



Escherization

Craig S. Kaplan¹

David H. Salesin^{1,2}

¹University of Washington

²Microsoft Corporation

Abstract

This paper introduces and presents a solution to the “Escherization” problem: given a closed figure in the plane, find a new closed figure that is similar to the original and tiles the plane. Our solution works by using a simulated annealer to optimize over a parameterization of the “isohedral” tilings, a class of tilings that is flexible enough to encompass nearly all of Escher’s own tilings, and yet simple enough to be encoded and explored by a computer. We also describe a representation for isohedral tilings that allows for highly interactive viewing and rendering. We demonstrate the use of these tools—along with several additional techniques for adding decorations to tilings—with a variety of original ornamental designs.

CR Categories: I.3.5 [Computational Geometry and Object Modeling]: Geometric algorithms, languages and systems; I.3.8 [Computer Graphics]: Applications; J.5 [Arts and Humanities]: Fine arts; J.6 [Computer-Aided Engineering]: Computer-aided design (CAD).

Keywords: Tilings, tessellations, morphing, optimization, simulated annealing, Escher

1 Introduction

Tilings are as old as civilization. Our ancestors’ earliest experience with tilings probably arose out of the quest for regularity in the construction of walls, floors, and ceilings. This regularity could at once simplify the task of construction and lend a sense of order and uniformity to the objects being constructed.

Historical uses of ornamental tilings abound; numerous examples from as early as the twelfth century survive today [15]. Perhaps the most renowned example is the Alhambra palace in Granada, Spain. The Moors who built the Alhambra became masters of geometric ornament, covering every surface of the palace with intricate tilings of astonishing beauty.

By the time the Dutch graphic artist M.C. 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 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 [20]. 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. 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 [18].



Figure 1 M.C. Escher in a self-portrait. M.C. Escher’s “Self-portrait” ©2000 Cordon Art B.V.–Baarn–Holland. All rights reserved.



Figure 2 Escher’s Escher Escherized.

Taking our inspiration from Escher and his elegant work, we attempt to solve the following problem in this paper:

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. copies of T fit together to form a tiling of the plane.

This problem is tricky in that for a sufficiently large perturbation of the goal shape, it is always possible to find a tiling in a trivial sense. (Let T , for example, be a square.) We need to formalize the measure of “closeness” in such a way that it both preserves the “essence” of the goal shape S and at the same time produces new shapes T that are known to tile.

This paper presents a solution to the Escherization problem that 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 2). Creating such tilings requires solving a number of subproblems, which we discuss in this paper. The first difficulty is in selecting a set of tiling types that are both simple enough to be encoded and manipulated by a computer, and flexible enough to express most of the ornamental designs we would like to create. A second problem is in finding consistent and complete parameterizations for these tilings—that is, parameterizations that are always guaranteed to produce correct tilings of a given type (“consistent”) and that are furthermore capable of producing all tilings of that type (“complete”). Though fundamental to the analysis of patterns and tilings, to our knowledge this problem has never before been addressed for the types of tilings we consider. A third problem is in choosing a good measure of closeness. A fourth challenge is in designing an optimizer to search over all possible tiling types, their parameterizations, and tile shapes in order to find a good approximation to the goal tile. A fifth problem is in creating a representation for these tilings that allows for highly interactive viewing and editing. A final problem is in decorating and rendering the resulting tiles.

Unlike most research projects in computer graphics, this one is motivated more by intellectual curiosity than by practical import. Nevertheless, a solution to the Escherization problem does have certain applications in the real world. Tilings are of course useful as floor and wall coverings. In manufacturing, the outlines of tiles can be cut repeatedly out of stone using a process known as water-jet cutting. Automatically-designed tilings could just as easily be carved out of wood or even sewn into a quilt. This suggests a further application, proposed by Chow [5]: a tiling program could be used to lay out copies of a part to be cut out a sheet of some material. If the copies are arranged in a tiling, they can be cut from the sheet without creating any waste material (except around the outer edges of the sheet).

1.1 Related Work

Several authors have explored the possibility of creating ornament in various forms by computer. A paper at the second annual SIGGRAPH conference featured a system for drawing figures constrained to the seventeen planar symmetry groups [1]. More recently, Glassner examined the synthesis of frieze patterns [8] and aperiodic tilings [9, 10], which can be used for generating ornaments for bands and for the 2-D plane, respectively. Wong *et al.* investigated algorithms for computer-generated floral ornament [21] and surveyed other previous work in creating these kinds of ornamental designs.

In addition, software created specifically for allowing users to construct tilings of the plane has been around for at least twenty years. Chow had a very successful FORTRAN program [4] that let the user input the portion of the tile that is independent, *i.e.*, not expressed in terms of some other portion of the tile. The program then filled in the remaining part of the tile and replicated it in the plane. Reptiles [14], by Huson and Friedrichs, is a complex system that understands a large class of mathematically-interesting tilings. Reptiles has since been expanded into Funtiles, an even more sophisticated tool that can create tilings in non-Euclidean geometries. Lee’s TesselMania! is a marvelous program for giving children an understanding of symmetry and tilings.

A number of individuals are actively designing new Escher-like tilings, aided by illustration software. Crompton [6] has compiled an extensive list of recent contributions to tessellation-based art.

Still, none of these earlier efforts attempt in any way to find tilings automatically whose tiles approximate a particular goal shape, the work we describe here.

1.2 Overview

We begin with background on the mathematical theory of tilings, leading into a description of the “isohedral” tilings (Section 2), on which the rest of this work is based. We then address each of the remaining subproblems in turn: parameterizing the isohedral tilings (Section 3); developing a measure of “closeness” between two tiles (Section 4); designing an optimizer for finding the best tiles (Section 5); representing the resulting tilings for efficient editing and viewing (Section 6); and decorating and rendering the tiles (Section 7). We end with a discussion of our results (Section 8) and ideas for future work (Section 9).

2 Mathematical theory of tilings

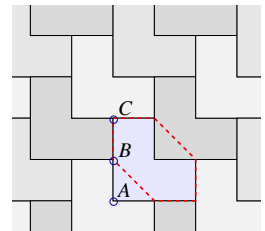
In this section, we present background on only the parts of tiling theory necessary to understand the research work presented in the rest of this paper. Readers seeking a more in-depth analysis of tilings should consult the highly accessible treatise on tiling theory, Grünbaum and Shephard’s *Tilings and Patterns* [11].

2.1 Tilings

A *tiling of the plane* is a collection of shapes, called *tiles*, that cover the plane without any gaps or overlaps. That is, every point in the plane is contained in at least one tile, and the intersection of any two tiles is a set with zero area (we regard tiles as closed sets, and allow them to intersect along their boundaries).

Given certain natural analytic restrictions on the shapes of tiles [11, sec 3.2], the intersection of any set of tiles will either be empty, a point, or a simple curve. When the intersection is a curve, we call that curve a *tiling edge*. When the intersection is a point, in which case that point will necessarily be a meeting place of at least three tiles, we call that point a *tiling vertex*.

Every tile can be decomposed, based on intersections with its neighbours, into a sequence of tiling vertices joined by tiling edges. These must be distinguished from the vertices and edges of the *tiles* (if the tiles are in fact polygons), which we will call *shape vertices* and *shape edges*, respectively, to differentiate them from their tiling counterparts. Although the features of the tiling occupy the same positions as the features of the tiles, they may break down differently. For the blue tile in the tiling on the right, *A* is a shape vertex but not a tiling vertex, *B* is a tiling vertex but not a shape vertex, and *C* is both a tiling vertex and a shape vertex. We will also make use of the *tiling polygon*, the polygon formed by joining the tiling vertices that lie on a given tile, shown here as a red dashed line. This polygon is important in describing the structure of the tiling.



In many of the tilings we see every day on walls and streets, the tiles all have the same shape. If any given tile in a tiling is congruent to any other through a rigid motion of the plane, we say that the tiling is *monohedral*. Similarly, a *k-hedral* tiling is one in which every tile is congruent to one of *k* different prototiles. When *k* = 2, we also use the term *dihedral* to describe the tiling.

2.2 Isohedral tilings

A *symmetry* of a figure in the plane is a rigid motion of the plane that maps the figure onto itself. Every figure in the plane necessarily has an associated set of symmetries, even if it is just the trivial set containing the “identity” motion. It is easy to see that the symmetries of a figure have a natural group structure under composition of

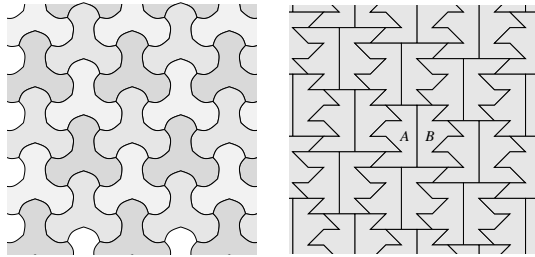


Figure 3 Both of these tilings are monohedral, but the one on the left is isohedral and the one on the right is not. The reflection that maps tile A onto tile B is not a symmetry of the tiling on the right.

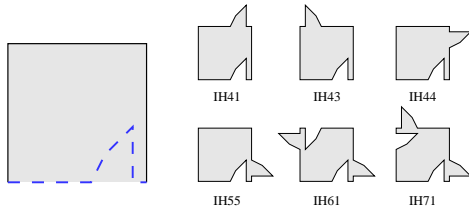


Figure 4 An isohedral tiling type imposes a set of adjacency constraints on the tiling edges of a tile. When the bottom edge of the square deforms into the dashed line, the other edges must respond in some way to preserve the tiling. The six resulting tiles here are from six different isohedral types.

rigid motions. The set of symmetries of a figure is therefore called the *symmetry group* of that figure. If the symmetry group of a figure contains linearly independent translations, we call that figure *periodic*.

For two congruent tiles A and B in a tiling, there will be some rigid motion of the plane that carries one onto the other (there may in fact be several). A somewhat special case occurs when the rigid motion is also a symmetry of the tiling. In this case, when A and B are brought into correspondence, the rest of the tiling will map onto itself as well. We then say that A and B are *transitively equivalent*.

Transitive equivalence is an equivalence relation that partitions the tiles into *transitivity classes*. When a tiling has only one transitivity class, we call the tiling *isohedral*. More generally, a k -isohedral tiling has k transitivity classes. An isohedral tiling is one in which a single prototile can cover the entire plane through repeated application of rigid motions from the tiling's symmetry group. Note that an isohedral tiling must be monohedral, though the converse is not true [11, p. 31], as Figure 3 illustrates.

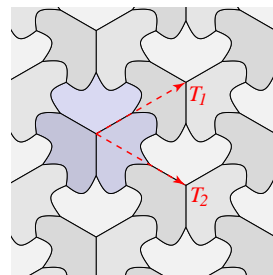
We use the isohedral tilings as a mathematical basis for our exploration of computer generation of ornamental tilings. They achieve a satisfying balance between flexibility and convenience. On the one hand, they are capable of representing a wide subjective range of tilings. Of all the monohedral tilings in Escher's notebook, only one is not isohedral (the exception is based on a special mono- but not isohedral tiling first shown to Escher by Roger Penrose). Moreover, in Escher's dihedral tilings, tiles of each of the two motifs can be paired up to form a single "supertile" that tiles that plane isohedrally. On the other hand, the isohedral tilings can be classified into a small number of symbolically-encoded families (the following subsections give more details about this classification). It is therefore fairly easy to create a system to manipulate and render them.

2.3 Isohedral families

By definition, an isohedral tiling is bound by a set of geometric constraints: congruences between tiles must be symmetries of the

tiling. Grünbaum and Shephard showed that those geometric constraints can be equated with a set of *combinatoric* constraints expressing the adjacency relationship between edges of a tile. They proved that these constraints yield a division of the isohedral tilings into precisely 93 distinct *types* or *families*,¹ referred to individually as IH1, . . . , IH93 and collectively as IH [11, sec. 6.2]. Each family encodes information about how a tile's shape is constrained by the adjacencies it is forced to maintain with its neighbours. A deformation in a tiling edge is counterbalanced by deformations in other edges; which edges respond and in what way is dependent on the tiling type, as shown in Figure 4.

Isohedral tilings have the property that if you list the valence of each tiling vertex as you move around any given tile, the list will be consistent across all tiles in the tiling. This list is fundamental to the topological structure of the tiling and is called its *topological type*.



For example, the topological type of IH16, shown on the left, is 3^6 , since there are 6 different tiling vertices around each tile, each of valence 3. Every isohedral tiling belongs to one of eleven different topological types [11, sec. 2.7].

In any periodic tiling, it is possible to identify a collection of tiles that together cover the plane using only the translations from the tiling's symmetry group. Any such collection that is connected

and minimal in size is called a *translational unit* of the tiling. Within a translational unit, all tiles must have different orientations, which are referred to as the *aspects* of the tiling. IH16 has three aspects, shown in varying shades of blue above. These three tiles comprise one possible translational unit, with translation vectors T_1 and T_2 .

2.4 Incidence symbols

The adjacency constraints between the tiling edges of a tile are summarized by an *incidence symbol*. Given a rendering of a tiling, the incidence symbol can be constructed in a straightforward way.

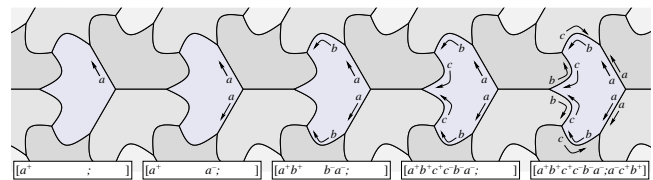


Figure 5 Five steps in the derivation of a tiling's incidence symbol.

Figure 5 shows five steps in the derivation of the incidence symbol for our sample tiling. To obtain the first part of the incidence symbol, pick an arbitrary tiling edge as a starting point, assign that edge a single-letter name, and draw an arrow pointing counterclockwise around the tile (step 1). We then copy the edge's label to all other edges of the tile related to it through a symmetry of the tiling (step 2). Should the edge get mapped to itself with a reversal of direction, it is given a double-headed arrow and becomes undirected. We then proceed counterclockwise around the tile to the next unlabeled edge (if there is one) and repeat the process (step 3). The first half of the symbol is obtained by reading off the assigned edge

¹In tiling theory, seemingly arbitrary numbers like 93 are not uncommon; enumerations of families of tilings tend to have sets of constraints that collapse certain cases and fracture others.

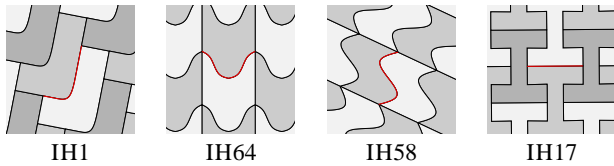


Figure 6 Examples (from left to right) of **J**, **U**, **S** and **I** edges. In each case, the tiling edge with the given shape is highlighted in red.

names (step 4). A directed edge is superscripted with a sign indicating the coherence of its arrow with the traversal direction. Here, a plus sign is used for a counterclockwise arrow and a minus sign for a clockwise arrow.

The second half of an incidence symbol records how, for each different name, a tiling edge of that name is related to the corresponding edge of the tile adjacent to it. To derive this part of the symbol, we copy the labeling of the tile to its neighbours (step 5). Then, for each unique edge letter assigned in the first step, we write down the edge letter adjacent to it in the tiling. If the original edge was directed, we also write down a plus or minus sign, depending on whether edge direction is respectively preserved or reversed across the edge. A minus sign is used if the arrows on the two sides of an edge are pointing in the same direction and a plus sign is used otherwise. For the running example, the incidence symbol turns out to be $[a^+b^+c^+c^-b^-a^-; a^-c^+b^+]$. Note that the incidence symbol is not unique; edges can be renamed and a different starting point can be chosen. But it can easily be checked whether two incidence symbols refer to the same isohedral type.

Every isohedral type is fully described in terms of a topological type and an incidence symbol. Enumerating all possible topological types and incidence symbols and then eliminating the ones that do not result in valid tilings or that are trivial renamings of other symbols leads to the classification given by Grünbaum and Shephard.

2.5 Tile shapes

Within a single isohedral type, tilings are distinguished from each other by their shapes, consisting of the positions of the tiling vertices and the shapes of the curves that join them. In the next section, we will address the question of finding, for each isohedral type, a parameterization of the tiling vertices that yields all and only those tiling polygons compatible with the type. To our knowledge, this problem has not been previously explored.

On the other hand, the constraints on the shapes of tiling edges are simple to describe. Although the underlying choice of how to represent a “curve” is left open, the tiling’s symmetries imply a large reduction in the tiling edges’ degrees of freedom. These constraints can be extracted directly from the tiling’s incidence symbol. We enumerate four cases for the structure of a tiling edge. For each case, Figure 6 gives a tiling with such an edge.

If some directed edge is adjacent to itself with 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 an edge other than itself, it is free to take on any curve with this mirror 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 specify one tiling edge, since the other is entirely constrained to it. Thus, 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 .

3 Parameterizing the isohedral tilings

Like the shape vertices, tiling vertices cannot move 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 to merely 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. To our knowledge, no tiling vertex parameterizations have ever been given for IH. They represent a nontrivial extension to the table of information about IH found in Grünbaum and Shephard.

We have developed a set of consistent and complete parameterizations for the isohedral types (of course, the history of tiling theory has experienced its share of imperfect analyses [11, Sec. 6.6]). They 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 Figures 9 and 10.

To give the flavor of these parameterizations, here is a sketch of the derivation for our running example, IH16 (see Figure 7). 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, $\triangle FAB$, $\triangle BCD$, 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

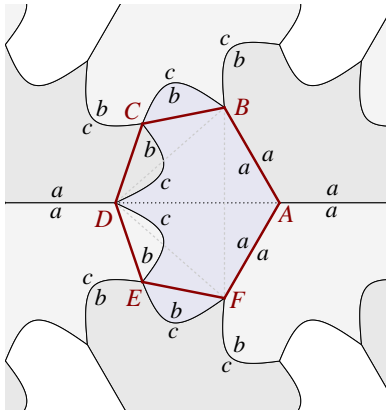


Figure 7 The diagram used to establish a tiling vertex parameterization for IH16. For simplicity, the arrows indicating edge direction have been left out of the diagram.

tiling polygon are the lengths of \overline{AD} and \overline{BF} . However, as discussed in the next section, the shape comparison metric that we would like to use is independent of scale. We can factor out the dependence on scale by fixing $\|\overline{BF}\| = 1$ and keeping just a single parameter: $v_0 \equiv \|\overline{AD}\|$. Figure 8 shows tilings of type IH16 that can result from different values of this single parameter.

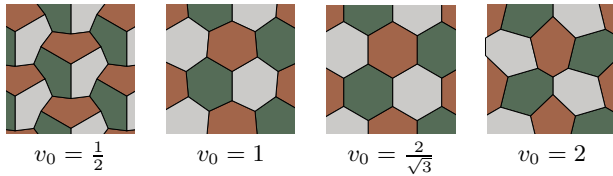


Figure 8 Some examples of IH16 with different values for the single parameter in its tiling vertex parameterization.

4 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. To simplify the rest of the Escherization algorithm, we would also like the metric to be insensitive to rigid motion or scale of either of the shapes.

Fortunately, such metrics have been developed by computer vision researchers. We use the metric created by Arkin *et al.* for comparing polygons [2]. 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. Turning functions are naturally translation and scale independent. Translation of a turning function corresponds to rotation of the polygon and movement of the point where the measurement of arc length begins. They compute the minimal L^2 distance between all translations of the two turning functions by proving that only a small number of such translations need to be checked.

Their algorithm is efficient and has a predictable run time: $O(n^2 \log n)$ in the total number of vertices n . The algorithm also corresponds fairly well to a subjective notion of the distance between two shapes. It is limited in its ability to cope with varying levels of detail across the shapes (which is a form of what they call “non-uniform noise”), but it is acceptable for our purposes.

We use the polygon comparison metric for both polygons and subdivision curves. In the case of subdivision curves, we first approximate the curve as a polygon with a large number of vertices and then make a call to the same routine.

5 Optimizing over the space of tilings

Armed with a set of tilings (the isohedral tiles), 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 tiles are close to the goal shape.

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

```

function FINDOPTIMALTILING(GOALSHAPE, FAMILIES):
  INSTANCES  $\leftarrow$  CREATEINSTANCES(FAMILIES)
  while  $\|\text{INSTANCES}\| > 1$  do
    for each  $i$  in INSTANCES do
      ANNEAL( $i$ , GOALSHAPE)
    end for
    INSTANCES  $\leftarrow$  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 instances of tilings from each isohedral family. It 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 tiling instance is left. This tiling is returned as the optimal tiling.

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

```

procedure ANNEAL(TILING, GOALSHAPE):
  for  $j = 1$  to  $N$  do
    while  $T > T_{\min}$  do
      OPTIMIZETILING(TILING, GOALSHAPE,  $T$ )
       $T \leftarrow$  REDUCE( $T$ )
    end while
    SMOOTHEDGESHAPES(TILING)
    SPLITEDGESHAPES(TILING)
    ( $T$ ,  $T_{\min}$ )  $\leftarrow$  UPDATESCHEDULE( $T$ ,  $T_{\min}$ )
  end for
  suspend
end procedure

```

The annealer takes a given tiling instance and a goal shape as input. It loops for a constant number of iterations to improve the tiling and then exits, maintaining its state, so that upon re-entry it can continue from where it left off, in the same cooling schedule. Within each iteration of the outer loop, the procedure takes a number of cooling steps, reducing the “temperature” at each step. Within this inner loop, it makes a call to a procedure that we have termed OPTIMIZETILING(). This procedure implements the “multidimensional minimization by simulated annealing combined with the downhill simplex method,” as described by Press *et al.* [17]. The procedure attempts to improve all of the parameters of the tiling, including the parameterizations of the tiling vertices (discussed in Section 3) and the positions of the shape vertices of the tile. The procedure always accepts a downhill step (one that improves the tiling instance) and sometime accepts an uphill step, with probability depending on the temperature T . Once the temperature has

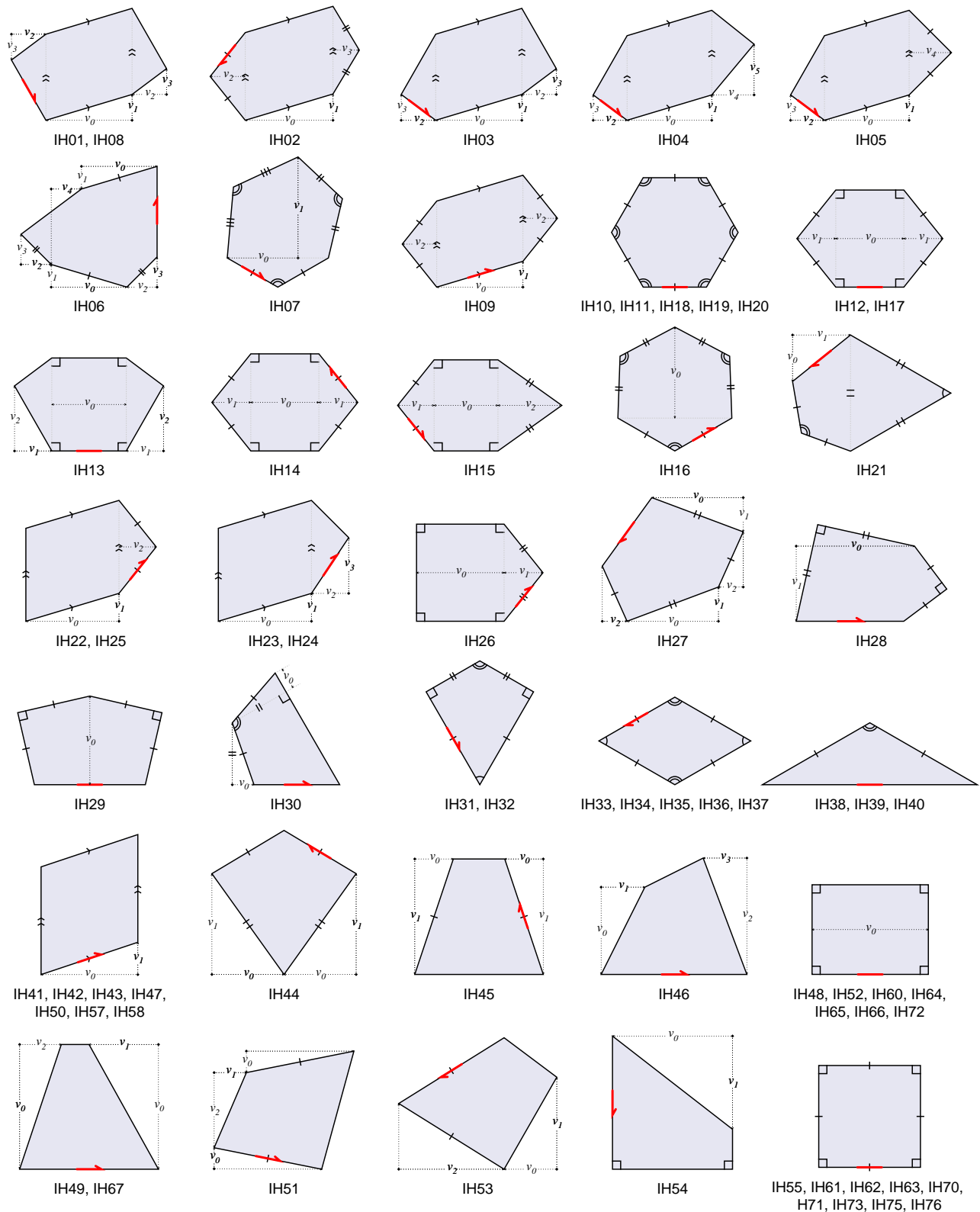


Figure 9 The complete set of tiling vertex parameterizations for the isohedral tilings, part one. In each tile, the edge marked with a red line is the first edge in the tiling type's incidence symbol. When that first edge is directed, the red line has an arrowhead. Labelled dotted lines represent parameter 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.

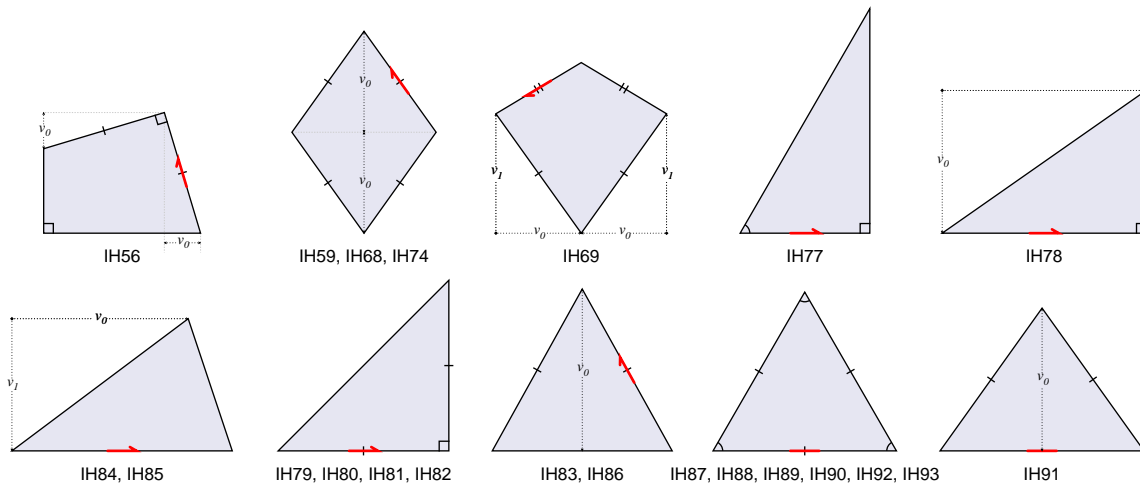


Figure 10 The complete set of tiling vertex parameterizations for the isohedral tilings, part two.

cooled to some minimum temperature T_{\min} , we exit this inner loop. At this stage, we run through the vertices of the tile and remove any vertices that are nearly collinear with their neighbors, thereby eliminating any unnecessary degrees of freedom that may have been introduced into the tiling instance. Next, we subdivide each edge of the tiling, essentially doubling the number of variables over which we optimize in the next stage. Finally, we restart the cooling schedule, generally with some slightly lower temperatures 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 B-spline subdivision curves after a certain stage in the optimization. We can then additionally optimize over weights on each vertex that control the smoothness of the curve near that point.

Our 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 on the initial shape of the tiling polygon and the initial positions of the shape vertices. We therefore generally start with multiple instances for each tiling type. As with any simulating annealing algorithm, the choice of cooling schedule can also make a difference. We 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. We did not spend a lot of time “optimizing” this cooling schedule, so other reasonable choices would probably work equally well or better.

6 Representation of isohedral tilings

We have developed a computer representation of isohedral tilings that allows us to express our Escherization algorithm efficiently and naturally. The key is to factor out the constraints on the tile imposed by adjacencies and internal symmetries, and to store only the minimal set of free parameters that encode the tile shape.

We break down the information associated with a tile into two components: the *tiling template* and the *tile instance*. The tiling template contains information about a tiling type in general. The tile instance refers to a template and contains a set of parameters for the tiling vertex parameterization, along with the minimal set of information required to reproduce the edge shapes. We first describe each of

```

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

```

Figure 11 The tiling type information stored for IH16

these components in detail, and then show how they can be used to support efficient editing and viewing in an interactive system.

6.1 Tiling templates

Tiling templates are computed once ahead of time, and stored in a master file that is read in when the tile library is initialized. Figure 11 shows a sample entry from the template file. The complete set of templates is available on the proceedings CD-ROM.

Along with the topological type and incidence symbol (lines 1 and 2), we store additional static information that increases the efficiency and functionality of our system.

First, we add a `colouring` field (line 3) that provides a default rule for filling the interiors of tiles with colours. An n -colouring of a tiling is a set of symbols $\{c_1, \dots, c_n\}$, together with a function f that assigns a colour c_i to each tile in the tiling. A *perfect colouring* is a colouring that respects the tiling’s symmetry in the sense that symmetries act as permutations of the colours. Every perfect colouring of an isohedral tiling can be conveniently encoded as an assignment of different colours to the different aspects in a single translational unit, along with an assignment of different colour permutations to each of the two translation vectors. The colouring field in the template gives, in order, the number of colours, the assignment of colours to aspects, and the permutations of the assignment associated with the two translation vectors. This encoding can express a superset of the perfect colourings. In the case shown here, the permutations are both the identity. (In all of his drawings, Escher was careful to ensure that no two adjacent tiles ever shared the same colour. He also used the minimum number of colours necessary to satisfy this condition. The default colourings we provide in our tiling templates have both of these properties.)

Another line of the template (line 4) specifies the number of aspects in the tiling, in this case, 3.

The `rules` section (lines 5 through 9) gives a collection of rules that, when applied to a tiling polygon, yield transform matrices for all the aspects of a translational unit, as well as the two translation vectors. These transforms cannot be computed ahead of time, as they depend on the tiling polygon. Each rule is expressed as a sequence of hops across edges, starting from the first aspect in the translational unit at the origin.

Aspect 1 is always given the identity matrix as its transform, and the other aspect transforms are computed from it. In this example, the first rule (line 6) says that the transform for creating aspect 2 from the first aspect is just the transform that creates the symmetry across edge 1 of the first aspect in the tiling—that is, a reflection about the first edge, labelled `a+`, in the incidence symbol. Similarly, the second rule (line 7) says that the transform for creating aspect 3 from the first aspect is the transform that creates the symmetry across edge 6 of the first aspect in the tiling—here, a reflection about the edge labelled `a-`. Sometimes, more than a single hop is required. For instance, the rule “`aspect 2 1, 2, 3`” would specify a sequence of hops: first, across edge 1 of the first aspect in the tiling, then across edge 2 of the first aspect’s neighboring tile, then across edge 3 of that neighbor’s neighbor.

The two translation vectors are specified in the same way. Thus, following across edge 1 of the first aspect in the tiling, then across edge 4 of the first aspect’s neighboring tile, gives the translation vector T_1 .

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 that implements the parameterizations is available on the CD-ROM.

6.2 Tile instances

The tile is stored as a set of parameters for the tiling vertex parameterization, along with a hierarchical model whose leaves are *fundamental edge shapes*—the portions of the tiling edges that cannot be further decomposed by symmetries.

The fundamental edge shapes are simply stored as arrays of points. Each fundamental edge shape implicitly begins at $(0, 0)$ and ends 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.

To rebuild the tile shape, we apply the parameterization to obtain the positions of the tiling vertices, and transform the edge shapes into place between them.

There are at most three levels of transformation 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 name in the incidence symbol share a lower-level shape object.

Specific tiles are stored in tile files, which are simply XML documents.

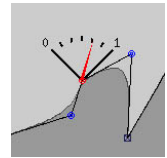
6.3 Interactive tools

To provide a convenient interface to the Escherization algorithm, and to explore the mathematical and aesthetic properties of isohedral tilings in general, we have constructed several graphical tools on top of the tile library and optimizer, using the free toolkits **GTK+** [19] and **GTK--** [12].

The simplest of these tools is a utility for tracing goal shapes from images. An image can be loaded into a viewer where the user can trace an outline of an image by hand. The outline can then be saved and passed to the optimizer.

The more sophisticated tool is a rich viewer and editor for tile files. The editor is highly responsive, running at interactive rates on an off-the-shelf Linux system with no graphics acceleration. 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-based edges are 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.



6.4 Filling a region with tiles

The most basic drawing operation for a tiling is to fill a region of the plane with copies of the tile. Beginning with a tile in its local coordinate system and a viewing region, we need to find the rigid motions to apply to the tile that replicate it across the region.

To find these motions, we project the viewing region’s corners into the coordinate system formed by the tiling’s translation vectors, derived from the template’s rules. In that coordinate system, the translational units become lattice squares; the lattice squares that intersect the projection of the viewing region are the ones that need to be drawn. For each needed translation, we place a tile relative to the rigid motion formed by composing the translation with each of the aspect transforms in turn.

7 Decorations and rendering

The output of the core Escherization algorithm is a geometric description of a tile, not a finished ornamental design. To complete the Escherization process, we need to surround the core algorithm with tools to add decorations to tiles and create high-quality renderings of the results. We have explored the use of both vector-based and image-based decorations and rendering styles.

A tile maintains a set of *markings*, sequences of weighted subdivision control points with various drawing attributes. Markings can be open, closed or filled, polygonal or subdivided, and have variable line thickness, line colour and fill colour. The line and fill colours

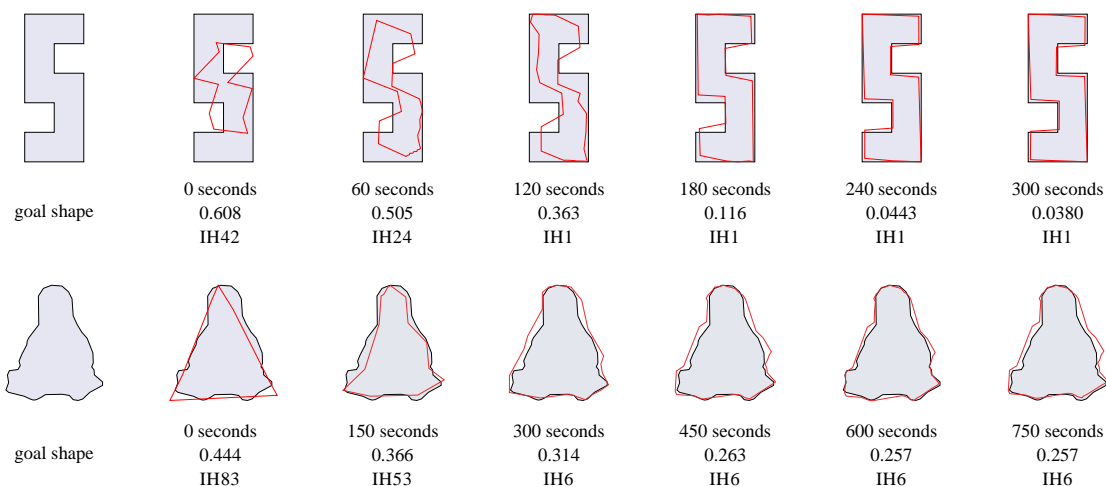


Figure 12 Timelines for two sample Escherization runs. Each step shows the current best tile in the system (in red) overlaid on the goal shape. The caption indicates the elapsed time, the score for that tile, and its isohedral type. The second goal shape is the penguin from Figure 15(c).

can also be mapped according to the tiling’s colouring. The markings can be created and edited from within the interactive tool, and are stored in the tile file as fragments of XML. The editor can also render a tiling decorated with markings as PostScript.

It may also be desirable to fill the interior of a tile with image-based markings. We have implemented an image-based tiling renderer using **libart**, a freely-available image manipulation library [16]. The renderer takes a tile file and a set of images to serve as backdrops. For each tile in a region, it starts with the image backdrop for that tile’s colour, applies a transparent wash of the tile colour, rasterizes the markings, draws an outline, and transforms the composited tile into its position in the final rendering.

The natural choice for an image-based marking is the interior of the goal shape in the image that was originally traced. Using the correspondence provided by the polygon comparison metric, we do a Beier-Neely style image warp [3] 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, we end up with 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.

To further increase the appeal of an image-based rendering, we apply various painterly effects to the warped tile image before replication. This post-processing step gives the artist creative control over the appearance of the final tiling, and can bring the result closer to the informal hand-drawn style of Escher’s notebook drawings.

8 Results

We have used our Escherization implementation and decoration tools to produce a number of ornamental tilings from various sources of imagery.

Figure 12 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 shape is a more typical outline traced from an image. 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).

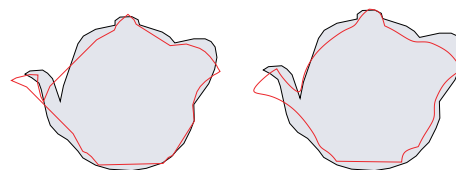


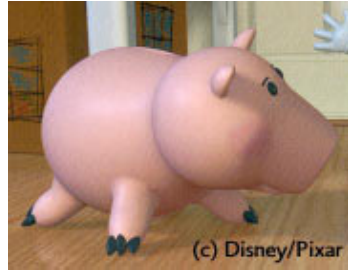
Figure 13 A comparison between the tile returned by the optimizer and the same tile with user modifications. Note also that the second tile has subdivision enabled.

Figure 13 shows the tile result produced by the optimizer for a teapot image, followed by the tile after a small amount of hand-tweaking in the interactive editor. Even when manual intervention improved the overall appeal of a tiling, Escherization did the hard work of determining how to make the goal shape fit together with itself in the first place. The edits shown here took a minute or two to perform and were fairly typical of our experience in creating tilings in this fashion.

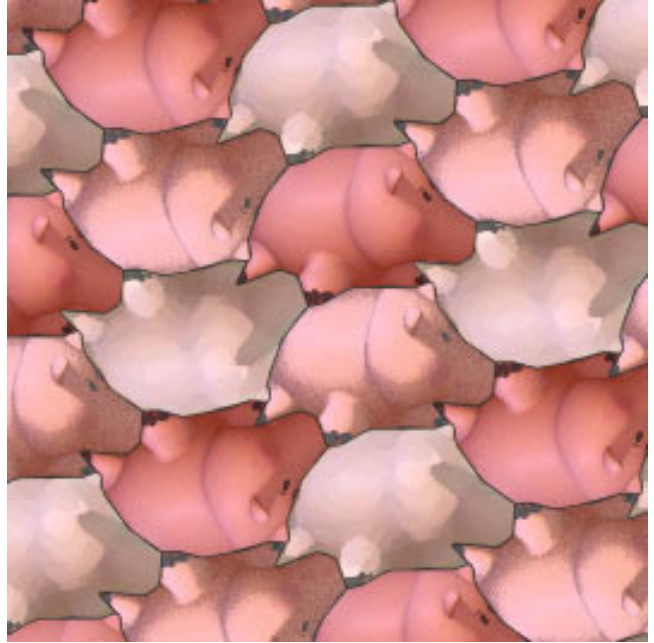
The remaining results can be seen in Figures 14 and 15. Figures 15 (d) is rendered as line art, and the remaining examples use the image-based renderer. In all cases, the optimizer generated a tile shape that was then modified slightly in the editor. The source image was warped into the tile shape, and copies of the warped image were recoloured and edited to make the final rendering. The user intervention was primarily to exert creative control, and rarely to guide the optimization process. On some occasions, it was helpful to watch the optimizer discover a tiling type suitable for a given goal shape, then stop and restart it with many tilings of that type, resulting in a narrower and deeper search.

9 Discussion

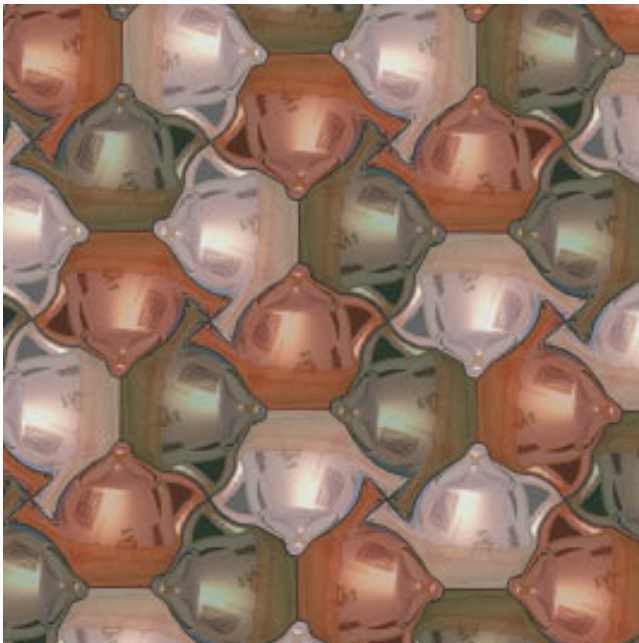
Most outlines are not tiles. For just about any goal shape, an Escherizer will have to produce an approximation, and a better Escherizer will produce a closer approximation. A perfect Escherizer would determine the smallest distance over all possible tile shapes, and return the tiling that achieves that 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 16 that our algorithm cannot successfully Escherize.



(a) *Dogs; Dogs Everywhere* (IH4)



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

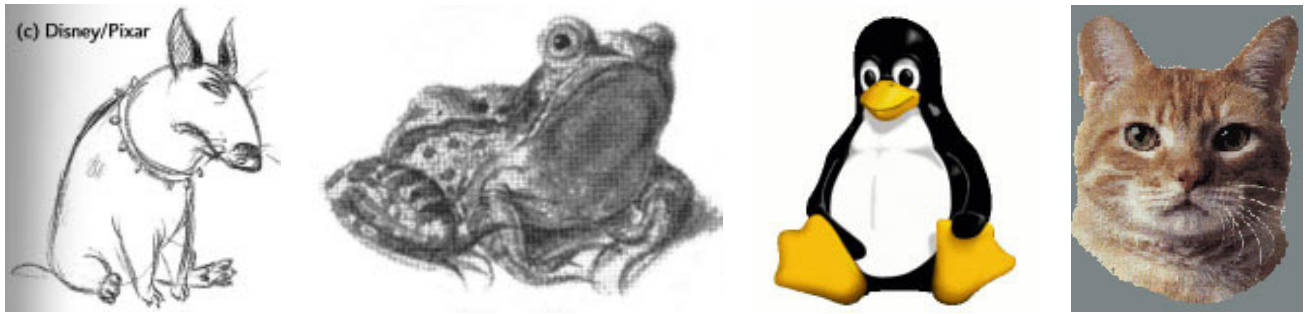


(c) *Tea-sselation* (IH28)

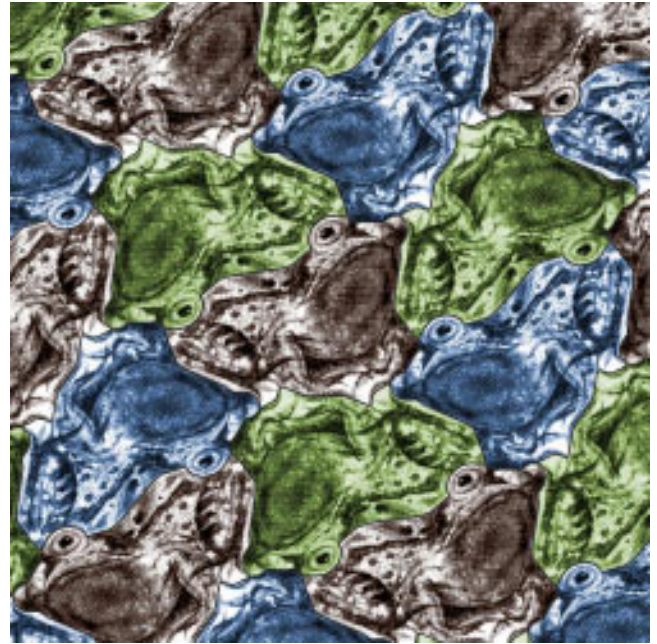


(d) *Twisted Sisters* (IH86)

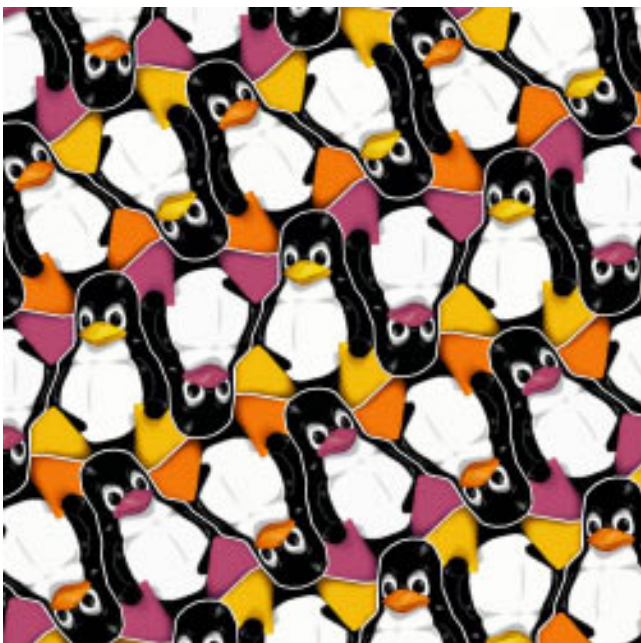
Figure 14 Some examples of Escherized images and the tilings they generate. Hamm the pig appears courtesy of Disney/Pixar.



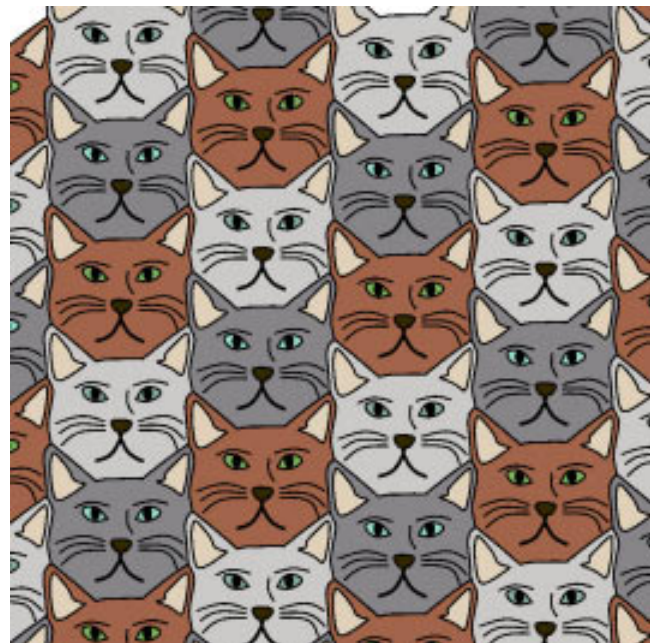
(a) *Sketchy Dogs (IH6)*



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



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



(d) *Bubbles the Cat (IH1)*

Figure 15 More examples of Escherization. Tux the Penguin appears courtesy of Larry Ewing (lewing@isc.tamu.edu). Sketchy Dog appears courtesy of Disney/Pixar.

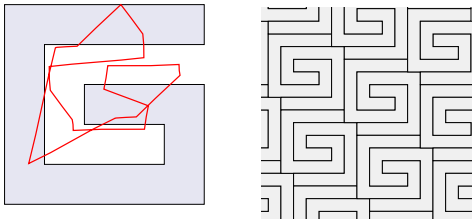


Figure 16 A tile for which Escherization performs badly, and a tiling that can be generated from it.

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. Furthermore, in our shape comparison metric, the importance of a section of outline is directly proportional to its fraction of the perimeter of the goal shape, even if from our own perspective 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 to us 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.

Moreover, 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. Our optimizer 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.

All this being said, the Escherizer we have built performs remarkably well on many different shapes for which no tiling is obvious. Who would have guessed that a teapot could tile the plane? We certainly couldn’t. Even when the optimizer fails to find an ideal tiling, it often finds a tiling that is close enough that it is easily converted into an acceptable result. Thus, it allows us to work in much the same way that Escher did, only with a very close starting point and more helpful interactive tools.

This research suggests many future directions, including generalizing our algorithms to handle multihedral and aperiodic tilings, parquet deformations [13, Chap. 10], or tilings over non-Euclidean domains, such as the hyperbolic plane [7]. Another intriguing idea is to allow some flexibility in the goal shape as well. For instance, instead of a 2D shape, we might use a 3D (and potentially parameterized) model and attempt to automatically discover a camera position from which the view of the model is most easily Escherized. Finally, along the lines of creating Escher tilings automatically is the problem of “automatic conventionalization”: somehow creating not just the tile boundaries, but the line-art graphical decorations that go inside the tilings, more or less automatically from a reference image.

Acknowledgments

Many people contributed to the development of this research. Michael Noth and Jeremy Buhler collaborated on FuTile, the class project that ultimately led to our ongoing research in tilings. Branko Grünbaum motivated the use of isohedral tilings as a powerful

and manageable system for ornamental design. Douglas Zongker sat in on many early discussions. Michael Cohen, Rick Szeliski and John Hughes participated in discussions and provided valuable feedback and insight that kept the project moving forward. Mike Ernst pointed us at the papers that led us to the polygon comparison metric. Dan Huttenlocher assisted us with the metric (and made his source code available). Tony DeRose filled in details on the implementation of subdivision curves. Zoran Popović helped with the taming of the continuous simulated annealing algorithm. Finally, Victor Ostromoukhov provided helpful feedback on a draft of this paper. This research was supported in part through industrial grants from Intel, Microsoft and Pixar.

References

- [1] Howard Alexander. The computer/plotter and the 17 ornamental design types. *Proceedings of SIGGRAPH’75*, pages 160–167, 1975.
- [2] E. M. Arkin, L. P. Chew, D. P. Huttenlocher, K. Kedem, and J. S. B. Mitchell. An efficiently computable metric for comparing polygonal shapes. *PAMI(13)*, pages 209–216, 1991.
- [3] Thaddeus Beier and Shawn Neely. Feature-based image metamorphosis. *Proceedings of SIGGRAPH’92*, pages 35–42, 1992.
- [4] William W. Chow. Automatic generation of interlocking shapes. *Computer Graphics and Image Processing*, 9:333–353, 1979.
- [5] William W. Chow. Interlocking shapes in art and engineering. *Computer Aided Design*, 12:29–34, 1980.
- [6] Andrew Crompton. Grottesque geometry. <http://dspace.dial.pipex.com/crompton/Home.shtml>.
- [7] Douglas J. Dunham. Creating hyperbolic escher patterns. In H.S.M. Coxeter et al., editor, *M.C. Escher: Art and Science*, pages 241–247. Elsevier Science Publishers B.V., 1986.
- [8] Andrew Glassner. Frieze groups. *IEEE Computer Graphics and Applications*, 16(3):78–83, May 1996.
- [9] Andrew Glassner. Andrew glassner’s notebook: Aperiodic tiling. *IEEE Computer Graphics & Applications*, 18(3):83–90, May – June 1998. ISSN 0272-1716.
- [10] Andrew Glassner. Andrew glassner’s notebook: Penrose tiling. *IEEE Computer Graphics & Applications*, 18(4), July – August 1998. ISSN 0272-1716.
- [11] Branko Grünbaum and G. C. Shephard. *Tilings and Patterns*. W. H. Freeman, 1987.
- [12] GTK-. <http://gtkmm.sourceforge.net>.
- [13] Douglas Hofstadter. *Magical Themes: Questing for the Essence of Mind and Pattern*. Bantam Books, 1986.
- [14] Daniel H. Huson and Olaf Delgado Friedrichs. Reptiles. <ftp://ftp.uni-bielefeld.de/pub/math/tiling/reptiles/>.
- [15] Hans Van Lemmen. *Tiles: 1000 Years of Architectural Decoration*. Harry N. Abrams, Inc., 1993.
- [16] Raph Levien. <http://www.levien.com/libart/>.
- [17] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. Numerical recipes in c: The art of scientific computing (2nd ed.). 1992. ISBN 0-521-43108-5. Held in Cambridge.
- [18] Doris Schattschneider. *M.C. Escher: Visions of Symmetry*. W.H. Freeman, 1990.
- [19] The GIMP toolkit. <http://www.gtk.org>.
- [20] M.C. Escher (tran. Karin Ford). *Escher on Escher: Exploring the Infinite*. Henry N. Abrams, Inc., 1989.
- [21] Michael T. Wong, Douglas E. Zongker, and David H. Salesin. Computer-generated floral ornament. *Proceedings of SIGGRAPH’98*, pages 423–434, 1998.