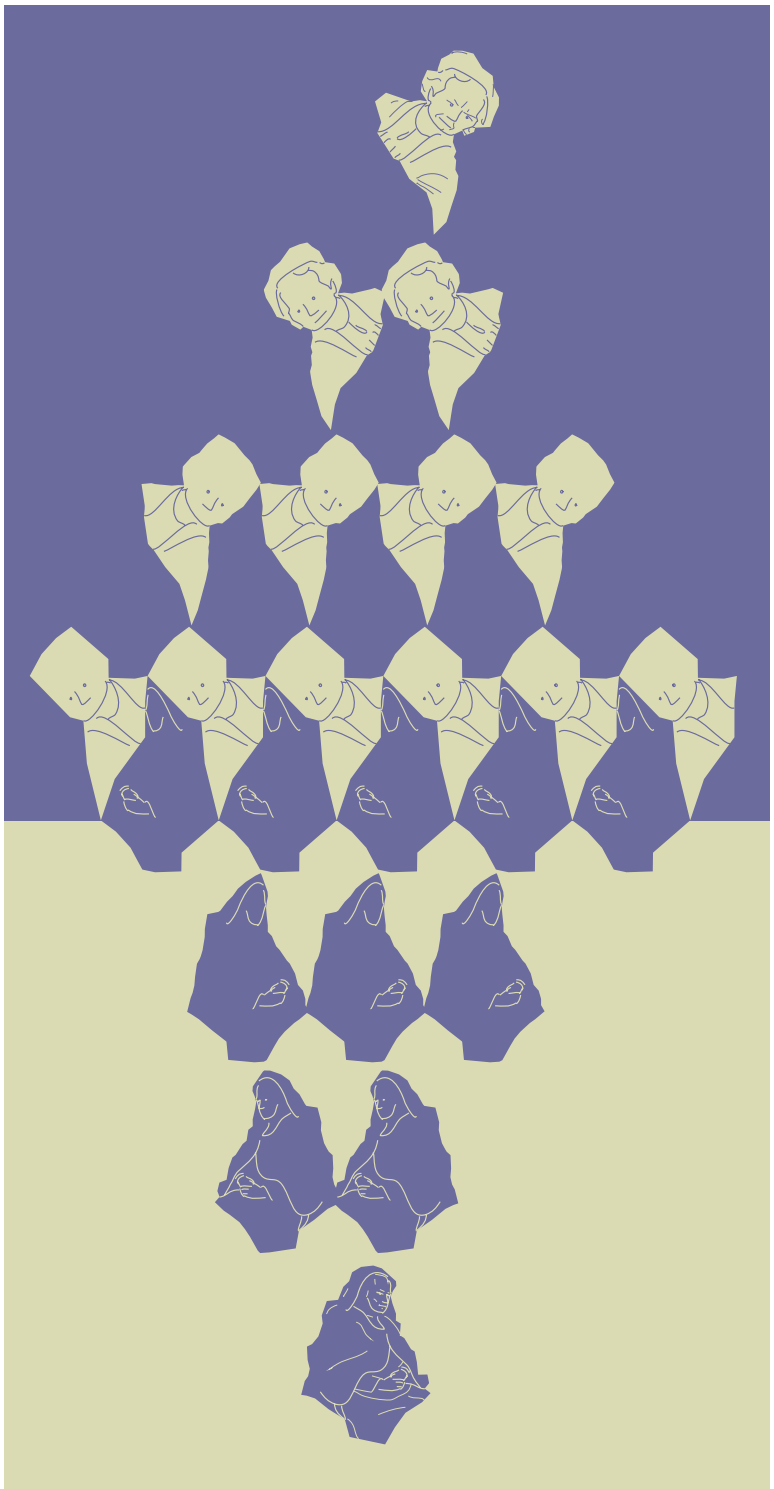


Figure 4.20 An example of a Sky and Water design, based on the goal shapes of Figure 4.18(d).

fit to the user's drawing gestures using the method of Schneider [125]. Additionally, each stroke is given a "priority" that determines how far from the interface line the tile must be before the stroke is drawn. This approach allows for a prioritized sequence of strokes ordered by their importance in expressing a stylized version of the goal shape.

Finally, a renderer assembles the final drawing (in Postscript), taking the output of the other two tools as input, together with colours for the A and B tile shapes. For every tile, the renderer interpolates that tile with its corresponding goal shape by an amount determined from its distance to the interface line. The interpolation is carried out so that any tiles that touch or cross the line are set to the tile shape, and maximally distant tiles are set to the goal shape. The A tiles are then drawn over a solid background of the B tile colour, and vice versa. Finally, the strokes are warped into place and drawn if they have sufficiently high priorities.

Figure 4.20 shows an example of an Escherized Sky and Water design.

4.6.4 Escherization using Penrose tiles

In Section 3.10, I showed how to construct quasiperiodic Islamic star patterns based on a lattice-projection method that produces tilings by rhombs. It happens that a suitably-chosen lattice projection yields the Penrose set $P3$, consisting of a thin and a thick rhomb. Here, we return to the subject of Penrose tiles and consider the question of creating Escher-like tilings based on Penrose sets $P2$ and $P3$.

Unfortunately, Escher did not live to see the development of Penrose tilings, and so we can only imagine what sorts of creatures he might have discovered in them. Penrose himself, who corresponded regularly with Escher, expresses his regret at the missed opportunity [118]. He also gives an example of what Escher might have drawn: a modification of the aperiodic tile set $P2$ where the kites and darts have been turned into chickens. Over time, other artists have created Escher-like designs based on Penrose tiles, though traditional periodic designs are still much more popular. This discrepancy can be explained at least partly by the fact that the range of legal shapes of Penrose tiles is far from obvious. Whereas any layperson can quickly grasp the structure of the simpler isohedral types, understanding the Penrose tiles requires some awareness of the underlying mathematics.

As was pointed out in Section 2.3.3, the basic kite and dart shapes are not an aperiodic tile set; many different periodic tilings can be constructed from them. In order to enforce aperiodicity, the tiles must be augmented with matching conditions that determine the legal ways one tile may be placed next to another. These matching conditions are expressed in several different ways. Symbolic colours can be assigned to the tile's vertices, in which case every tile that shares a tiling vertex must have the same colour there. Or the edges can be labeled, in which case two tiles that share an edge must label it the same way. Most importantly, the edges can be deformed so that tiles only fit together in the desired ways. Grünbaum and Shephard give such geometric matching conditions for Penrose's aperiodic sets $P2$ and $P3$ [68, Section 10.3]. In both cases, the matching conditions are boiled down to two non-congruent **J** edges and the way they are arranged around the two tiles in the set.

The geometric matching conditions immediately give rise to an Escherization algorithm for Penrose tiles. The tiling vertices remain fixed, and the optimization operates on the degrees of freedom in the two fundamental edge shapes. These edge shapes are assembled into two tile shapes that are then compared against two goal shapes as usual.

Unfortunately, this interpretation of the possible shapes of Penrose tiles is rather limited, as can be seen in Grünbaum and Shephard's reproduction of Penrose's aperiodic chicken tiling [68, Figure 10.3.13]. They overlay the chickens with the corresponding unmodified tiling. The registration of these two tilings reveals that the chickens have tiling vertices that are different from those of the original tiling! There are additional degrees of freedom to the Penrose tilings that must be explored and exploited if we are to extend the reach of aperiodic Escherization. Accordingly, I derive a parameterized space that generalizes the aperiodic set $P2$, the Penrose kite and dart. The same arguments apply to set $P3$, the Penrose rhombs.

The additional degrees of freedom are wrapped up in the positions of the tiling vertices, and so we seem to be looking for yet another set of tiling vertex parameterizations. But some care must be taken when thinking about the tiling vertices. The Penrose tilings make no guarantee about the transivity of tiles, meaning that in a given tiling, different instances of the same prototile will be surrounded differently. As a result, two tiles with the same shape may nevertheless have different tiling vertices. In that case, it makes little sense to speak of parameterizing a prototile's tiling vertices once and for all.

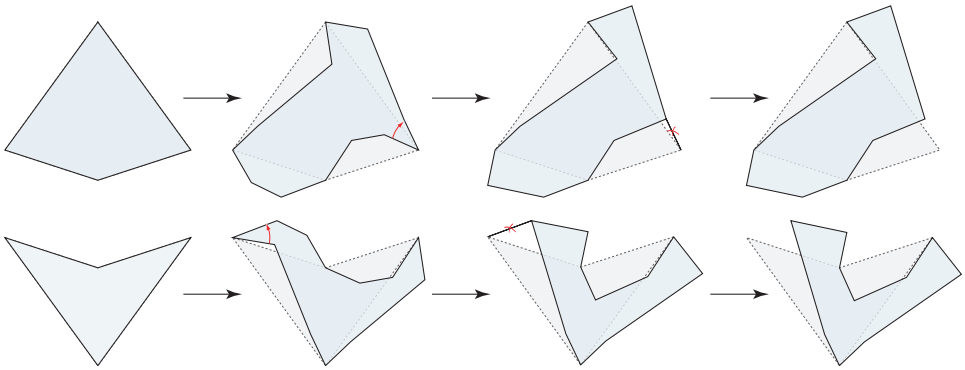


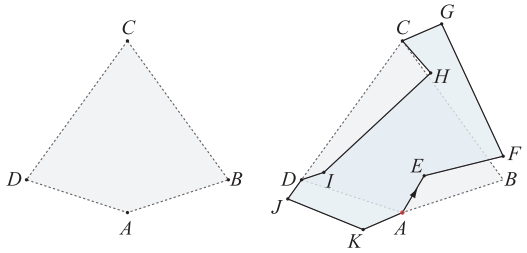
Figure 4.21 An illustration of how a tiling vertex parameterization can be derived for the Penrose kite and dart. The original edges are modified using Grünbaum and Shephard's edge matching conditions [68, Page 539]. When adjacent edges coincide, they are removed, displacing the tiling vertices between the edges. The kite and dart each have one unconstrained tiling vertex. The others are all implied by the original matching conditions.

By experimenting with the geometric matching conditions, I have discovered an extended set of points that can be parameterized like the tiling vertices of an isohedral tiling. I call these points the prototile's *quasivertices*. The quasivertices include all the points on a prototile's boundary that act as tiling vertices somewhere in a Penrose tiling, and some additional points that are forced into existence by the creation of these new tiling vertices.

Figure 4.21 shows how one may use the ordinary geometric matching conditions to derive the new set of parameterizable points for the kite and dart. The fundamental edge shapes are modified so that they partially overlap. The overlapping regions can then be excised from the tiles, producing new tiles that no longer share all of the tiling vertices of the original kite and dart. This process necessarily introduces other vertices into the shapes of the two tiles.

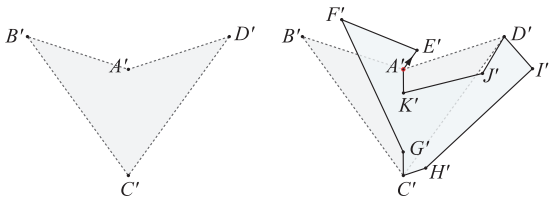
From Figure 4.21, we conclude that the quasivertices of the kite and dart can be parameterized using four real-valued parameters, determined by the positions of the tiling vertices created at the tips of the two excised regions. Similarly, four parameters suffice to parameterize the Penrose rhombs. Once the free parameters are understood, we can derive explicit formulae for the positions of the quasivertices. Formulae are given for the kite and dart in Figure 4.22, and for the rhombs in Figure 4.23.

This additional effort at parameterization may at first seem superfluous, because once portions of



$A = (0, 0)$
$B = (\cos(\frac{\pi}{10}), \sin(\frac{\pi}{10}))$
$C = (0, \frac{1+\sqrt{5}}{2})$
$D = (-\cos\frac{\pi}{10}, \sin\frac{\pi}{10})$

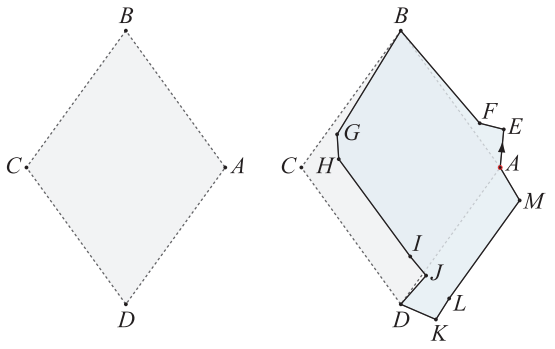
$E = A + r_1(\cos(\theta_1 + \frac{\pi}{10}), \sin(\theta_1 + \frac{\pi}{10}))$
$F = B + r_2(\theta_2 + \cos(\frac{7\pi}{10}), \sin(\theta_2 + \frac{7\pi}{10}))$
$G = C + r_1(\cos(\theta_1 - \frac{\pi}{10}), \sin(\theta_1 - \frac{\pi}{10})$
$H = \text{rotate}(C, -\frac{2\pi}{5}, H)$
$I = \text{rotate}(C, -\frac{2\pi}{5}, G)$
$J = \text{rotate}(A, -\frac{4\pi}{5}, G)$
$K = \text{rotate}(A, -\frac{4\pi}{5}, F)$



$A' = (0, 0)$
$B' = (-\cos\frac{\pi}{10}, \sin\frac{\pi}{10})$
$C' = (0, -1)$
$D' = (\cos(\frac{\pi}{10}), \sin(\frac{\pi}{10}))$

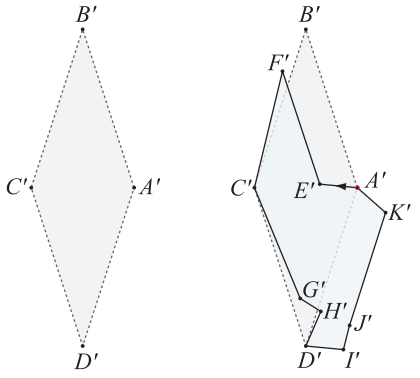
$E' = A' + r_2(\cos(\theta_2 + \frac{\pi}{2}), \sin(\theta_2 + \frac{\pi}{2}))$
$F' = B' + r_1(\cos(\theta_1 - \frac{\pi}{10}), \sin(\theta_1 - \frac{\pi}{10}))$
$G' = C' + r_2(\cos(\theta_2 + \frac{7\pi}{10}), \sin(\theta_2 + \frac{7\pi}{10}))$
$H' = \text{rotate}(C', -\frac{2\pi}{5}, G')$
$I' = \text{rotate}(C', -\frac{2\pi}{5}, F')$
$J' = \text{rotate}(A', -\frac{4\pi}{5}, F')$
$K' = \text{rotate}(A', -\frac{4\pi}{5}, E')$

Figure 4.22 A tiling vertex parameterization for generalized Penrose kites and darts, controlled by four real-valued parameters r_1, θ_1, r_2, and θ_2. The vertices are enumerated in counterclockwise order starting at A for the kite and A' for the dart. The function $\text{rotate}(p, \theta, q)$ rotates point q by angle θ about point p.



$$
\begin{aligned}
A &= (\sin\tfrac{\pi}{5}, 0)\\
B &= (0, \cos\tfrac{\pi}{5})\\
C &= (-\sin\tfrac{\pi}{5}, 0)\\
D &= (0, -\cos\tfrac{\pi}{5})
\end{aligned}
$$

$$
\begin{aligned}
E &= A + r_1(\cos(\tfrac{7\pi}{10} - \theta_1), \sin(\tfrac{7\pi}{10} - \theta_1))\\
F &= A + r_2(\cos(\tfrac{7\pi}{10} - \theta_2), \sin(\tfrac{7\pi}{10} - \theta_2))\\
G &= \text{rotate}(B, -\tfrac{2\pi}{5}, F)\\
H &= \text{rotate}(B, -\tfrac{2\pi}{5}, E)\\
I &= D + r_2(\cos(\tfrac{\pi}{2} - \theta_2), \sin(\tfrac{\pi}{2} - \theta_2))\\
J &= D + r_1(\cos(\tfrac{\pi}{2} - \theta_1), \sin(\tfrac{\pi}{2} - \theta_1))\\
K &= \text{rotate}(D, -\tfrac{2\pi}{5}, J)\\
L &= \text{rotate}(D, -\tfrac{2\pi}{5}, I)\\
M &= \text{rotate}(D, -\tfrac{2\pi}{5}, H)
\end{aligned}
$$



$$
\begin{aligned}
A' &= (\sin\tfrac{\pi}{10}, 0)\\
B' &= (0, \cos\tfrac{\pi}{10})\\
C' &= (-\sin\tfrac{\pi}{10}, 0)\\
D' &= (0, -\cos\tfrac{\pi}{10})
\end{aligned}
$$

$$
\begin{aligned}
E' &= A' + r_1(\cos(\tfrac{6\pi}{5} - \theta_1), \sin(\tfrac{6\pi}{5} - \theta_1))\\
F' &= B' + r_2(\cos(\tfrac{7\pi}{5} - \theta_2), \sin(\tfrac{7\pi}{5} - \theta_2))\\
G' &= \text{rotate}(C', -\tfrac{4\pi}{5}, F')\\
H' &= D' + r_1(\cos(\tfrac{3\pi}{5} - \theta_1), \sin(\tfrac{3\pi}{5} - \theta_1))\\
I' &= D' + r_1(\cos(\tfrac{\pi}{5} - \theta_1), \sin(\tfrac{\pi}{5} - \theta_1))\\
J' &= \text{rotate}(A', \tfrac{4\pi}{5}, F')\\
K' &= \text{rotate}(A', \tfrac{4\pi}{5}, E')
\end{aligned}
$$

Figure 4.23 A tiling vertex parameterization for generalized Penrose rhombs, controlled by four real-valued parameters r_1, θ_1, r_2, and θ_2. The vertices are enumerated in counterclockwise order starting at A for the thick rhomb and A' for the thin rhomb.

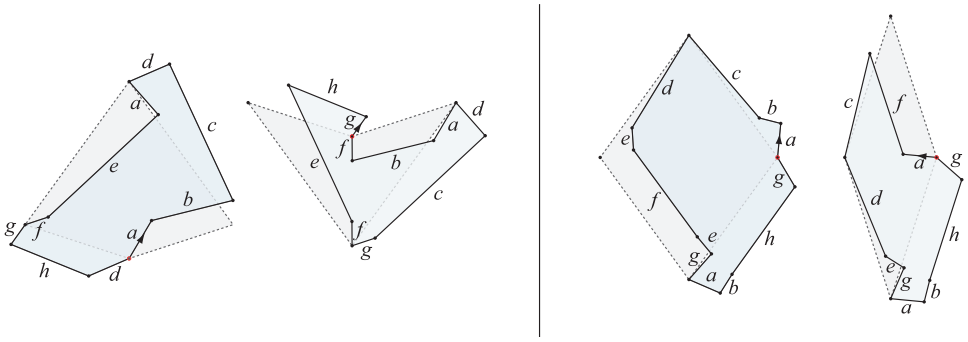


Figure 4.24 Edge labels for the tiling edges of the two sets of Penrose tiles, in the spirit of the incidence symbols used for the isohedral tilings. The kite and dart are shown on the left, and the two rhombs on the right. (The edge labels are not related between the two sets.) Pairs of labels correspond as described in the text.

the tile boundaries are made to overlap, the tiling is indistinguishable from the one created with the overlapping edges excised. The problem is that although tiles with degenerate regions are visually indistinguishable from those without, they are very different from the point of view of the shape metric. The degenerate regions still occupy part of the tile's arclength, and become quite visible when the tile shape is converted into a turning function. We choose to solve this problem at the level of the original tiles, rather than attempting to extend the shape metric with heuristics to detect and discount such regions.

Now that we have grown our original set of tiling vertices to this new set of quasivertices, some additional work needs to be done to give new matching conditions on the edges that join quasivertices. Taking inspiration from the use of incidence symbols in isohedral tilings [68, Section 6.2], the possible edge shapes can be specified by labeling the edges around each tile and indicating adjacency rules for the labels. The edges of the kite and dart can be labeled $abcdaefghd$ and $ghefgcdabf$ respectively, where the enumerations start at the edges marked with arrows in Figure 4.22. To enforce matching between adjacent tiles, we require that the pairs (a, d), (b, h), (c, e), and (f, g) interlock. In effect, a given kite and dart will have only four non-congruent edge shapes between them. Similarly, we can label the edges of the thick and thin rhombs respectively as $abcdefegabhg$ and $afcdegabhg$, with the requirement that pairs (a, g), (b, e), (c, d), and (f, h) interlock. These labelings of the edges of the Penrose tiles are shown in Figure 4.24.

The edge shapes, combined with the four parameters controlling the tiling vertices, yield a space suitable for Penrose Escherization. As with the split isohedral case, I optimize over this space, attempting to minimize the maximum of the pairwise shape comparisons. Note that this parameterization cannot in general represent both a particular pair of tiles shapes and the reflections of those shapes, and that in each of the Penrose sets $P2$ and $P3$ the two prototiles are fundamentally different shapes. For these reasons, given two goal shapes S_1 and S_2 and their reflections S_1' and S_2', we must optimize for the eight combinations $\{(S_j, S_k), (S_j', S_k), (S_j, S_k'), (S_j', S_k')\}_{j,k\in\{1,2\},j\neq k}$ (where S' is a reflection of S, as in Section 4.6.1).

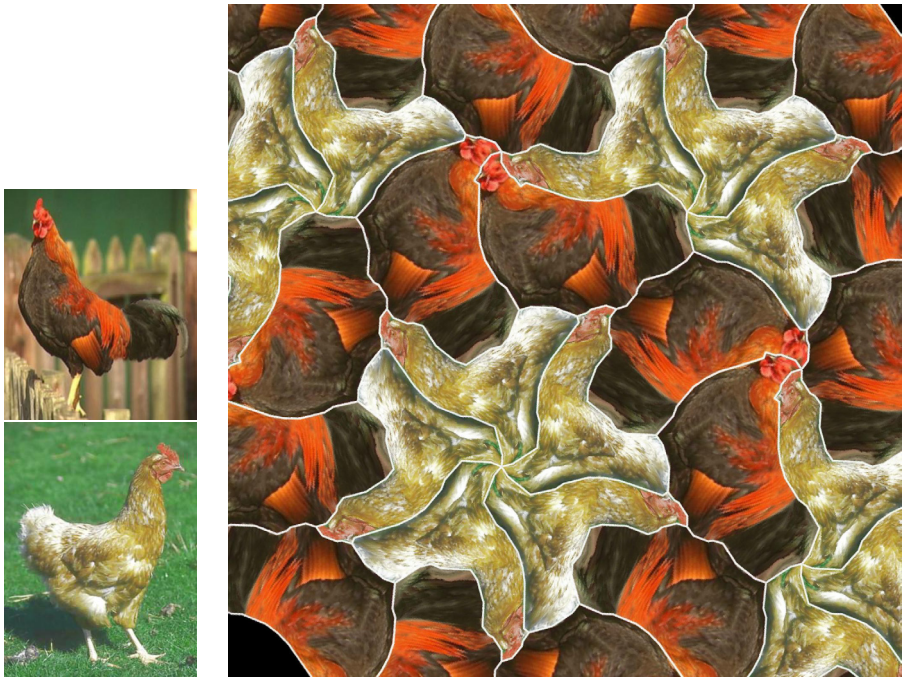
Rendered drawings based on Penrose tilings are given in Figure 4.25 and Figure 4.26. In general, it is much more difficult to discover satisfactory Escherizations using Penrose tilings. The range of possible tile shapes is limited and somewhat peculiar, always having many sharp angles. More obviously, the number of Penrose "tiling types" is much smaller than with the isohedral tilings: we no longer have the luxury of hunting over many quite different types for one that happens to be particularly well suited to a given pair of goal shapes. The results are therefore less successful than in the isohedral and 2-isohedral cases, but interesting nevertheless for their connection to the interaction (both mathematical and personal) between Escher and Penrose. On the other hand, the interactive editor for Penrose tiles still allows profitable forward exploration of the space of tilings. Figure 4.26(b) shows an example of a tiling that was not Escherized but developed from scratch in a few minutes, and then decorated in a cartoon style. As always, Escherization is but one possible step in the design of decorative tilings. The remaining steps, divorced from the Escherization algorithm, may still be used to create attractive tesselations.

4.7 Non-Euclidean Escherization

Our search for methods to create Escher-like tesselations brings us finally to non-Euclidean geometry.

One of Escher's personal artistic quests was the representation of infinity. The infinite manifested itself in many ways in his art — as the infinity of three dimensional perspective in *Depth* and *Cubic Space Division*, as the infinity of endless spirals in *Path of Life* and *Sphere Surface with Fish*, and as the implied infinity of strange loops in *Ascending and Descending* and *Waterfall*.

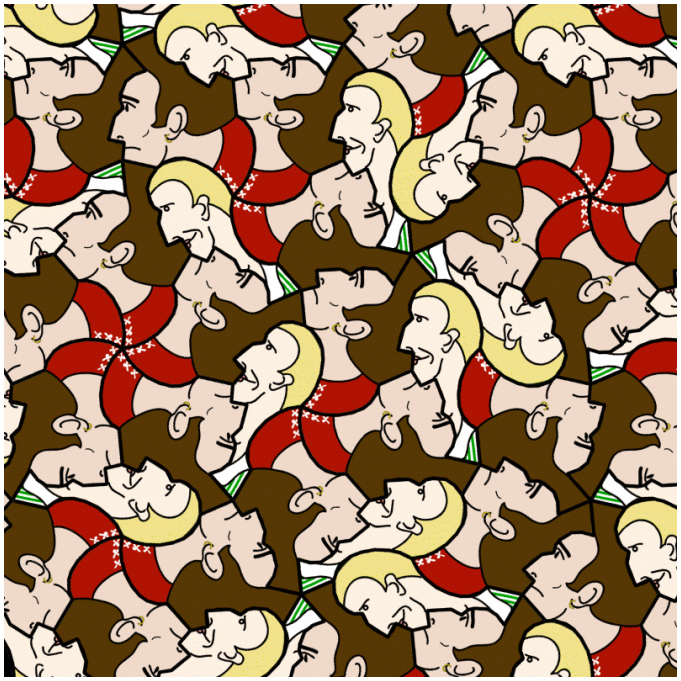
(a) *Dogs and Cats Living Together* (P2)



(b) *Busby Berkeley Chickens* (P2)

Figure 4.25 Examples of dihedral Escherization using Penrose kites and darts. Because of the more limited search space, kites and darts enjoy fewer successful Escherizations than split isohedral tilings.

(a) *A Walk in the Park* (P3)



(b) *The Pentalateral Commission* (P3)

Figure 4.26 Examples of ornamental tesselations based on Penrose rhombs. The top image was Escherized in the usual sense; the bottom one was developed from scratch in a few minutes using an interactive editor.

It was in his regular divisions of the plane that Escher searched the hardest. There, the artistic infinity might find expression in a mathematical infinity, and the totality of an imagined world might be realized on a single piece of paper.

Certainly, his notebook drawings offer the suggestion of infinity, though only in a theoretical sense. We can imagine the tiles of a tiling being extended out in all directions but any drawing of the tiling can only contain a small patch. Escher worked hard at surmounting this obstacle, devising a number of clever systems of tiles that interlocked with scaled-down versions of themselves, creating an ever-diminishing pattern that crept toward a limit. Often, that limit was the center of a circle (as seen for example in *Smaller and Smaller*). Escher found these designs unsatisfying because there was no natural boundary to the drawing. New tiles could always be added to the outer edge, implying that infinity had still not been reined in. Truly capturing infinity in a single drawing would require a limit along the drawing's *outer* edge; that way, an entire universe of tiles would be completely contained in a finite region with no logical room for additions [49, Page 41].

Escher finally achieved his goal when he learned of the work of Coxeter. Coxeter's books on geometry contain visualizations of hyperbolic symmetry groups in the Poincaré model, visualizations that were precisely the kinds of tilings Escher was searching for. With Coxeter's guidance, he managed to produce a small number of remarkable *Circle Limit* prints. Each one is a testament not only to Escher's artistry and intuition, but also to his tenacity — he tirelessly pushed the design as close as possible to the bounding circle, down to the physical limits of his tools, his medium, and his body.

Escher also created several lovely spherical interpretations of his tesselations. They are beautifully carved bas-relief wooden sculptures, sometimes stained in multiple shades. These spheres are a clear expression of boundlessness, an intellectual and aesthetic cousin of infinity.

I would of course like to create Escher-like tesselations in the hyperbolic plane and on the sphere. There are two approaches that might be taken to accomplish this goal. We could rewrite the Escherization algorithm to operate directly in hyperbolic or spherical geometry (or, as was done in Chapter 3, perhaps we could write a single algorithm in absolute geometry). Alternatively, we could adopt the approach that Dunham uses to great effect [40, 43], taking existing tilings and adapting them to other geometries (or other symmetry groups in the same geometry).

The first option, though theoretically possible, is infeasible in practice. This section will ul-

timately present tools based on the second option, but beforehand some discussion is warranted concerning the challenges involved in expressing the Escherization algorithm in non-Euclidean geometry.

As always, the natural approach to an Escherization algorithm is to view it as an optimization process and to provide a suitable parameterized space and evaluation function.

The definition of an isohedral tiling carries over into non-Euclidean geometry, and so our first step would be to parameterize the space of non-Euclidean isohedral tilings. The combinatorial structure of these tilings is well understood. Grünbaum and Shephard give a classification of the spherical isohedral tilings in the same spirit as their Euclidean classification [67]. Delaney symbols can be used to enumerate the hyperbolic types [86]. Although there are infinitely many isohedral tiling types in the hyperbolic plane, it would be possible to enumerate a manageable set of them that are suitable for Escherization.

Deriving the tiling vertex parameterizations would be the next challenge. As before, for each tiling type we can write down a set of constraints that must be satisfied by the angles and edges of the tiling polygon. Unfortunately, in non-Euclidean geometry those constraints will not have solutions that are linear in some real-valued parameters. The tiling vertex parameterizations will therefore be more complex.

The main problem would be in adapting the shape comparison metric. "Shape" does not mean the same thing in Euclidean and non-Euclidean geometry. Arkin's metric takes for granted the fact that shape is to be considered independent of a polygon's absolute size. But in the hyperbolic plane and on the sphere, size is intimitely tied to shape. A triangle with a given set of interior angles will necessarily have a uniquely-determined size. Factoring out scale does not make sense outside of the Euclidean plane (rotation, too, would present difficulties). New shape metrics would have to be developed for the hyperbolic plane and on the sphere, metrics that do not require ambient space to be affine.

We therefore take Dunham's approach, and look for a way to re-map a preexisting tiling into another geometry. The technique and results presented here are similar to his and are included as a proof of concept, in the interest of completeness. One difference from Dunham's method is that I demonstrate how to render hyperbolic designs from *images*, yielding non-Euclidean image-based results. Dunham works only from vector-based art. I also demonstrate a technique for mapping

some Euclidean tilings to the sphere.

The translation into non-Euclidean geometry begins with the identification of a region of a Euclidean tiling called a "swatch," which is suitable for adaptation. Typically, the swatch will be a square or an equilateral triangle (the examples in this section use a square).

Then, a transformation maps the swatch into a region of the Poincaré model of the hyperbolic plane that plays an equivalent role in a hyperbolic symmetry group. If the mapping is well-behaved, then the transformed copy of the original region can be replicated to cover the hyperbolic plane, elaborating a tiling similar to the Euclidean source. An analogous process maps Euclidean swatches onto the surface of a sphere.

Not every Euclidean tiling can be adapted in this way, and not every non-Euclidean symmetry group can serve as the target of translation for some given source region. The periodicity of the isohedral tiling is a Euclidean property, defined in terms of translational symmetries. A translation to non-Euclidean geometry must be made based on properties of tilings that do not depend on translation.

We do have one well-known family of symmetry groups at our disposal that span Euclidean and non-Euclidean geometry. These are the groups of the form $[p, q]$ and $[p, q]^+$ (the latter were mentioned in Section 3.7.3). We have already seen in the case of Islamic star patterns how small changes to p or q can create a family of related designs in different geometries: see Figure 3.26 for examples. It seems reasonable therefore that Euclidean tilings with symmetry groups $[4, 4]$, $[4, 4]^+$, $[6, 3]$, and $[6, 3]^+$ could be adapted to produce closely related non-Euclidean interpretations. In the usual international notation for wallpaper groups, these groups are known respectively as $p4m$, $p4$, $p6m$, and $p6$. We can look for isohedral tiling types with these symmetry groups as sources for experimentation. Groups $p4m$ and $p6m$ are only associated with tiling types that make for uninteresting Escher tilings. The remaining isohedral types worth considering in this approach are IH11, IH21, IH28, IH31, IH34, IH39, IH55, IH61, IH62, IH79, IH88, and IH90. Of course, other types can be given non-Euclidean analogues, as Dunham consistently demonstrates. The types above are those that apply directly in the simple method presented here.

For the moment, we will ignore the problem of colouring, and focus on translating an uncoloured tiling. Let us consider the tiling called "Tea-sselation," shown in Figure 4.15(e). It is of isohedral type IH28, which has symmetry group $[4, 4]^+$ as discussed above. In this symmetry group, centers

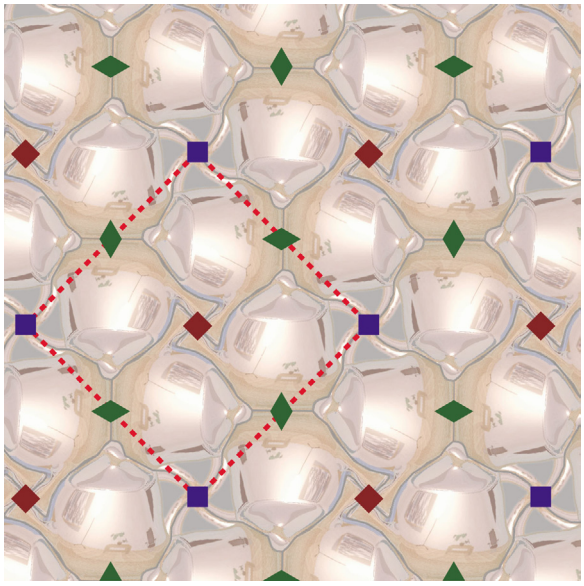
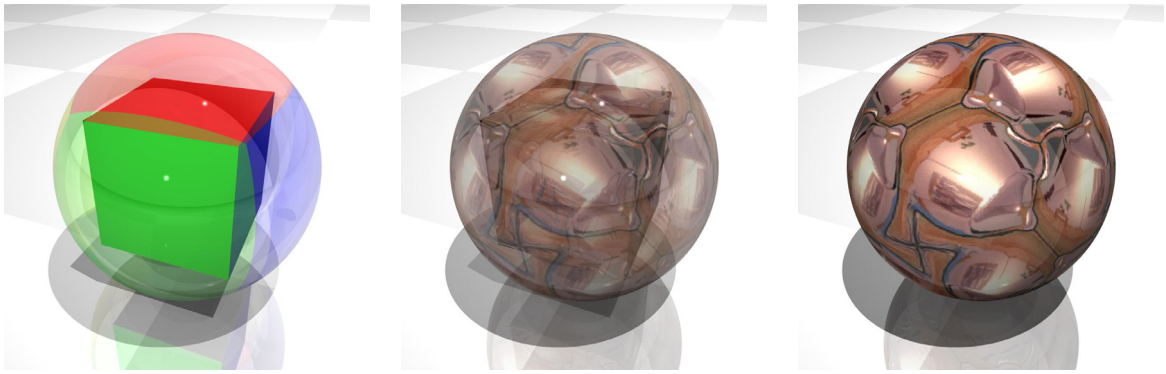


Figure 4.27 A visualization of the symmetry group for *Tea-sselation*. Red and blue squares indicate two inequivalent families of fourfold rotational symmetries. Green lozanges indicate twofold rotational symmetries. The highlighted square can be extracted from the tiling and used to construct related designs in non-Euclidean geometry.

of fourfold rotation come in two inequivalent families. By joining up one set of centers into square units as shown in Figure 4.27, we can think of any pattern with $p4$ symmetry as being expressed by a collection of square swatches. In "Tea-sselation," there are two ways to build this region: we can join together the meeting places of the handles, or of the spouts. I choose the former. By varying q, we can translate the teapots to hyperbolic tilings with symmetry groups $[4,5]^+$ and $[4,6]^+$ (and, more generally, any $[4,q]^+$ for $q > 4$). We can also translate the tiling to $[4,3]^+$ (the symmetry group of the cube).

It is relatively easy to map the tiling onto a sphere, using the symmetry group of the cube. Given a point on the sphere, we project that point onto the surface of a concentric cube. Each face of the cube is square and can be covered with a copy of the swatch. When those pixels are mapped back onto the spherical surface, the result is a spherical interpretation of the original tiling. This mapping process is illustrated in Figure 4.28.

For the hyperbolic plane, we can transfer the design to symmetry group $[4,q]$ for any $q > 4$. We



(a) (b) (c)

Figure 4.28 A visualization of how a texture is mapped onto the surface of a sphere via the faces of a cube. In (a), we see how each face of the cube relates to a portion of the sphere. The same mapping is repeated in (b), but with the texture mapped to the cube faces. The resulting spherical tiling is shown in (c).

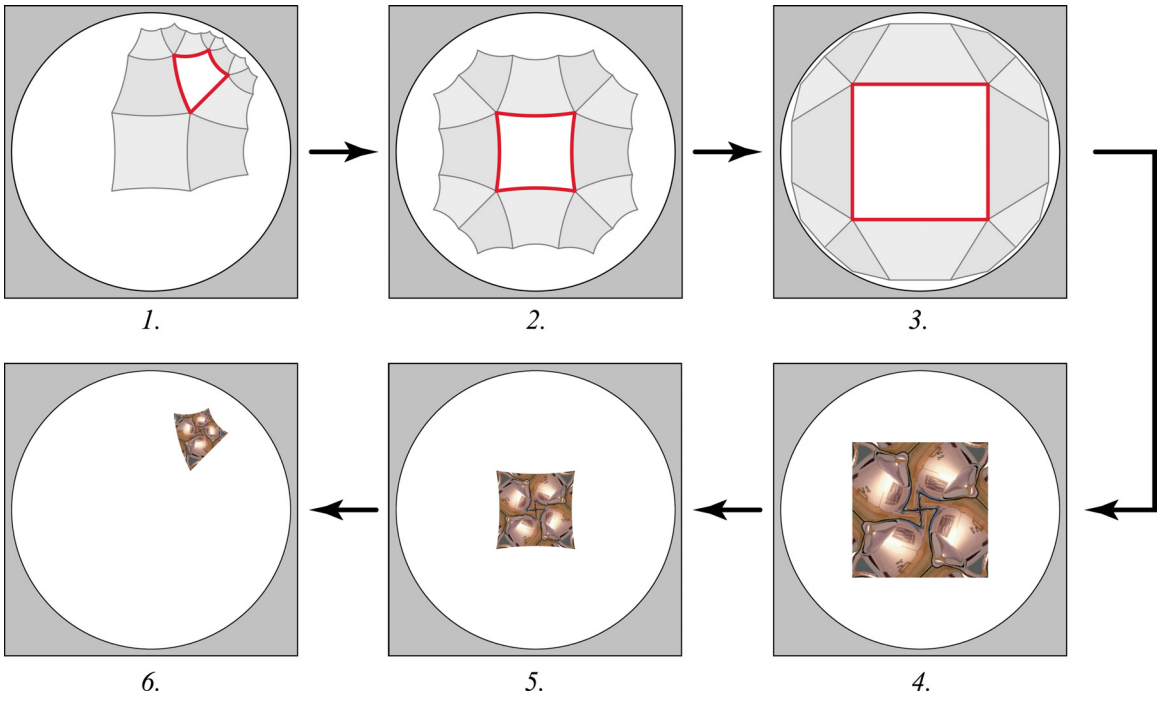


Figure 4.29 Six steps showing how a square Euclidean texture can be warped to fit a given equiangular quadrilateral in the hyperbolic plane. The first step shows the quadrilateral. We use a hyperbolic rigid motion to move it to the origin in step 2, and reproject into the Klein model in step 3. In the Klein model, the quadrilateral is drawn as a square, allowing the texture to be mapped into it easily in step 4. We then undo the reprojection and rigid motion in steps 5 and 6, yielding the final mapped quadrilateral.

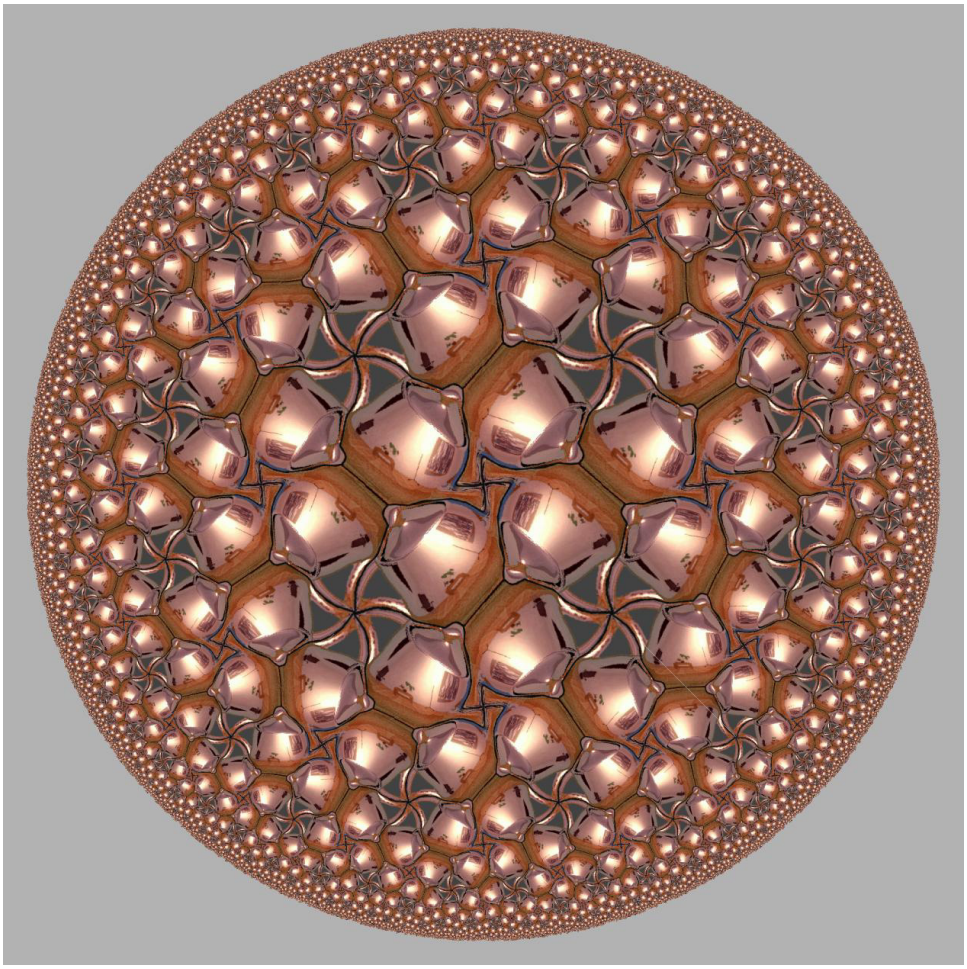


Figure 4.30 An uncoloured interpretation of “Tea-sselation” mapped into the hyperbolic plane with symmetry group $[4,5]^+$.

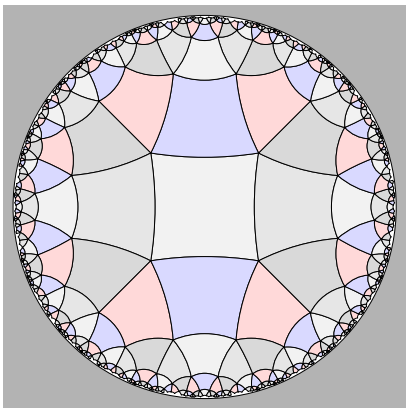
tile the hyperbolic plane with the tiling $\{4, q\}$. Each tile is a regular 4-gon. For each tile, we can rigidly move the tile until it lies centered at the origin of the unit disk in the Poincaré projection. We then switch to the Klein projection, where the regular 4-gon must get mapped to a square. This square can receive the original swatch, and we can then undo the transformations to move the pixels back up to the original 4-gon. The sequence of transformations is visualized in Figure 4.29. An interpretation of "Tea-sselation" in the hyperbolic symmetry group $[4, 5]^+$ is shown in Figure 4.30.

It is also possible to add colour to these designs. The notion of perfect colouring works fine in non-Euclidean geometry. I create coloured designs by constructing multiple swatches that express all the colour combinations I wish to use in the non-Euclidean interpretation, and assembling those swatches according to a selection rule that embodies a perfect colouring. For spherical designs in $[4, 3]^+$, it is easy enough to place the six swatches manually; an example using teapots is shown in Figure 4.32. In the hyperbolic plane, I implement colouring as described by Dunham [39]. He extends the data structure representing a rigid motion to include a permutation of the set of colours. Each primitive rigid motion (rotations around the p-centers and q-centers) get initial permutations. As rigid motions are composed, their associated permutations are composed as well. The colour of every tile in the resulting tiling can be read out as the first element of the transformed permutation. Figure 4.31 gives a perfectly 5-coloured hyperbolic teapot tiling in $[4, 5]^+$. Moving to $[4, 6]^+$ allows for a simpler perfect colouring with only three colours, as shown in Figure 4.33.

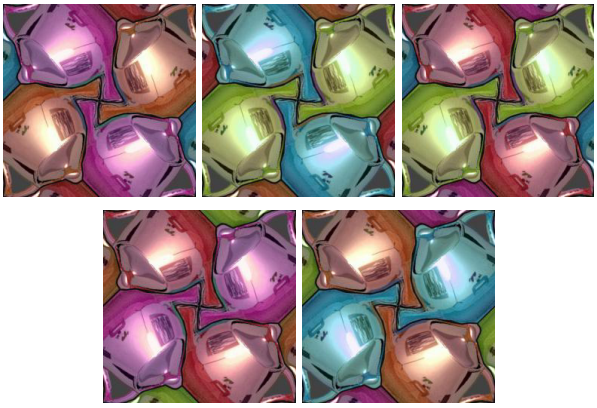
Other tilings can serve as a source for translation to non-Euclidean geometry. As a final example, I turn to Escher's own notebooks. His well known symmetry drawing number 104 features black and white lizards in a tiling of isohedral type IH55. Once again, we can translate this design from the Euclidean $[4, 4]^+$ to the hyperbolic $[4, 6]+$ to simulate a new member of Escher's *Circle Limit* series. The resulting design, "Circle Limit V," is shown in Figure 4.34. Because the original tiling is of topological type 4^4, the hyperbolic interpretation can be perfectly coloured using only two colours.

4.8 Discussion and future work

Most polygons are not tiles. For just about any goal shape, an Escherizer will have to produce an approximation, and a better Escherizer will produce a closer approximation. When the shape metric



(a)



(b)

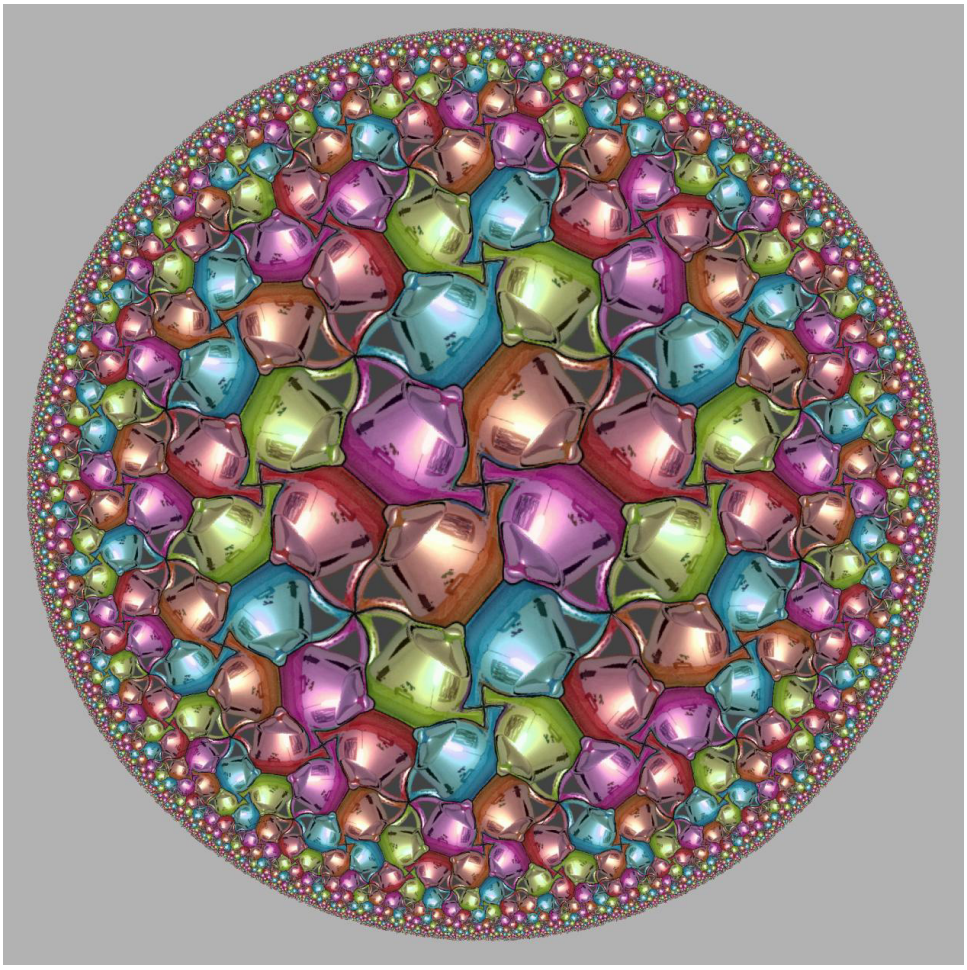


Figure 4.31 A hyperbolic interpretation of *Tea-sselation* using symmetry group $[4,5]^+$. This realization contains almost seven thousand tiles. The tiling is perfectly coloured using five colours, mapped in pairs onto the five swatches shown in (b). The choice of swatch is dictated by the colouring shown in (a).

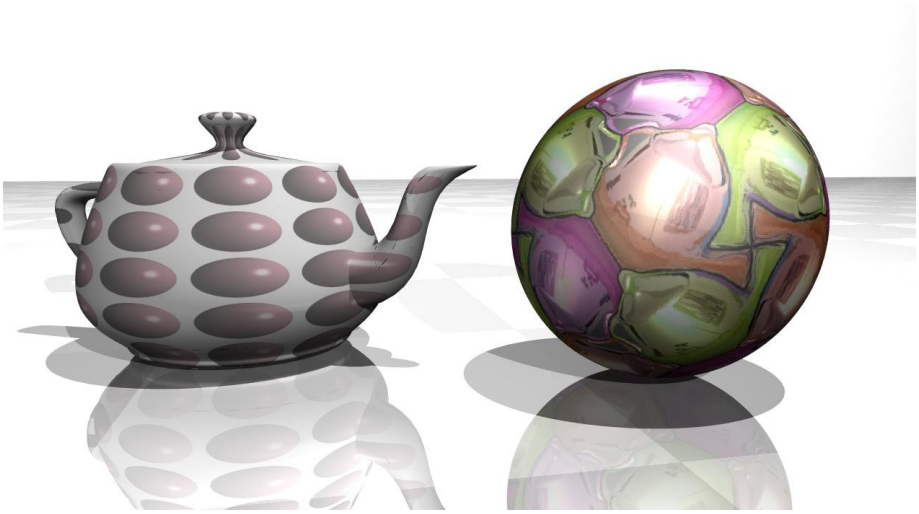


Figure 4.32 Teapots mapped onto a sphere, together with spheres mapped onto a teapot. The sphere features a perfectly 3-coloured teapot tiling in the symmetry group $[3,4]^+$.

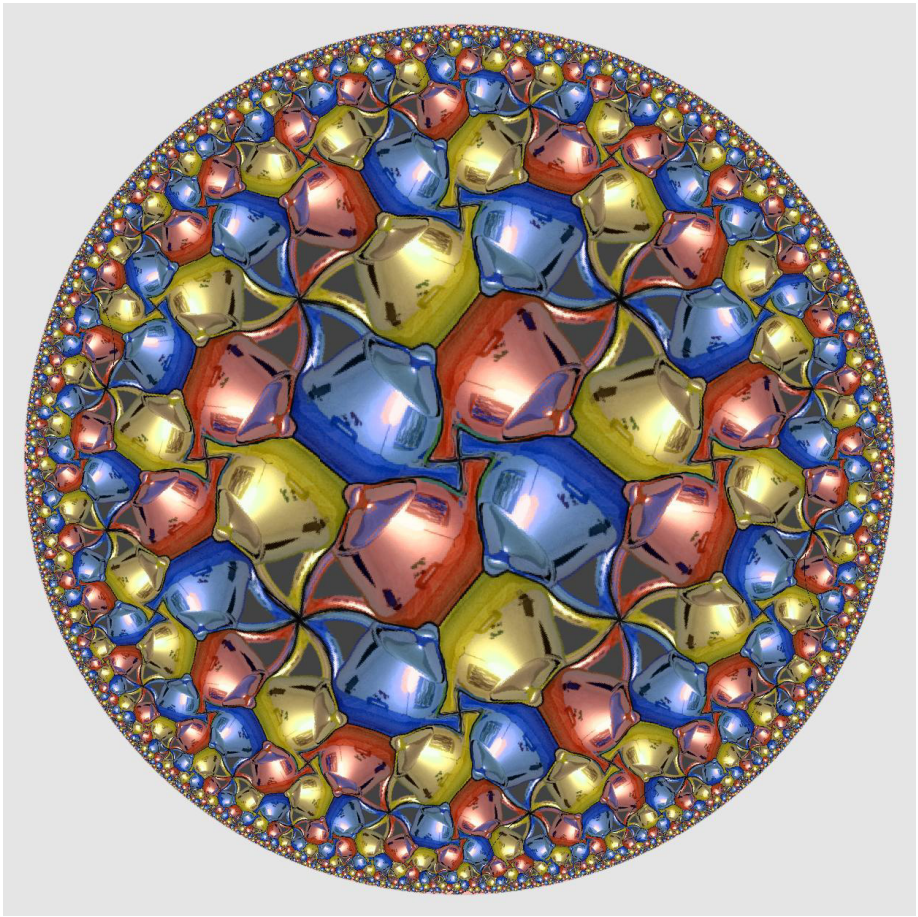


Figure 4.33 Teapots in the hyperbolic symmetry group $[4,6]^+$. This symmetry group permits a perfect colouring of the tiling using only three colours.

Figure 4.34 A simulated successor to Escher's *Circle Limit* drawings, constructed by translating Escher's symmetry drawing number 104 into the hyperbolic symmetry group $[4,6]^+$. *M.C. Escher's symmetry drawing number 104* ©*2000 Cordon Art B.V. – Baarn – Holland. All rights reserved.*

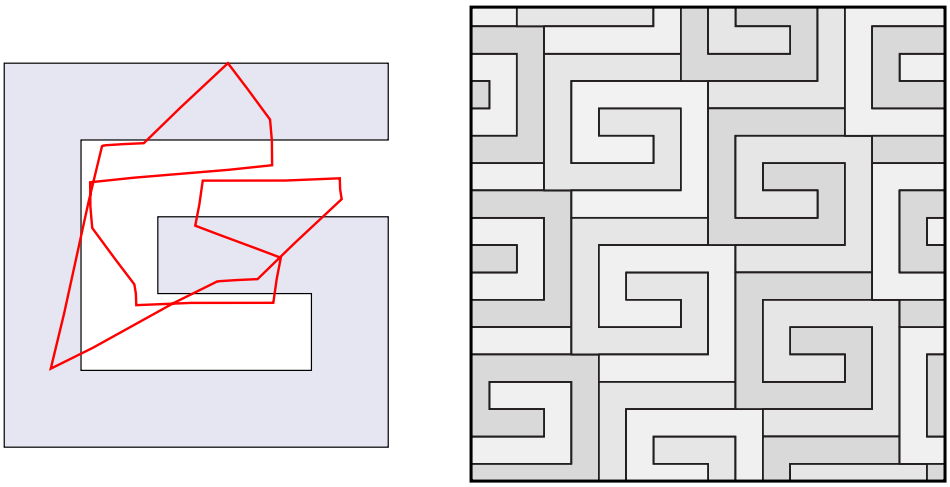


Figure 4.35 A goal shape for which Escherization performs badly, and a tiling that it admits.

is used as the measure of closeness, a perfect Escherizer would determine the smallest distance over all possible tile shapes, and return the tiling that achieves that bound. Our imperfect optimizer, by contrast, coarsely samples the space of isohedral tilings in a directed fashion and returns the best sample it finds. Consequently there are seemingly easy cases, such as the one in Figure 4.35, that our algorithm cannot successfully Escherize.

In practice, our Escherization system performs well on convex or nearly convex shapes. The shapes that tend to fail are the ones with long, complicated edges between the tiling vertices. It is difficult for the optimizer to come up with just right the sequence of vertex adjustments to push a tendril of detail out, especially when constrained by the "no non-uniform noise" condition of the metric. A shape metric for Escherization should not be sensitive to local deformations. As a tendril of detail is pushed out, its contribution to the overall perimeter of the tile shape compresses the rest of the tile's turning function horizontally, potentially worsening the overall comparison. The metric of Arkin *et al.* seems to resist these local changes in detail.

Furthermore, in the shape comparison metric used, the importance of a section of outline is directly proportional to its length as a fraction of the perimeter of the goal shape, even if from an aesthetic point of view outlines may obey different measures of significance. For example, the precise profile edge of a face in silhouette, descending along eyes, nose, and mouth, is much more important than the hairline. But to the current shape metric these might be relatively insignificant details. It would be valuable to investigate an extension to the polygon comparison metric wherein a section of outline could be assigned a measure of importance, a weight controlling which parts of the polygon should match more closely.

Although Escher's tiles are almost always immediately recognizable as particular kinds of animals, they generally bear little actual resemblance to a real image: they are more like conventionalizations, or cartoons. The results presented here inherit a distinctive three-dimensionality from the goal images they capture. My algorithm does not "understand" the shapes it is manipulating, so it has no way to deform them while preserving their essential recognizability. It must instead rely on a purely geometric notion of proximity. A powerful extension to the algorithm would be to add degrees of freedom that control the goal shape as well as the tile shape. In a lizard outline, for instance, these new parameters might allow the legs, head, and tail to bend. The optimization could proceed as before, but now instead of having only the tile evolve to approximate the goal, we would

have both shapes work towards a common meeting point. A very coarse version of this enhancement would be to assemble a collection of related goal shapes, any of which satisfies the artist's intent, and optimize over all of them simultaneously. In the dihedral case, we might for example have a set of bird outlines and a set of fish outlines, and attempt dihedral Escherization on all pairs of a bird and a fish.

Another way to add degrees of freedom to the goal shape would be to begin with a three dimensional model as a goal, and include rotation parameters in the optimization. The model is converted into a polygon by rotating it and taking the silhouette. Some care would be needed here to keep the model from rotating into a position that tiles very well by hiding all of its interesting geometry inside the silhouette. For additional guidance, we might turn to Escher's writing. He points out [49, Page 106] that certain living forms are most easily recognized when viewed from certain characteristic angles: "Four-footed mammals are usually best recognized by looking at them from the side. Reptiles and insects, on the other hand, generally present their most typical aspect when seen from above, while the human figure is at its most characteristic when viewed from the front."

This chapter has presented Escherization algorithms for a few different families of tilings: the isohedral, split isohedral, and Heaven and Hell tilings, and tilings from Penrose sets $P2$ and $P3$. There are always more families of tilings to explore, allowing us to slowly move beyond the mathematical structures Escher worked with. The logical extension of the work with transitive tilings would be to run Escherization on tilings encoded by Delaney symbols, in which case the user might in theory supply any number of goal shapes and have the system discover a k-isohedral tiling that approximates all of them. The number of combinatorial types to be searched grows very quickly with k, but perhaps some small subset of those types could be determined automatically ahead of time. As a benchmark for k-isohedral Escherization, we might take Escher's symmetry drawing 71, a remarkable periodic tesselation featuring twelve distinct bird motifs.

Escher's final symmetry drawing, number 137, hints at an extension to k-anisohedral prototiles (recall that a k-anisohedral prototile is a shape that can only form monohedral tilings with at least k tile transitivity classes). The k-anisohedral tilings are still not very well understood, although Berglund has begun a classification of the case $k = 2$ [9]. The desire to create Escherized anisohedral shapes might help further motivate (and even aid) the search for anisohedral tiles.

Finally, we can consider aperiodic tiling types other than Penrose's. Perhaps by parameterizing

the shapes of other well-known tile sets such as those of Ammann [68, Section 10.4], we might discover candidates for Escherization that can produce more successful tile shapes.

Escherization is merely one powerful tool to be applied to the creation of ornamental tilings. It is no substitute for the mathematical insight or artistic know-how of Escher himself, but it can help the designer of a tiling to take a large step towards a solution. The artist still plays an important role in the process and must harness the power of Escherization to bend it to their aesthetic goals. In creating art by computer, Escherization is simply one means of communication in the dialogue between human and machine.