

Snakes in the Plane

by

Paul Church

A thesis

presented to the University of Waterloo

in fulfillment of the

thesis requirement for the degree of

Master of Mathematics

in

Computer Science

Waterloo, Ontario, Canada, 2008

© 2008 Paul Church

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Recent developments in tiling theory, primarily in the study of anisohedral shapes, have been the product of exhaustive computer searches through various classes of polygons. I present a brief background of tiling theory and past work, with particular emphasis on isohedral numbers, aperiodicity, Heesch numbers, criteria to characterize isohedral tilings, and various details that have arisen in past computer searches.

I then develop and implement a new “boundary-based” technique, characterizing shapes as a sequence of characters representing unit length steps taken from a finite language of directions, to replace the “area-based” approaches of past work, which treated the Euclidean plane as a regular lattice of cells manipulated like a bitmap. The new technique allows me to reproduce and verify past results on polyforms (edge-to-edge assemblies of unit squares, regular hexagons, or equilateral triangles) and then generalize to a new class of shapes dubbed *polysnakes*, which past approaches could not describe. My implementation enumerates polyforms using Redelmeier’s recursive generation algorithm, and enumerates polysnakes using a novel approach. The shapes produced by the enumeration are subjected to tests to either determine their isohedral number or prove they are non-tiling.

My results include the description of this novel approach to testing tiling properties, a correction to previous descriptions of the criteria for characterizing isohedral tilings, the verification of some previous results on polyforms, and the discovery of two new 4-anisohedral polysnakes.

Acknowledgements

I would like to thank my thesis supervisor, Craig Kaplan, for inspiring me to pursue this research area and providing extensive advice, feedback, and support. I would also like to thank Joseph Myers, both for providing the footsteps that this work follows in, and for some very helpful suggestions along the way.

Dedication

This thesis is dedicated to Samuel L. Jackson, a fellow expert on snakes in the plane.

Contents

1	Introduction	1
2	Background	4
2.1	Introduction to Tiling Theory	4
2.1.1	Unbalanced Tilings	14
2.1.2	Aperiodic Tilings	17
2.1.3	Heesch’s Problem	21
2.1.4	Isohedral Tiling Criteria	25
2.2	Polyforms	29
2.2.1	Sufficiency of Edge-to-Edge Tiling	30
3	Thesis Summary	34
3.1	Boundary Words	35
3.2	Boundary-Based Tiling Criteria	38
4	Implementation: Enumeration	40
4.1	Enumeration of Polyforms	40
4.2	Redelmeier’s Algorithm	41
4.2.1	Implementation	44

4.3	Parallelization	45
4.4	Conversion to Boundary Word Representation	46
4.5	Canonicalization	46
4.6	Hole Detection	47
4.7	Enumeration of Polysnakes	47
5	Implementation: Tiling Tests	49
5.1	Isohedral Tiling	50
5.2	Anisohedral and Non-tiling	53
5.2.1	Basic Approach	53
5.2.2	Boundary Merge Implementation	56
5.2.3	2-tile Patch Construction	57
5.2.4	Surround Depth-First Search Implementation	57
5.2.5	Optimizations	58
6	Results	61
7	Conclusions and Future Work	70

List of Tables

6.1	Tiling results for polyominoes.	64
6.2	Isohedral numbers for polyominoes.	64
6.3	Tiling results for polyhexes.	65
6.4	Isohedral numbers for polyhexes.	65
6.5	Tiling results for polyiamonds.	66
6.6	Isohedral numbers for polyiamonds.	66
6.7	Tiling results for 4-snakes.	67
6.8	Isohedral numbers for 4-snakes.	67
6.9	Tiling results for 5-snakes.	68
6.10	Isohedral numbers for 5-snakes.	68

List of Figures

2.1	Part of a Voderberg spiral tiling [11, Section 9.5].	7
2.2	A monohedral, non-periodic, non-edge-to-edge tiling.	8
2.3	A monohedral, non-periodic, edge-to-edge tiling with rotational symmetry.	8
2.4	An isohedral tiling.	9
2.5	A 2-isohedral tiling.	9
2.6	A 2-isohedral tiling by a 2-anisohedral shape.	10
2.7	A 3-isohedral tiling by Stein’s 3-anisohedral pentagon, transitivity classes indicated by shading [28].	11
2.8	A 4-isohedral tiling by a 4-anisohedral polyiamond, transitivity classes indicated by shading [4].	11
2.9	A 10-isohedral tiling by a 10-anisohedral polyhex, one representative of each transitivity class shaded [21].	13
2.10	An unbalanced 2-anisohedral 8-hex.	15
2.11	An unbalanced 2-anisohedral 18-omino.	16
2.12	An unbalanced 2-anisohedral 20-omino.	16
2.13	An unbalanced 2-anisohedral 20-omino.	16
2.14	An unbalanced 2-anisohedral 22-omino.	16
2.15	Robinson’s R1 set of aperiodic tiles.	18

2.16	Penrose's kite and dart aperiodic set.	19
2.17	The aperiodic Penrose rhombs.	19
2.18	Penrose rhombs with geometric matching conditions.	20
2.19	Part of a tiling by the Penrose rhombs.	20
2.20	A tile with Heesch number 0.	22
2.21	A tile with Heesch number 1.	22
2.22	A tile with Heesch number 5. (Image courtesy of Casey Mann.)	24
2.23	The 9 boundary criteria for isohedral tiling.	26
2.24	A non-tiling polyomino that passes the weakened version of criterion 5.	28
2.25	A non-tiling polyomino that passes the weakened version of criterion 6.	29
2.26	Sample polyomino, polyhex, and polyiamond.	30
2.27	Proof that all tilings by polyhexes are edge-to-edge.	31
2.28	Proof of faultline existence.	32
3.1	Polyform boundary language directions.	36
3.2	A 6-hex with boundary word 050501232321210123454545.	36
3.3	A 5-snake with boundary word 004125866.	37
4.1	Lattice cell legend.	41
4.2	Failure of uniqueness under translation.	42
4.3	Addition of BLOCKED cells.	43
4.4	Multiple orders of generation.	43
4.5	Finished algorithm execution up to 3-ominoes, and the 2x2 4-omino.	44
5.1	6-hex problem case.	56

5.2	2-patch data structures.	58
6.1	The 4-anisohedral 4-snake represented by 013124725065, shown in a 4- isohedral tiling with transitivity classes indicated by shading.	69
6.2	The 4-anisohedral 4-snake represented by 010346347572, shown in a 4- isohedral tiling with transitivity classes indicated by shading.	69

Chapter 1

Introduction

Tiling theory, the study of the ways shapes can fit together to cover the infinite plane, has begun to receive attention in the past century as a topic worthy of mathematical study in addition to its ancient place in art and aesthetics. In recent years, some problems in tiling theory have taken on an experimental direction in response to a lack of progress in proving the limits of some basic tiling properties. The rapid growth of computing power has allowed exhaustive searches through parameterized classes of shapes to supplement the ad hoc intuition of researchers that has produced many shapes of interest in the past.

There are three key problems that experimental approaches have aimed to shed light on. The first, and the one with which this thesis is primarily concerned, is the problem of finding shapes with progressively higher *isohedral number*. Briefly, the isohedral number of a shape is the smallest possible number of transitivity classes of tiles induced by the symmetries of any tiling by that shape. In a manner of speaking, it is a measure of the shape's minimum tiling complexity. Past experimental results have pushed the highest known isohedral number from 4 to 10. The second, which serves as a secondary objective for this thesis, is the problem of finding a single *aperiodic* shape. A set of shapes is called aperiodic if it tiles the plane but only admits tilings that are non-periodic, that is, having no finite region that repeats by translation to cover the entire plane. The smallest known aperiodic sets are of two shapes, and the question of whether a single aperiodic

shape can exist is one of the major open problems in tiling theory. The third, which this thesis addresses only indirectly, is *Heesch's problem*: finding shapes with progressively higher *Heesch number*. The Heesch number of a shape that does not tile the plane is the maximum number of layers to which the shape can be completely surrounded by copies of itself. It can be thought of as a measure of tiling complexity for non-tiling shapes.

These three problems have a variety of potential consequences and applications. The existence of an aperiodic shape would have implications for the decidability status of many tiling-related problems. A bound on Heesch numbers would imply a “short-cut” to proving that a shape tiles the plane simply by exhibiting a large enough number of surrounding layers, without demonstrating anything about an actual infinite tiling. New types of tilings provide new tools for geometric styles of art. Sets of aperiodic shapes have been used in texture synthesis for computer graphics to fill arbitrary areas with non-repeating arrangements of small fixed building blocks. Properties of planar tilings have been used in the study of physical phenomena such as crystal structure and the layout of organic molecules. It is possible that a discovery in tiling theory could imply the existence of previously-unknown physical structures. At the heart of all three problems is the way that the local properties of a shape—how it fits together with its immediate neighbours—can force global properties in an infinite tiling.

There are a number of reasons why my particular approach is interesting. The correctness of past work that has been undertaken by exhaustive computer search depends almost entirely on a computer program, which is subject to undetected bugs or hardware errors during thousands of hours of execution. By providing an independent reimplementation using significantly different techniques, this research reduces the chance of undetected errors in past results and puts the experimental study of tilings on a more solid footing. The novel approach to testing tiling properties using *boundary words* (a string describing the boundary of a shape as a series of characters from a language of unit-length steps in fixed directions) opens the door to a generalization into new classes of shapes, dubbed *polysnakes*.

Chapter 2 presents an extensive background on tiling theory, past work, and the problems of interest. Chapter 3 explains my new approach to the problem, defines boundary words and polysnakes, and explains how they can be used to test for tiling properties. Chapter 4 covers the enumeration problems of generating a complete list of shapes from a parameterized class of shapes, and the implementation of that enumeration. Chapter 5 describes the implementation of the actual tests for tiling properties, and some of the difficulties and optimizations along the way. The experimental results are summarized in Chapter 6, followed by conclusions and ideas for future work in Chapter 7.

Chapter 2

Background

2.1 Introduction to Tiling Theory

Tiling theory can be most succinctly described as a branch of combinatorial geometry concerned with the ways shapes can fit together to fill a metric space (for our purposes, the Euclidean plane) without gaps or overlap. Although tilings hold a central place in art and ornament dating back thousands of years, the mathematical study of the geometric properties of tilings has received relatively limited attention, with many fundamental questions left unsolved. The definitive work on tiling theory is Grünbaum and Shephard's *Tilings and Patterns* [11], which brought together a century of published and unpublished work into one cohesive presentation. Much of the background material in this chapter, particularly the first section, is paraphrased from Chapter 1 of *Tilings and Patterns*, with the addition of results and concepts that came after its publication.

Our setting is the familiar Euclidean plane. We assume all the usual concepts of distance, area, angle, and congruence are known to the reader.

Definition: A *tiling* T is a countable family of closed sets $\{T_1, T_2, \dots\}$ such that the union of T_1, T_2, \dots is the whole plane, and the interiors of the sets T_i are pairwise disjoint. The sets T_1, T_2, \dots are called the *tiles* of T .

For the purposes at hand this definition is far too general, as it admits many kinds of

tiles that are not “well behaved” shapes. We will restrict our interest to the case where each tile is a *topological disc*, that is, it has a boundary that is a *single simple closed curve*. By this we mean a single curve with no crossings or branches whose ends join up to form a “loop”.

The definition of a tiling ensures that the intersection of any pair of tiles of T containing at least two distinct tiles will have zero area. For the types of tilings we are concerned with, the intersection with either be empty or consist of a set of isolated points and arcs. In these cases the points will be called *vertices* of the tiling and the arcs will be called *edges*. A tiling by polygons will be called *edge-to-edge* if the vertices of the tiling lie only on vertices of the polygons and not anywhere else along the polygon edges. (In some contexts, the term edge-to-edge is used to mean that all polygon vertices are tiling vertices and vice versa, a more restrictive definition. This thesis will exclusively use the looser definition.)

We say two tilings are *equal* if there is a similarity transformation of the plane that maps one of the tilings onto the other. This allows us to disregard variations in scale, orientation, or position and talk about *the* tiling in a purely combinatorial sense.

A *patch* of tiles in a given tiling is a finite collection of tiles of the tiling with the property that their union is a topological disc.

A tiling T is called *monohedral* if every tile in T is congruent (directly or reflectively) to one tile T_* , that is, all the tiles are the same size and shape. The tile T_* is called the *prototile* of T , and we say the prototile T_* *admits* the tiling T . This terminology generalizes to *dihedral* (a tiling by two distinct shapes), *trihedral*, \dots , n -hedral. Almost all of the material in this thesis deals with monohedral tilings by polygons.

To be precise, it should be noted that any figure depicting “a tiling” is of course a finite region of a tiling with an implied continuation in every direction. Figure 2.1 shows part of a spiral tiling, which is a monohedral and edge-to-edge tiling by a polygon. Since each revolution of the spiral is slightly different than the one before it (but according to a fixed pattern), the process of continuing it out to cover the entire plane is not a

cut-and-paste operation.

An *isometry* is any mapping of the Euclidean plane onto itself that preserves distances. It is a well-known fact of Euclidean geometry that every isometry is one of four types:

1. *Rotation* about a point O through a given angle Θ . O is called the *center of rotation*.
The case where $\Theta = \pi$ is sometimes called a *halfturn*.
2. *Translation* in a given direction through a given distance.
3. *Reflection* in a given line L , called the *line of reflection*.
4. *Glide reflection*, which combines reflection in a line L with a translation through a given distance parallel to L .

For an isometry σ and a set S we write σS for the image of S under σ . A *symmetry* of S is an isometry σ that maps S onto itself, that is $\sigma S = S$. The isometry that maps every point onto itself, called the *identity isometry* is a symmetry of every set. If rotation through $2\pi/n$ about a point O is a symmetry of a given set, then we call O a *center of n -fold rotational symmetry*.

For any set T we denote by $S(T)$ the set of symmetries of T . This set forms a *group* under the operation of composition, and the number of symmetries in $S(T)$ is the *order* of the group.

We extend the definition of symmetry to tilings as follows: we say an isometry σ is a *symmetry of a tiling T* if it maps every tile of T onto a tile of T . An informal way of thinking about this concept (but slightly lacking in mathematical validity) is to imagine drawing the tiling on an infinite sheet of paper, and then tracing it onto a transparent sheet. A symmetry corresponds to a rigid motion of the transparent sheet (including the possibility of turning it over) such that, after the motion, the tracing fits exactly over the original drawing.

As before, we can speak of the group of symmetries $S(T)$ for a tiling T . If a tiling admits any symmetry in addition to the identity symmetry, we call it *symmetric*. If the

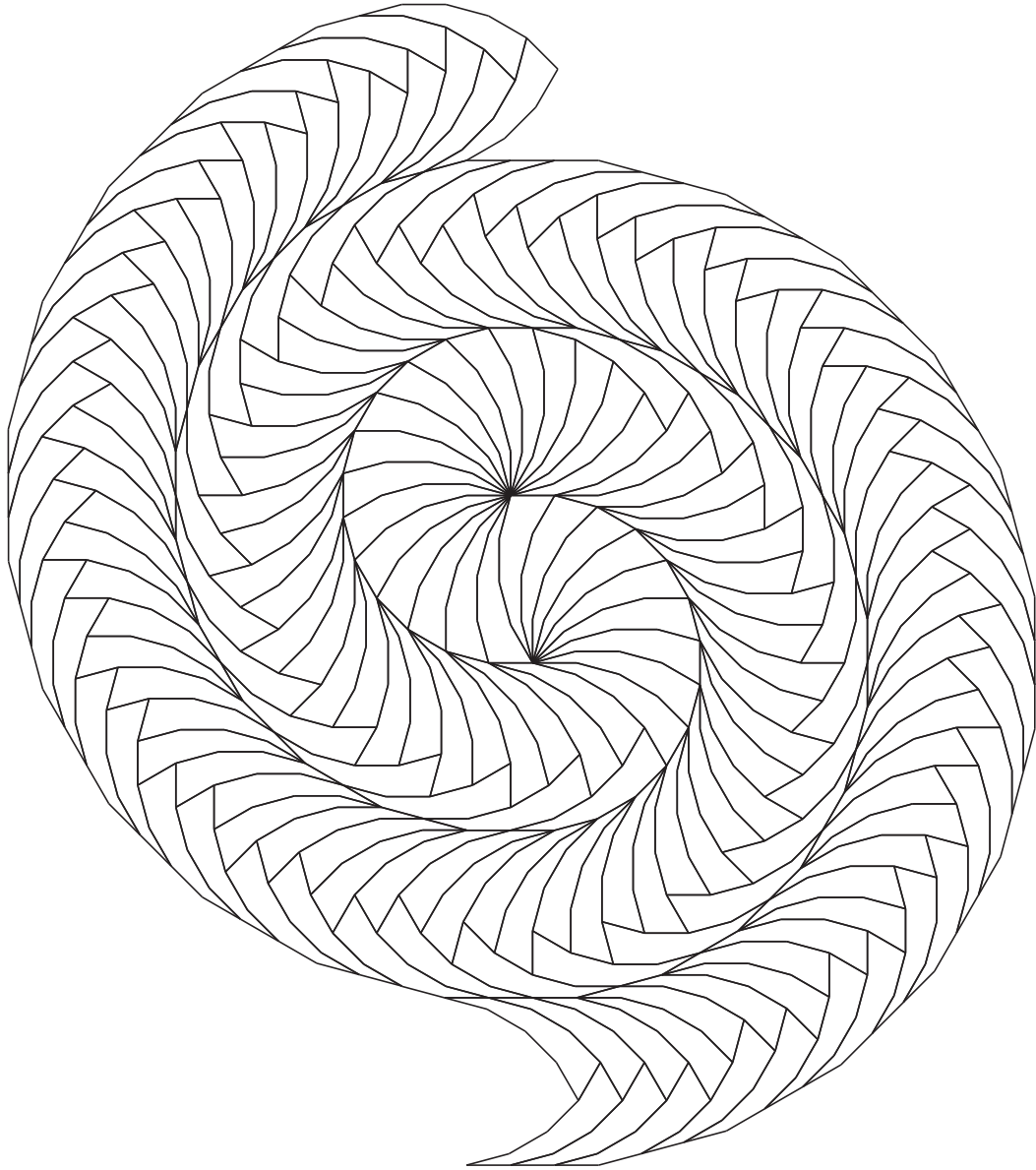


Figure 2.1: Part of a Voderberg spiral tiling [11, Section 9.5].

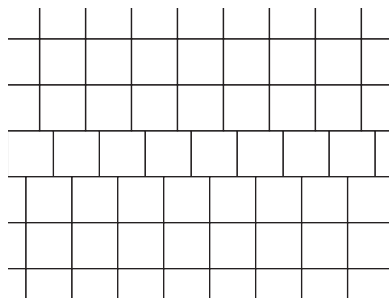


Figure 2.2: A monohedral, non-periodic, non-edge-to-edge tiling.

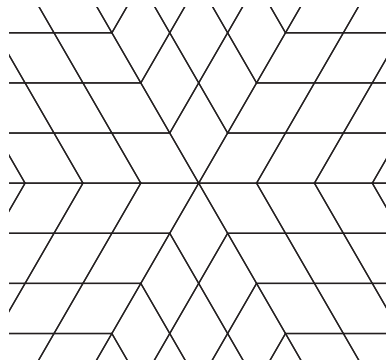


Figure 2.3: A monohedral, non-periodic, edge-to-edge tiling with rotational symmetry.

symmetry group contains at least two translations in non-parallel directions then the tiling will be called *periodic*, or *non-periodic* if it does not. The related concept of an *aperiodic set of tiles* will be discussed in Section 2.1.2. A tiling can be non-periodic if it has no symmetries at all, or a family of translational symmetries all parallel to one direction as in Figure 2.2, or only non-translation symmetries as in Figure 2.3.

Two tiles T_1, T_2 of a tiling T are said to be *equivalent* if the symmetry group $S(T)$ contains a transformation that maps T_1 onto T_2 . The collection of all tiles of T that are equivalent to T_1 is called the *transitivity class* of T_1 . If all tiles of T form one transitivity class, we say T is *isohedral*. If T is a tiling with precisely k transitivity classes then T is called *k-isohedral*. All isohedral and k -isohedral tilings are necessarily periodic. Another way to think about k -isohedral tilings is that a tile’s relative relationships to its neighbours are exactly the same for every tile in the same transitivity class. That is, in an isohedral tiling every tile is surrounded by other tiles in the same way, although the “view from a tile’s perspective” may be rotated or reflected when compared to another tile’s viewpoint if the two tiles are rotated or reflected relative to each other.

Figure 2.4 shows an isohedral tiling by rectangles. Any tile is related to any other tile by a translation that is also a symmetry of the tiling. Figure 2.5 shows a 2-isohedral tiling by 3×1 rectangles, sometimes called the “cheese sandwich” tiling. The outer two

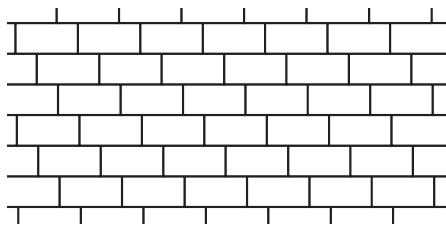


Figure 2.4: An isohedral tiling.

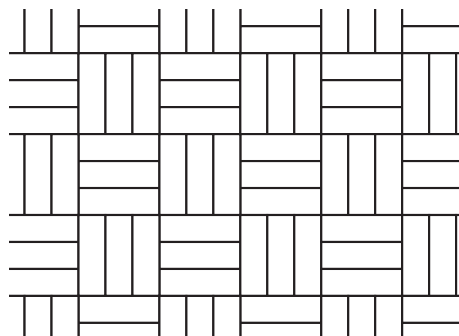


Figure 2.5: A 2-isohedral tiling.

rectangles in each block of three represent slices of bread, with the middle cheese rectangle between them. The top and bottom slices of bread in one sandwich can be mapped onto each other by a 180 degree rotation about the sandwich’s center. Bread in one sandwich can be mapped to bread in another sandwich by a translation and possibly a 90 degree rotation, and the same for mapping cheese to cheese. All of these transformations can be accomplished with symmetries of the tiling. However, any attempt to map bread onto cheese or vice versa will cause part of a sandwich to land on a rotated sandwich, causing culinary disaster. The bread and cheese therefore form the two separate transitivity classes and the tiling is 2-isohedral.

A central point of this thesis, which will be expanded upon later, is to classify shapes according to the types of tilings they admit. A shape is called *anisohedral* if it admits at least one tiling but does not admit any isohedral tiling. More generally, a shape is called *k-anisohedral* or said to have *isohedral number k* if it admits a *k*-isohedral tiling but no *m*-isohedral tiling for any $m < k$. A shape is called *aperiodic* if it admits at least one tiling but does not admit any periodic tiling.

For example, the “biting fish” in Figure 2.6 is 2-anisohedral. If we imagine the shape as a fish having a nose and a mouth, it is evident that the only way they can fit together is to have one fish biting its neighbour’s nose and that fish biting the first fish’s chin. Pairs of biting fish can be arranged by translation to tile the plane. Any symmetry of the tiling can only map nose-biters to nose-biters and chin-biters to chin-biters, so the tiling is 2-isohedral. Since any tiling must at least have this property, no isohedral tilings by

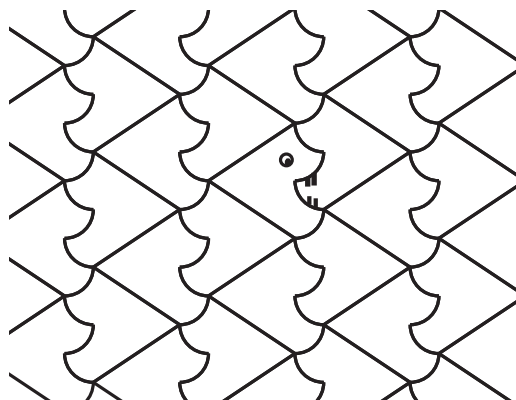


Figure 2.6: A 2-isohedral tiling by a 2-anisohedral shape.

this shape are possible and the shape itself is 2-anisohedral.

The study of monohedral tilings in the plane was at one time thought to be a solved problem, as they were dismissed in Hilbert’s 18th Problem in favour of a question about anisohedral tilings in three dimensions. The implicit assumption at the time seems to have been that all shapes that tiled the plane admitted at least one isohedral tiling. It was not until 1935 when Heesch exhibited a 2-anisohedral shape [12] (similar to the biting fish tile) that this assumption was demonstrated to be incorrect. In the next half-century relatively little progress was made. The reasonably accessible nature of the problem led to a variety of results going either unpublished or published in popular magazine articles, making it difficult to identify individual breakthroughs. It is clear that Stein described a 3-anisohedral pentagon in 1985 (Figure 2.7) [28], and by 1993 a number of 4-anisohedral shapes were known (e.g., Figure 2.8) [4].

The primary source of recent progress is the unpublished work of Joseph Myers [21], who used a computer program to perform an exhaustive enumeration of various polyforms (defined in Section 2.2), testing each shape to identify its tiling properties and isohedral number. The results identified shapes with isohedral numbers 5, 6, 8, 9, and 10, and provided the inspiration for this work. The current “record holder”, a 10-anisohedral polyhex, is shown in Figure 2.9. Similar work was carried out by Glenn Rhoads [24], but with less progress in identifying high isohedral numbers. Both of these past works can be described as “area-based” approaches; that is, they treat the underlying square, hexagon,

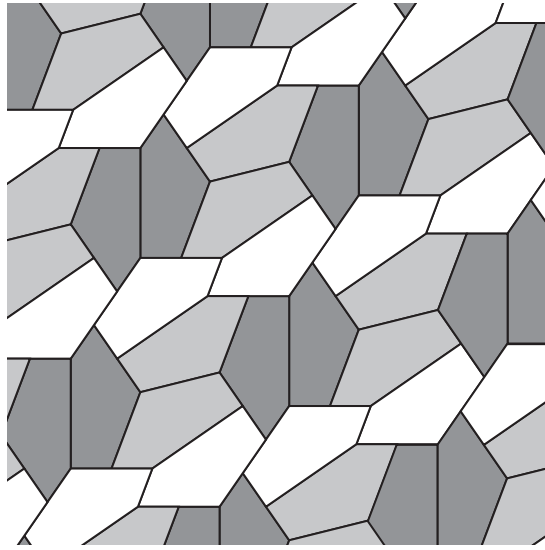


Figure 2.7: A 3-isohedral tiling by Stein's 3-anisohedral pentagon, transitivity classes indicated by shading [28].

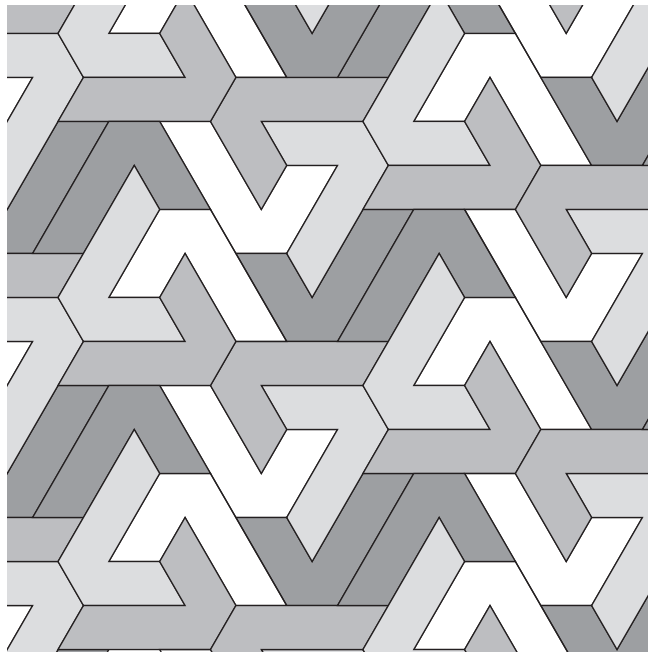


Figure 2.8: A 4-isohedral tiling by a 4-anisohedral polyiamond, transitivity classes indicated by shading [4].

or triangle lattice as a bitmap and place tiles on the lattice by setting regions of bits after applying isometries on a cell-by-cell basis to produce different tile aspects.

The study of anisohedral tilings is particularly interesting because new results often create as many questions as they answer. Myers' results are full of oddities that are not easily explained. For example, no 7-anisohedral shapes were found. It could be the case that there can be no such thing as a 7-anisohedral shape, or that we simply have not looked at the right class of shapes to find one, or even that the program had a subtle bug that prevented it from correctly identifying such a shape. The results display a clear irregularity between isohedral numbers of shapes of odd area and shapes of even area (that is, shapes composed of an odd or even number of basic polygons, see Section 2.2), with no obvious explanation. There is no clear trend of isohedral numbers increasing as the area (and hence the potential complexity of the boundary) increases. Every shape with high isohedral number seems to be a rare outlier, almost random. The difficulty of extracting any generalizations from this data motivates the push to get more data in the hope of unraveling the mystery.

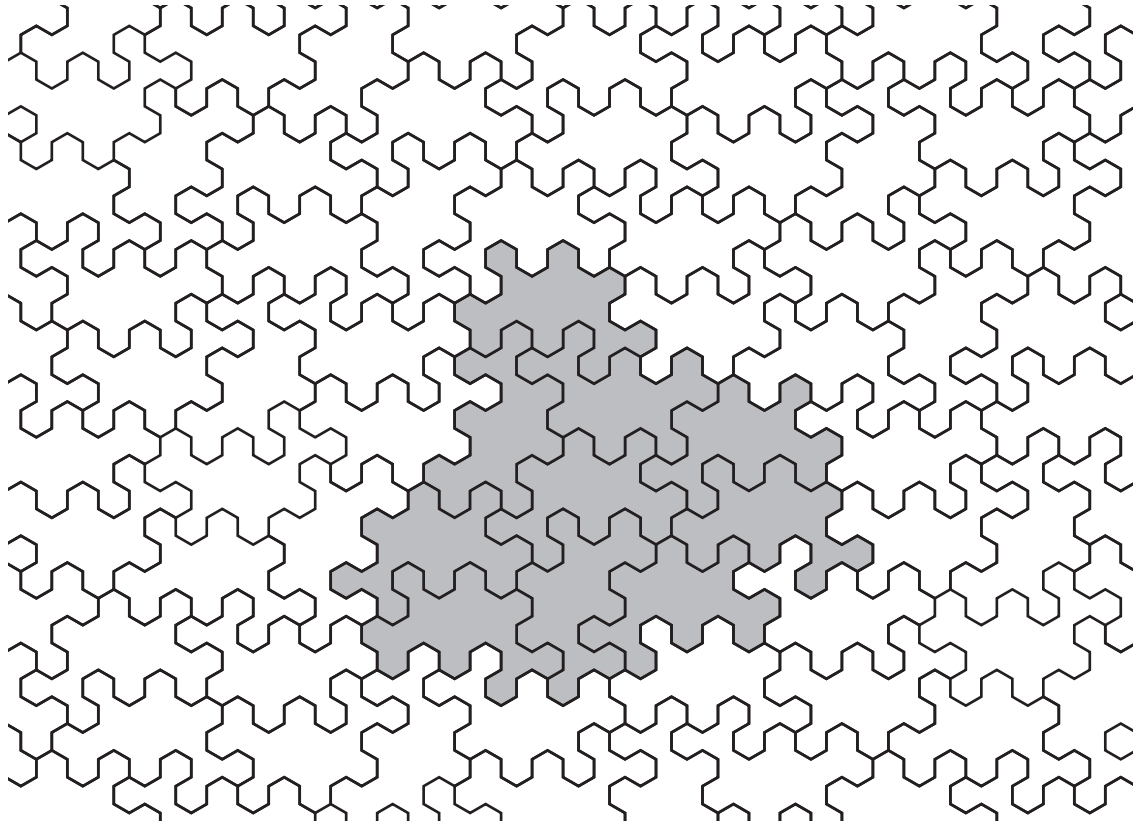


Figure 2.9: A 10-isohedral tiling by a 10-anisohedral polyhex, one representative of each transitivity class shaded [21].

2.1.1 Unbalanced Tilings

A concept closely related to the property of a tile being *k-anisohedral* is that of a *minimal isohedral repeating unit*.

A tiling has a *minimal isohedral repeating unit of size k* if k is the number of tiles in the smallest connected patch of the tiling that is itself an isohedral prototile, and the entire tiling can be partitioned into copies of such a patch that are arranged in an isohedral tiling.

We may define the concept analogous to a tile being *k-anisohedral* using the size of the isohedral repeating unit instead. From an algorithmic perspective, finding the minimal isohedral repeating unit of a shape is a convenient option for reasons described later in this section. Indeed, *Tilings and Patterns* mentions the concept of a minimal isohedral repeating unit with no explicit distinction made from isohedral number. However, the two definitions are not equivalent, and a small number of examples have been found that exhibit the difference.

It is evident that if the tiling is *k-isohedral*, the size of the minimal isohedral repeating unit must be greater than or equal to k , for the following reason: If the repeating unit did not contain a representative from one of the transitivity classes, the images of every tile in the repeating unit under symmetries of the tiling would cover the plane without encountering any representative of the missing class, and so the transitivity class would be empty. However, the converse is not true. There are tiles for which the isohedral number is strictly less than the size of the minimal isohedral repeating unit for any tiling. The first known example, shown in Figure 2.10 was discovered by John Berglund [3], from an examination of Joseph Myers' tiling data. Transitivity classes are indicated by shading, and a minimal isohedral repeating unit is outlined in bold.

This shape is 2-anisohedral, but the smallest isohedral repeating unit in this tiling—the only tiling admitted by this shape—is of size 3. Berglund introduced the term *unbalanced* to describe this tile, noting that one of the two transitivity classes was “twice the size” of the other in the sense that the ratio of tiles in each transitivity class that

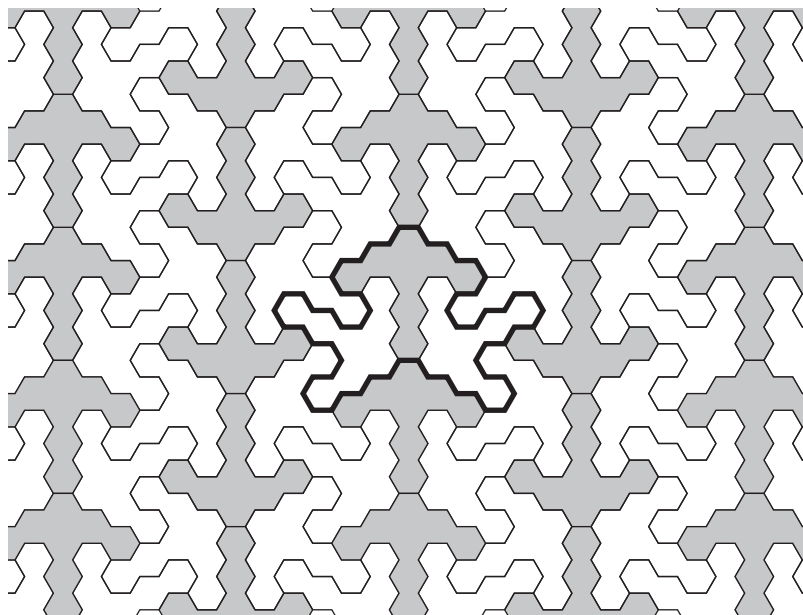


Figure 2.10: An unbalanced 2-anisohedral 8-hex.

intersect a disc of radius r converges to 2 as r increases. We can make this observation easier to apply by noting that the transitivity classes have an unequal number of representatives in a minimal isohedral repeating unit. We shall call a tiling unbalanced if it has this property, and call a tile unbalanced if it has isohedral number k and admits only unbalanced k -isohedral tilings. This property of a tile coincides with having an isohedral number smaller than the size of the minimal isohedral repeating unit.

Myers undertook a further search through his existing data [20], which revealed a small number of other k -anisohedral examples for which the only k -isohedral tiling is unbalanced. Figures 2.11, 2.12, 2.13, and 2.14 show all of the other currently known examples.

Intuitively, we can observe that the “problem” with these tiles is that we can find an isohedral repeating unit of size 3 that has an internal symmetry that maps one tile onto itself and the other two tiles onto each other, and that symmetry is a symmetry of the entire tiling. If we allowed an isohedral repeating unit to consist of part of a tile (with some matching conditions to require the tile to be put back together), we could identify an isohedral repeating unit of 1.5 tiles. This observation can be applied to each

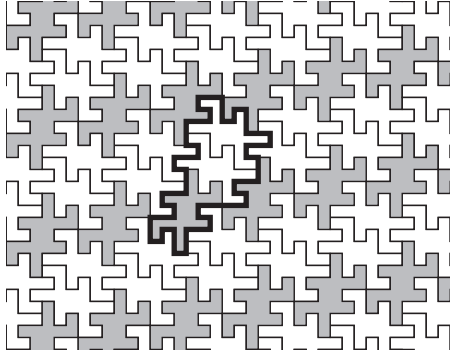


Figure 2.11: An unbalanced 2-anisohedral 18-omino.

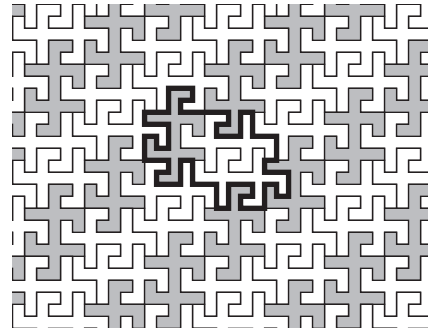


Figure 2.12: An unbalanced 2-anisohedral 20-omino.

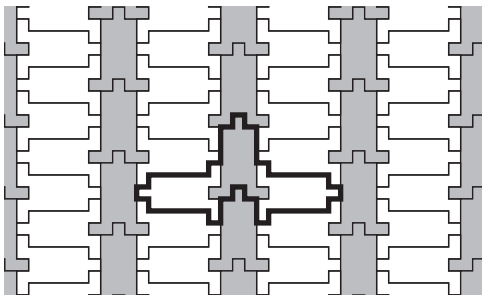


Figure 2.13: An unbalanced 2-anisohedral 20-omino.

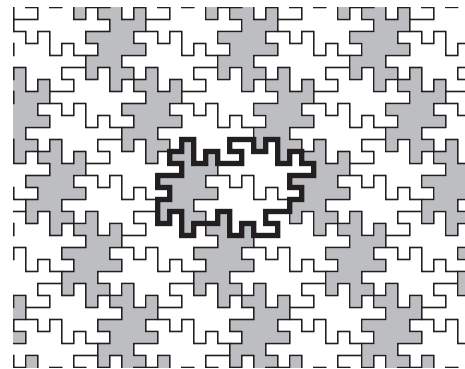


Figure 2.14: An unbalanced 2-anisohedral 22-omino.

of the five known examples by taking one tile from the larger transitivity class and a suitably-chosen half of one tile from the smaller transitivity class. So far, every known example has isohedral number 2 and a minimal isohedral repeating unit of size 3. There is no obvious reason why this situation could not occur for larger isohedral numbers, but no examples have been identified and no theoretical basis has been found to characterize shapes that admit only unbalanced tilings.

When implementing a test for tiling properties, finding the minimal isohedral repeating unit is an attractive option. The procedure is simply to assemble progressively larger patches of tiles and check if the patch is an isohedral tiler, using other optimizations to limit the patches tested to only those that can participate in a tiling. The downside is that the results will disagree with Myers' previous results, which properly computed the isohedral number. Fortunately, with such a small list of known unbalanced shapes any discrepancies can be individually checked.

2.1.2 Aperiodic Tilings

A set of tiles is called *aperiodic* if the set admits at least one tiling of the plane, but every such tiling is necessarily non-periodic. No underlying theory has been found to fully characterize such sets, although many of the examples discovered so far and the techniques used to prove aperiodicity have certain common features such as Ammann bars [11, Chapter 10]. The discovery that aperiodicity was possible came as something of a surprise to researchers, as it has significant implications for the decidability status of many tiling problems. If every set that admits a tiling admits a periodic tiling, then there exists an algorithm to decide whether a given input tiles the plane by considering progressively larger patches of tiles until a translational repeating unit is found or no patch can be extended any further. The fact that this is not the case was discovered in the context of Berger's proof that the decision problem "does a given set of tiles admit a tiling of the plane" is undecidable [2].

Berger's first example of an aperiodic set, published in 1966, was an elaborate con-

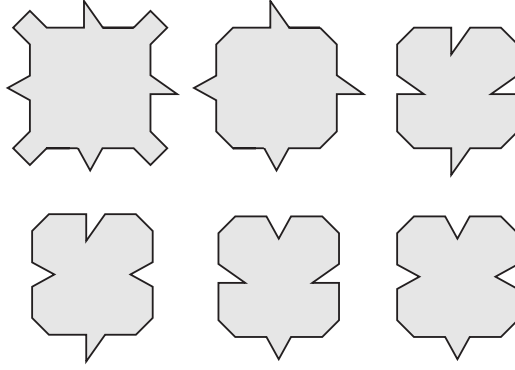


Figure 2.15: Robinson's R1 set of aperiodic tiles.

struction consisting of 20,426 Wang tiles. Wang tiles are a particular restriction of tiling theory to unit squares with coloured edges that may only be assembled edge-to-edge by translation (not rotation or reflection) in such a way that adjacent colours match. This construction was quickly simplified by Berger to require only 104 tiles, and various researchers have since exhibited much smaller sets. As of this writing, the smallest aperiodic set of Wang tiles appears to be one with 13 tiles published by Karel Culik in 1996 [7].

Of course, we wish to deal with the more general case of tilings by polygons with no colouring rules and all rotations and reflections allowed. The first known example of an aperiodic set under these conditions was published by Robinson in 1971 [25]. Robinson's paper discusses several variants, but the one most commonly presented is a set of 6 shapes, reproduced in Figure 2.15.

The next breakthrough came from Roger Penrose in 1974 [22], with the discovery of an aperiodic set of five shapes that led quickly to the discovery of a related set of size two. This set of size two, known as the "kite and dart", are usually presented as quadrilaterals with coloured corners as in Figure 2.16 subject to the matching conditions that corners touching each other in a tiling must have the same colour. However, these conditions can be enforced geometrically by suitably modifying the sides of the quadrilaterals to limit the ways they can fit together. The sides come in two lengths, in the ratio $1:\tau$ (the golden ratio), and the interior angles are multiples of $\pi/5$.

Through a modification uncovered by the analysis of tilings by the kite and dart,

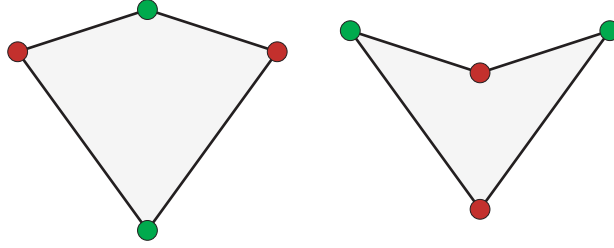


Figure 2.16: Penrose's kite and dart aperiodic set.

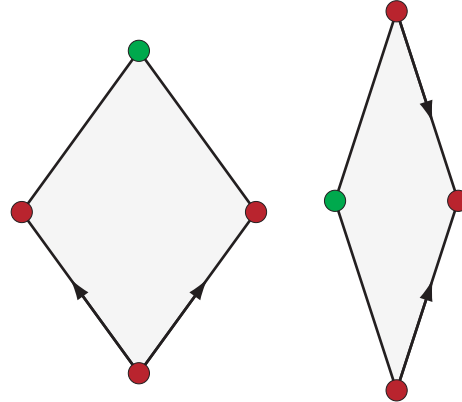


Figure 2.17: The aperiodic Penrose rhombs.

another related aperiodic set of size two was discovered. This set, known as the “Penrose rhombs”, is even simpler (in one sense) as it consists of two quadrilaterals with all sides the same length and all internal angles multiples of $\pi/5$, shown in Figure 2.17 with two matching conditions: coloured corners such that like colours must coincide and directed edges such that adjacent arrows must point in the same direction. As with the kite and dart, these conditions can be enforced by modifying the shape of the sides, for example as shown in Figure 2.18. Figure 2.19 shows part of a tiling by the Penrose rhombs constructed according to the matching conditions.

The question of whether there exists a single aperiodic shape remains open. This question has not even been settled for such restricted cases as convex pentagons. One of the motivations for this research is that if aperiodic shapes exist, an exhaustive enumeration of various classes of shapes will eventually stumble across an example. The choice of *polysnakes* (defined in Section 3.1) is motivated in part by the fact that the outline of any

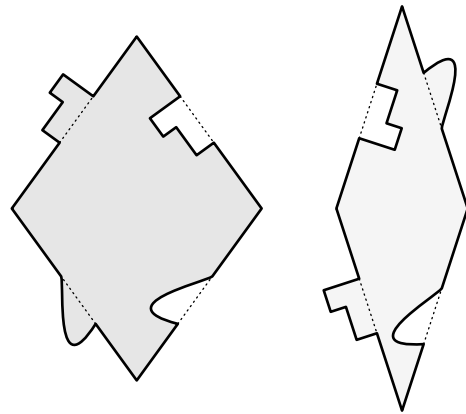


Figure 2.18: Penrose rhombs with geometric matching conditions.

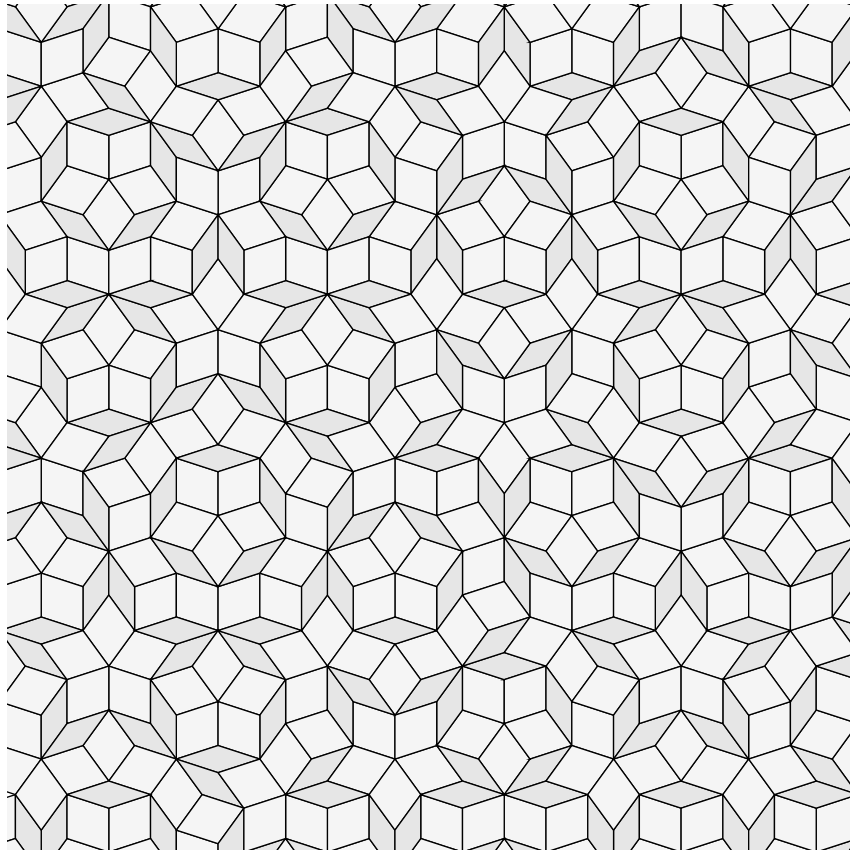


Figure 2.19: Part of a tiling by the Penrose rhombs.

patch of Penrose rhombs can be encoded (but without matching conditions) as a 5-snake. This creates the speculative idea that these shapes might be in some way sufficient to encompass the property of aperiodicity.

2.1.3 Heesch’s Problem

For a shape that does not tile the plane, we define the shape’s *Heesch number* to be the maximum number of layers, called *coronas*, to which the shape can be surrounded by congruent copies of itself. For a shape that tiles the plane, this number would be infinite.

The precise definition of a corona has varied in past work. The most common definition is that given a patch of tiles A , a corona is a set of tiles B such that:

- the tiles of B are disjoint from the interiors of the tiles of A
- $A \cup B$ forms a patch (note that a patch must be a topological disc)
- the closure of the complement of $A \cup B$ is disjoint from A .

The coronas of interest are usually minimal in the sense that removing any one tile from B makes it fail to meet the definition. Some researchers have required only that the set B be *connected* and relaxing the requirement that $A \cup B$ be a topological disc, which in some cases allows an additional corona to be formed that contains holes. Such a corona cannot be surrounded a further time, because filling the holes would require tiles disconnected from the rest of the next corona. I will adopt the former convention.

In my implementation of a general test for non-tiling described in Section 5.2, I use the term *surround* to mean something slightly different from the above definition of corona. Because the boundary word approach is concerned with “covering” each edge along the boundary, a surround will be like a corona except that the part of the plane not covered by $A \cup B$ must be disjoint from A *except possibly at vertices*.

Heesch’s Problem [13] asks for which positive integers does there exist a shape with that Heesch number. The most common concern is whether there is an upper bound

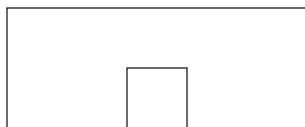


Figure 2.20: A tile with Heesch number 0.

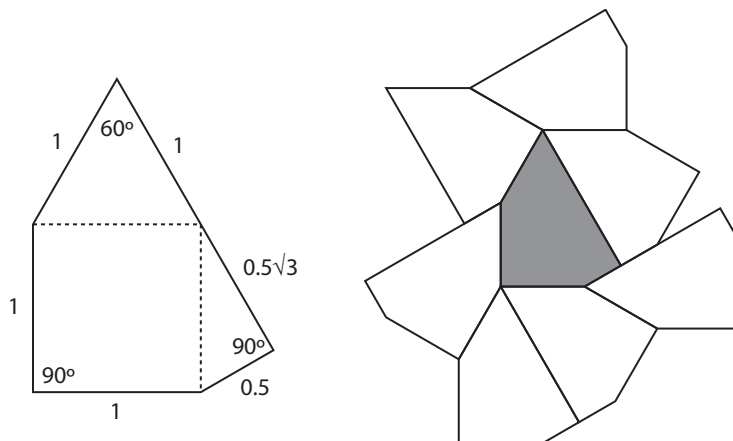


Figure 2.21: A tile with Heesch number 1.

on Heesch numbers, but it is also of interest whether there are positive integers below any such bound that cannot occur as the Heesch number of a shape. It should be noted that if there is an upper bound on Heesch number, the bound implies an algorithm that decides the problem of whether a shape tiles the plane in finite time for any case where tilings must be edge-to-edge or otherwise can be discretized into a finite set of relevant adjacent pairs: simply explore all possible surrounds until this depth is exceeded.

It is easy to see that there are plenty of shapes with Heesch number 0, that is, shapes that cannot be surrounded even once by copies of themselves. Perhaps the most straightforward example is a shape with a concavity that no other part of the shape can fit into, for example Figure 2.20.

At the time he posed this question, Heesch was only able to present a shape with Heesch number 1 (Figure 2.21). It requires some work to explore the various first coronas and verify that none of them can be surrounded by a second corona.

Most subsequent work has explored hexagons or polyhexes with various combinations

of bumps and notches on their edges. This has uncovered shapes with Heesch numbers 2, 3, 4, and 5, with the current “record-holder” of 5 discovered by Casey Mann (Figure 2.22) [17].

In a curious way, Heesch numbers are the dual of isohedral numbers. Where the isohedral number is a measure of increasing complexity of a shape that tiles the plane, the Heesch number is a measure of increasing complexity of a shape that does not tile the plane. Both measures have no known upper bound and little formal or even intuitive justification for why an upper bound should or should not exist. An aperiodic shape, if such a thing exists, can be informally thought of as being the limit at infinity of the brute force algorithms for determining both isohedral numbers and Heesch numbers: the boundary between tiling and non-tiling where the straightforward proof techniques break down.

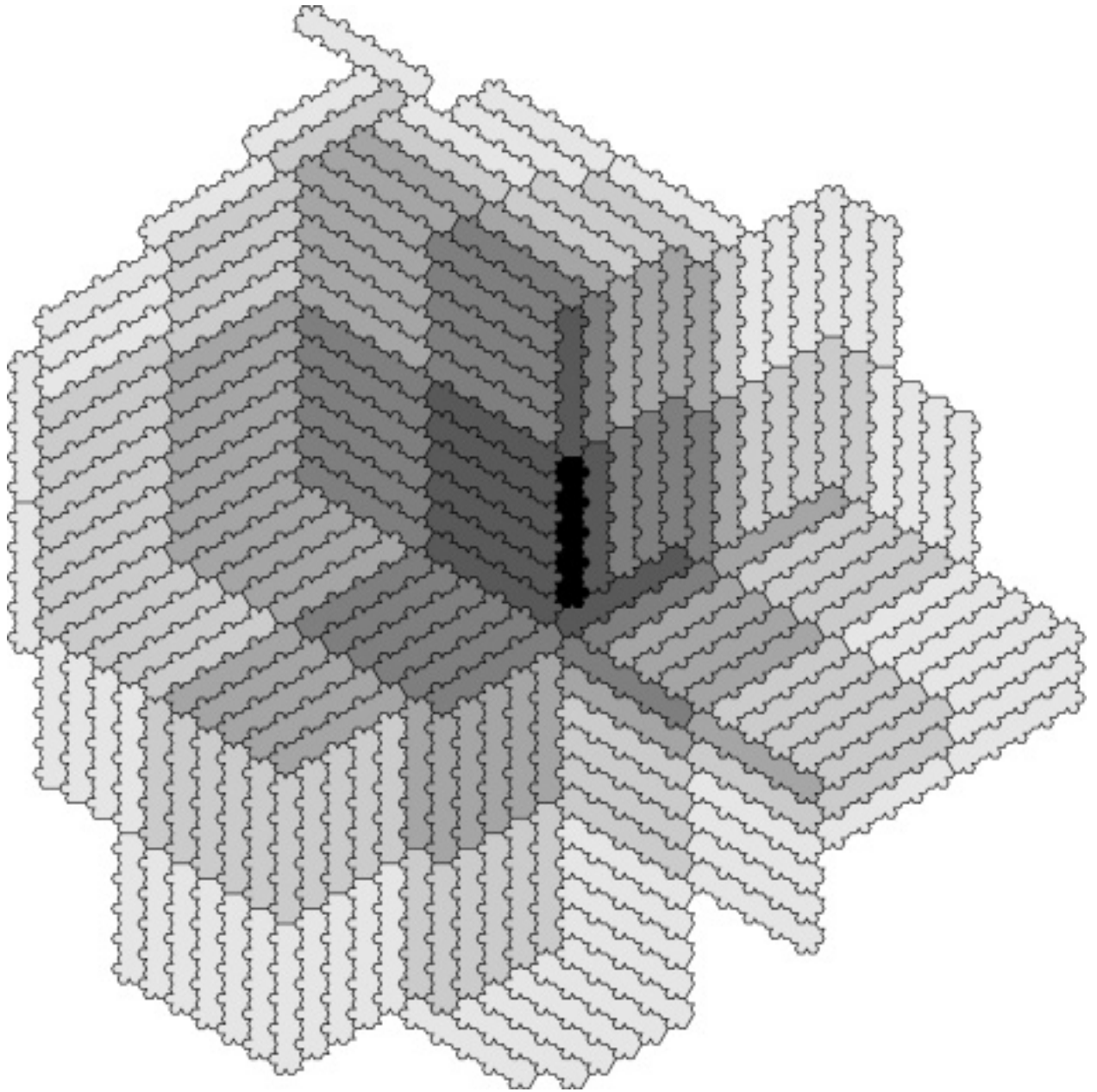


Figure 2.22: A tile with Heesch number 5. (Image courtesy of Casey Mann.)

2.1.4 Isohedral Tiling Criteria

A key component of the work described later in this thesis is the implementation of an algorithmic test to determine whether a given shape admits an isohedral tiling or not. Fortunately, isohedral tilings have been classified in ways that lead to the development of a set of necessary and sufficient criteria that can be used to test an input shape.

Grünbaum and Shephard [11] classified isohedral tilings into 93 types denoted by IH1, IH2, ..., IH93, and proved their list complete. Heesch [14] also gave a system of classification with only 28 types that predates the IH types, also singling out 9 of those types as the most general ones from which the rest can be obtained as special cases. IH types are a refinement of Heesch types, since Heesch was only considering tiles with no internal symmetries where Grünbaum and Shephard elaborated on each possible internal symmetry and its effect on the overall properties of each tiling.

Heesch types are characterized by a sequence of symbols indicating a decomposition of the perimeter of the shape into segments having specific geometric relationships to other segments or to themselves. The relationships allowed are that of being centrosymmetric (unchanged by a 180° rotation about its own center), being in a pair of segments that are identical under a translation but have opposite orientations traveling clockwise around the perimeter, being in a pair of segments that are identical under a glide reflection, and being an adjacent pair of segments that are identical under a 60° , 90° , or 120° rotation about their meeting point.

The criteria implied by Heesch's 9 most general types, emphasized more recently by Schattschneider [27], imply a test for isohedral tiling by testing whether the perimeter of a shape admits a decomposition into 3 to 6 segments with suitable geometric relationships. The decomposition of the shape implies the geometric relationship between a tile and all of its immediate neighbours. In an isohedral tiling, this is sufficient to determine the entire tiling since every tile has the same relative relationship to its neighbours. Any centrosymmetric segment or pair of related segments indicated in a criterion may be empty so long as at least 3 segments in total remain non-empty.

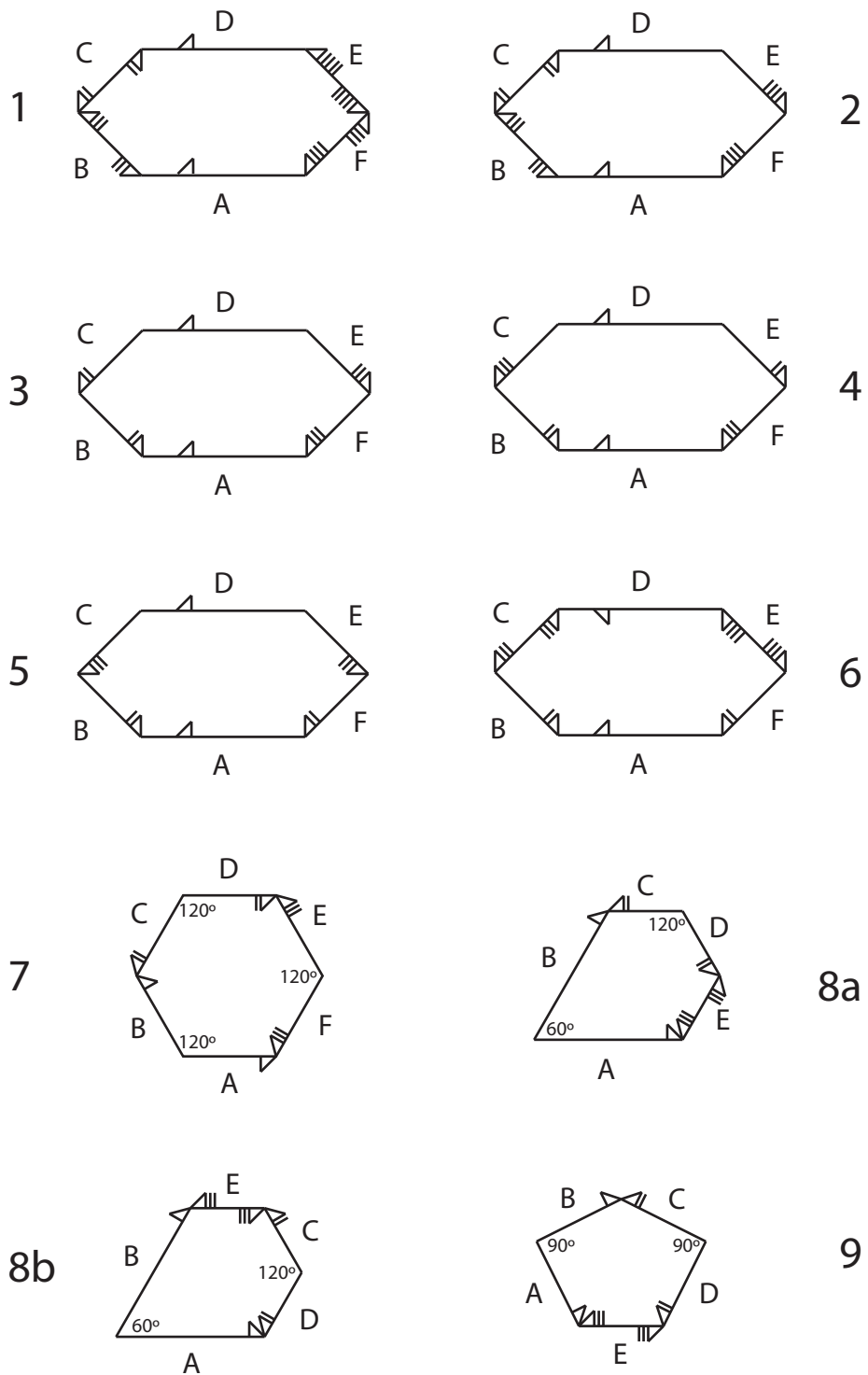


Figure 2.23: The 9 boundary criteria for isohedral tiling.

The criteria are presented here both as diagrams in Figure 2.23 with edge segments marked to show their relevant properties and relationships to each other, and in words in the list below.

1. ABCDEF: A and D related by translation, B, C, E, and F centrosymmetric.
2. ABCDEF: A and D related by translation, B and C centrosymmetric, E and F related by glide reflection.
3. ABCDEF: A and D related by translation, B and C related by glide reflection, E and F related by glide reflection.
4. ABCDEF: A and D related by translation, B and E related by translation, C and F related by translation.
5. ABCDEF: A and D related by translation, B and F related by glide reflection, C and E related by glide reflection, with the two glide reflections parallel.
6. ABCDEF: A and D related by glide reflection, B and F related by glide reflection, C and E centrosymmetric, with the two glide reflections perpendicular.
7. ABCDEF: A and B related by 120° rotation, C and D related by 120° rotation, E and F related by 120° rotation.
8. ABCDE or ABECD: A and B related by 60° rotation, C and D related by 120° rotation, E centrosymmetric.
9. ABCDE: A and B related by 90° rotation, C and D related by 90° rotation, E centrosymmetric.

Criterion 1 is the well-known *Conway Criterion* [26], often singled out both for its simplicity and for the curious fact that (empirically) it seems to identify a large fraction of polyforms that tile the plane.

Heesch proved that this is a necessary set of criteria—any shape that admits an isohedral tiling will satisfy one (or more, in a few degenerate cases) of these criteria. Grünbaum

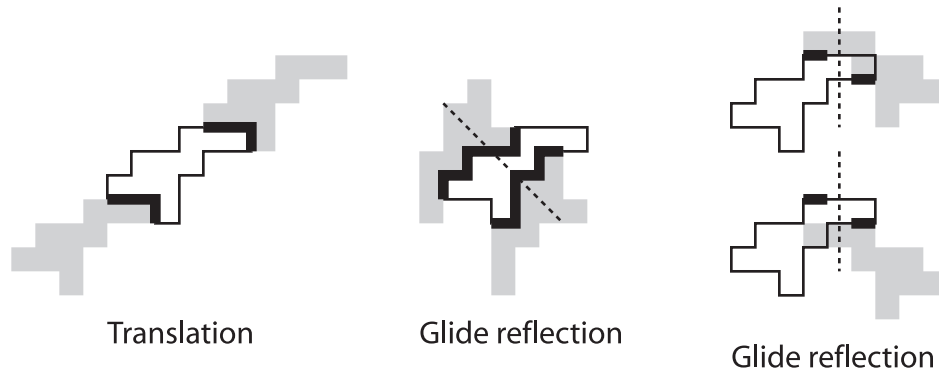


Figure 2.24: A non-tiling polyomino that passes the weakened version of criterion 5.

and Shephard likewise proved that their list of IH types is a complete classification of all isohedral tilings. However, previous descriptions of these criteria lacked the additional constraints in criteria 5 and 6 on how the two glide reflections must be related.

During the testing of my implementation of a test for isohedral tiling, I identified counter-examples that prove that the weaker versions of criteria 5 and 6 are not sufficient. These examples, shown in Figure 2.24 and Figure 2.25, are shapes that do not tile the plane, but admit a decomposition under one of the weakened criteria. It is not clear whether Heesch was aware of this constraint, as the focus of his system of classification was to give necessary criteria and he may not have considered sufficiency.

With the additional constraints, the experimental results of my isohedral tiling test implementation agreed completely with the results of past work, which did not rely on these criteria at all. However, I have not proved that these modified criteria, or indeed any of the other seven, are sufficient.

Previous descriptions have also been lax in emphasizing that Criterion 8, unlike the other criteria, comes in two enantiomorphic versions (unequal versions produced by a reflection).

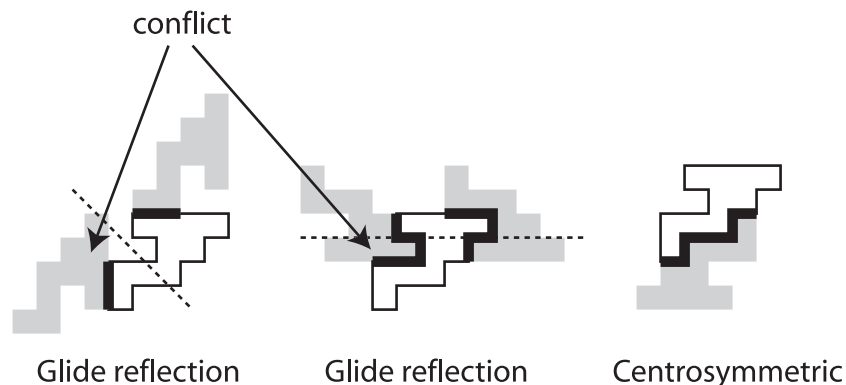


Figure 2.25: A non-tiling polyomino that passes the weakened version of criterion 6.

2.2 Polyforms

A *polyform* is a shape constructed by joining copies of a basic polygon together in a connected edge-to-edge configuration. The specific types of polyforms we will consider have their own names: the *polyomino* (squares), *polyhex* (regular hexagons), and *polyiamond* (equilateral triangles), as shown in the examples in Figure 2.26. Using a computer program to search through various polygons looking for ones with interesting tiling properties requires some sort of search strategy, and polyforms are a useful class of shapes to enumerate as they are characterized by an integer area, with only finitely many polyforms of each area (exponentially increasing as area increases). This provides a natural way to expand the search space to very large numbers of polygons. Polyforms also have unit-length edges, which will become a useful property later on.

For consistency with most previous work, we allow polyforms to have internal holes. Since our definition of tiling does not allow any shape with an internal hole to tile the plane, such polyforms can be identified and discarded before applying any other tests of tiling properties. Polyforms are characterized by their area, expressed as the number of basic polygons of which they are composed.

To restrict interest to the combinatorial aspects of a polyform and disregard both scale and the infinity of possible orientations, we think of polyforms as being subsets taken from the regular tiling by the basic polygon. We further distinguish between *fixed polyforms*,

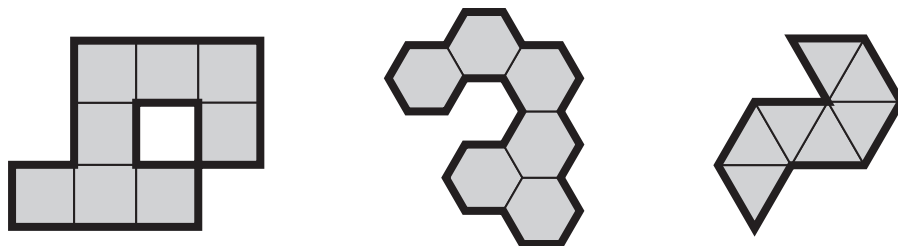


Figure 2.26: Sample polyomino, polyhex, and polyiamond.

which are unique up to translation only, *one-sided polyforms*, which are unique up to translation and rotation, and *free polyforms*, which are unique up to translation, rotation, and reflection. That is, two distinct rotated versions of the same shape are considered to be the same free polyform but two different fixed polyforms. The enumeration of these classes for a fixed area or perimeter is a difficult problem that has no known closed-form solution except for restricted cases.

Polyominoes have a long history in recreational mathematics, going back to the natural development of polyomino puzzles arising from games like chess and Go. The term polyomino itself was coined by Solomon Golomb [10], and popularized by Martin Gardner's columns in the magazine *Scientific American* [9]. Most of the study of polyominoes deals with the enumeration problems, tiling a finite (often rectangular) region, or tiling the plane.

2.2.1 Sufficiency of Edge-to-Edge Tiling

When testing the tiling properties of a polyform, my implementation will only consider edge-to-edge tilings. This is a decision that requires some justification of why relevant tiling properties will not be overlooked. The past work of Myers [21] and Rhoads [24] also makes this implicit assumption, without justification. It is intuitively tempting to believe that this is a non-issue, but intuition with regard to tiling theory has been suspect before.

For one class of polyforms, at least, this is trivially a non-issue:

Proposition 2.2.1.1. *Every tiling by a polyhex is edge-to-edge.*

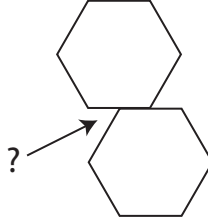


Figure 2.27: Proof that all tilings by polyhexes are edge-to-edge.

Proof. Since a polyhex is a subset of the regular hexagonal lattice, the interior angle at any vertex is either 120° or 240° . If a tiling vertex occurs in the middle of a hexagon edge, the angles around that vertex must be 120° , 180° and something to fill the remaining 60° as shown in Figure 2.27, which is impossible. Therefore every tiling vertex coincides exclusively with polygon vertices and the tiling is edge-to-edge. \square

For the remaining kinds of polyforms, I am only able to state the desired result as a conjecture and sketch the steps that justify the intuition that the conjecture ought to be true. The key property of polyforms in this situation is that they are polygons composed entirely of unit-length edges. This will also apply to polysnakes when they are defined in Section 3.1. Since I will not be computing Heesch numbers, and aperiodicity would require a proof by hand anyway, the main concern is that isohedral numbers are being correctly determined.

Conjecture (Sufficiency of edge-to-edge tiling) 2.2.1.2. *If a polygon with unit length edges admits a non-edge-to-edge m -isohedral tiling, then it admits a k -isohedral edge-to-edge tiling for some $k \leq m$. That is, it is not the case that considering only edge-to-edge tilings will assert that a shape is k -anisohedral when there is a non-edge-to-edge tiling with fewer transitivity classes.*

Definition 2.2.1.3. *A faultline of a tiling T is a line in the plane that does not intersect the interior of any tile of T and every tiling vertex along the line lies in the interior of some polygon edge.*

Proposition 2.2.1.4. *Every non-edge-to-edge tiling by a polygon with unit length edges contains a faultline.*

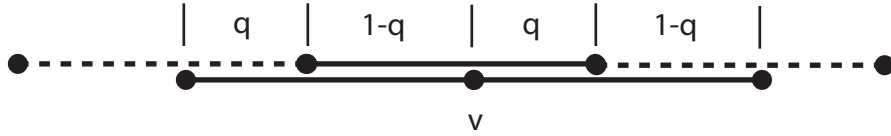


Figure 2.28: Proof of faultline existence.

Proof. Since the tiling is not edge-to-edge, it contains a tiling vertex v that lies in the interior of some polygon edge e , as shown in Figure 2.28. Since all edges are unit-length v divides e into a segment of length q and a segment of length $1 - q$ for some $0 < q < 1$. v must be the meeting point of two edges colinear with e . One of those edges has an overlap with e of length q , and the other $1 - q$. Then these edges have remaining segments of length $1 - q$ and q that must be covered by some other polygon edges, but since all edges have unit length, covering those segments will produce new segments of length q and $1 - q$ that need to be covered. This process repeats forever in both directions along the line that is the extension of e . Therefore, every tiling vertex along this line must lie in the interior of some polygon edge, and the line can never cross into the interior of a tile. This satisfies the definition of a faultline. \square

Proposition 2.2.1.5. *If a non-edge-to-edge tiling by a polygon with unit length edges contains more than one faultline, all faultlines are parallel to each other.*

Proof. Suppose two faultlines are not parallel to each other. Then their intersection point is a place where two non-parallel lines, neither of which intersect the interior of any tile, meet. This ensures that no polygon edge can cross the intersection point, or one of the faultlines would intersect the interior of the tiles on either side of the edge. Therefore the intersection point does not lie in the interior of any polygon edge, and multiple tiles must meet at the intersection point so it is also a tiling vertex. This contradicts the definition of a faultline. \square

Proposition 2.2.1.6. *A non-edge-to-edge tiling by a polygon can be transformed into an edge-to-edge tiling by “mending” each faultline.*

Proof. The proof of Proposition 2.2.1.4 implies the obvious method of removing a faultline from a tiling: translate the two parts of the tiling separated by the faultline relative to each other by distance q to make all of the tiling vertices along the faultline coincide with polygon vertices. Proposition 2.2.1.5 ensures that this process can be carried out on every faultline independently. Therefore, all faultlines may be systematically removed from the tiling until it becomes edge-to-edge. \square

This at least ensures that any polyform that tiles the plane admits an edge-to-edge tiling. By extending the faultline definition to make sense in a patch of tiles, it is evident that any patch of tiles can be converted to an edge-to-edge patch of tiles; therefore if a polyform can be surrounded k times by itself, it can be so surrounded in an edge-to-edge way. As the process of mending a faultline may expose a vertex of the previous corona, this does not guarantee that Heesch number is preserved.

The natural route to proving the conjecture would seem to be proving that given an m -isohedral non-edge-to-edge tiling, transforming it to an edge-to-edge tiling as described in Proposition 2.2.1.6 does not increase the number of transitivity classes. Since all m -isohedral tilings are periodic, this situation implies the existence of an infinite family of parallel faultlines, with a finite number of possible perpendicular distances between them and a finite number of values of q . Therefore the tiling consists of a collection of “strips”, which the mending process will translate relative to each other in a consistent way.

The presence of faultlines also heavily constrains the possible symmetries in the tiling. The only possible rotation is a halfturn, since it must map lines to parallel lines. Likewise, the only possible reflections are across axes either parallel to or perpendicular to the faultlines. Given these constraints, it may be possible to prove that if tiles t_1 and t_2 were in the same transitivity class in the non-edge-to-edge tiling, they must be in the same transitivity class in the transformed tiling.

Chapter 3

Thesis Summary

The study of tilings by a single polygon has several major open problems:

- What isohedral numbers are possible?
- Is there an aperiodic shape?
- What Heesch numbers are possible?

Most recent progress on these questions, particularly the advancement of the highest known isohedral number to 10 and the highest known Heesch number to 5, has been the result of computer programs searching through various classes of polygons and identifying any with relevant properties. This thesis will continue in this direction by reimplementing the work of Myers [21] to provide some confirmation of his experimental results, and extend the search to a new class of shapes. The primary objective is to continue the search for higher isohedral numbers, with the additional hope that the search might stumble across an aperiodic shape. My work will not directly determine Heesch numbers, although it could in principle uncover shapes of interest for that search.

The past work of both Myers and Rhoads [24] can be described as “area-based” in the sense that their implementations represented the plane as a lattice of cells, treating a shape as a set of connected cells and applying bitmap operations to manipulate, transform,

and arrange shapes. This approach is suited to handling polyforms. However, the space of polyforms has in a manner of speaking been “mined out”, as the number of polyforms for a given area (number of cells) grows exponentially with the area, making it difficult to extend the search much farther than Myers’ program has already gone.

My implementation instead takes a “boundary-based” approach, describing a shape by a sequence of characters that represent steps along its boundary. This approach can represent polyforms to reproduce the existing results, and also can be generalized to represent a new class of shapes that I call *polysnakes*.

The implementation consists of two major parts: an enumeration step to produce shapes, and a set of tests to determine their tiling properties. The tests determine whether a shape can tile by translation, tile by the Conway Criterion, tile isohedrally, tile k -isohedrally for some $k \geq 2$, or not tile at all.

The remainder of this chapter describes the concepts of boundary words that are unique to my approach, and defines polysnakes. Chapters 4 and 5 describe the algorithms and implementation details of the two major parts of the program.

3.1 Boundary Words

The boundary of a polyform consists of a sequence of unit-length steps chosen from a fixed set of directions. We can observe that all of the tiling properties of the shape are implicitly contained in the sequence of directions, since the sequence uniquely determines the shape. We can express the set of directions as an alphabet that we call the *boundary language* and write the boundary as a string over that alphabet, which we call the *boundary word*. In the polyomino case, the characters correspond to north, east, south, and west. Each direction is oriented, so there is a negation operator on the alphabet. Since the negation operator is its own inverse, the boundary language alphabet must have an even number of characters.

For implementation purposes, it is particularly convenient to use the alphabet

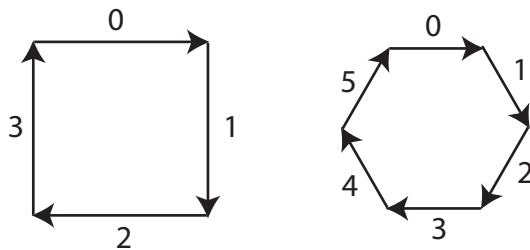


Figure 3.1: Polyform boundary language directions.

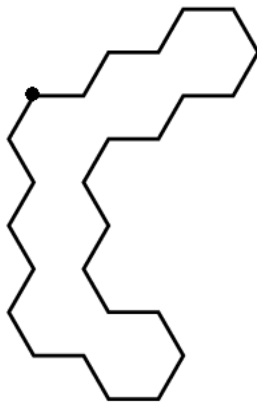


Figure 3.2: A 6-hex with boundary word 050501232321210123454545.

$0, 1, 2, \dots, N - 1$, where N is the number of directions (4 for polyominoes, 6 for polyhexes and polyiamonds). Each direction is obtained by a rotation by $2\pi/N$ of the previous direction. Figure 3.1 shows the directions of the boundary language used by my implementation. The polyhex in Figure 3.2, described clockwise starting from the marked vertex, is represented by the boundary word 050501232321210123454545.

In this scheme, we can negate a character k by the operation $k \rightarrow (k + (N/2)) \bmod N$, rotate it clockwise by $2\pi/N$ by the operation $k \rightarrow (k + 1) \bmod N$, and reflect it across the axis parallel to the 0 direction by $k \rightarrow (N - k) \bmod N$. Negation will be denoted by \bar{k} , with string negation being the composition of string reversal and per-character negation.

It should be noted that a string in this alphabet representing a closed shape is a circular word, having no inherent starting point. It does however have an inherent orientation. For the purposes of consistency, we write all words in the clockwise direction.

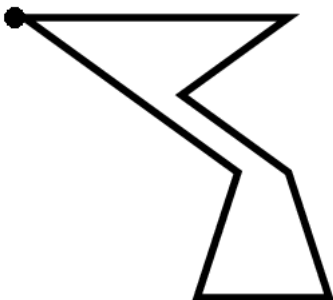


Figure 3.3: A 5-snake with boundary word 004125866.

The reflection operator described above reverses this direction as a side effect. We can reverse the direction explicitly by applying string negation as described above.

As the following sections will show, we can implement tests for tiling properties as if they were problems in formal languages. This creates an interesting opportunity to generalize the boundary language: if $N = 4$ describes the polyominoes and $N = 6$ describes polyiamonds and polyhexes (the latter are described by $N = 6$ with the restriction that consecutive characters must differ by $\pm 1 \pmod N$), what can we describe by $N = 8$, $N = 10$, etc.?

We define an k -snake, or *polysnake* in general, to be a string over the language $0, 1, 2, \dots, 2k - 1$, where the characters represent unit steps parallel to the edges of a regular $2k$ -gon, such that the string represents a self-avoiding polygon. That is, the string represents a series of steps forming a closed loop from an origin point that does not intersect with itself except where the last step returns to the origin. The negation, rotation, and reflection operations already described are all valid for polysnakes using values of $k \geq 2$. Polysnakes are characterized by their length (perimeter), unlike the usual characterization of polyforms by their area. An example of a 5-snake of perimeter 9, represented as 004125866 going clockwise from the marked vertex and using the generalized language with 0 being a unit step to the right, is shown in Figure 3.3.

There is a body of literature on self-avoiding polygons [16]; however, the usual definition of a self-avoiding polygon is a closed self-avoiding walk on the integer lattice. A

variety of combinatorial properties of these objects have been studied, but none of that material seems to be relevant to my use of polysnakes.

3.2 Boundary-Based Tiling Criteria

The isohedral tiling criteria of Section 2.1.4 can be translated into a problem of finding a decomposition of a circular string into substrings with appropriate properties. We need only describe how centrosymmetric, translated, glide reflected, and rotated segments can be recognized.

Centrosymmetric segments are palindromes.

Translational pairs of segments are strings that are each others' negation.

Glide reflected pairs of segments are strings that are related by applying the generic reflection (across the 0 direction) to one string and then adding a constant modulo $2k$ to each character. The rotation constant determines which axis of reflection the final transformation corresponds to.

Rotational pairs of segments are strings that meet at a common endpoint and are related by reversing one of the strings and adding a constant modulo $2k$ to each character. The constant depends on the angle of rotation in question, and not every boundary language allows every direction.

The identification of a substring or pair of substrings as having one of these properties corresponds to a way of laying down two neighbouring tiles. The length of the substring corresponds to the number of edges that match up along the intersection point. If the substring identified is not maximal, then the corresponding geometric interpretation will have edges touching each other that are not accounted for in the boundary word interpretation. For centrosymmetric and rotational segments, this means being maximal with respect to a fixed center location; for translational and glide reflected pairs, maximal with

respect to a fixed pair of locations that align with each other. This observation is useful in cutting down the number of substrings that must be considered. However, the fact that the 9 isohedral criteria each define a legal way of surrounding a tile with copies of itself actually ensures that no decompositions with non-maximal segments will ever be found.

Chapter 4

Implementation: Enumeration

4.1 Enumeration of Polyforms

The definitive algorithm for enumerating fixed polyforms in a way that explicitly produces each shape is known as Redelmeier's algorithm from its original development by D. H. Redelmeier in 1981 [23]. Various details have been added by other authors [19, 18] over the years, so I will present the algorithm and its implementation details in their entirety. The pure enumeration problem of counting fixed polyforms by area has been considerably advanced in recent years by Jensen's method [15]; however, this approach obtains its exponential speedups specifically by not producing each shape, making it unsuited for the problem at hand where we want to inspect each shape to determine its tiling properties. Redelmeier's algorithm was initially presented for polyominoes, but is trivially generalizable to polyhexes and polyiamonds, or in principle it could be used on any undirected graph with suitable caveats about the meaning of what it produces. In the interests of brevity, my presentation will deal with the algorithm on polyominoes, with minor notes where modifications are required for other shapes.






	FREE
	OCCUPIED
	UNTRIED
	REACHABLE
	BLOCKED

Figure 4.1: Lattice cell legend.

4.2 Redelmeier's Algorithm

The basic approach of Redelmeier's algorithm is recursive generation of polyforms of size $k + 1$ by adding a single cell to each polyform of size k in each possible position. Most of the details are concerned with ensuring that each distinct polyform is only produced once.

The principal data structure is a conceptually infinite two-dimensional array of lattice cells. As the primary operation on a cell is inspecting its neighbours in no particular order, each cell has pointers to its four neighbours. Each cell has a status field that can encode the values FREE, OCCUPIED, UNTRIED, BLOCKED, and REACHABLE. The cells are initialized to FREE. At any given time, the cells marked OCCUPIED represent the current polyomino. Generating polyominoes lends itself naturally to a recursive approach, since all k -ominoes can be generated by adjoining one new cell in each possible position to each $(k - 1)$ -omino. The base case is simply the 1-omino. Note that there are actually two fixed 1-iamonds, as the triangle lattice has two types of triangles that are 180 degree rotations of each other. As a simple optimization, we maintain a list of cells that are adjacent to OCCUPIED cells and mark such cells as UNTRIED. When a cell is set to OCCUPIED, we need only update the list by removing the newly OCCUPIED cell and adding any of its neighbours that were previously FREE and are now UNTRIED. This removes the need to exhaustively scan the lattice to find cells that are candidates to become OCCUPIED at each recursive call.

The naive approach is summarized in Algorithm 1.

Algorithm 1 redelmeier-naive(*lattice*, *untriedList*, *depth*)

```

if depth == number of cells in completed polyomino then
  output completed polyomino
else
  for each cell c in untriedList do
    set c to OCCUPIED in lattice
    newUntriedList = EMPTY
    for each FREE cell d adjacent to cell c do
      set d to UNTRIED
      add d to newUntriedList
    end for
    redelmeier-naive( lattice, (untriedList - c) ∪ newUntriedList, depth + 1 )
    set c to UNTRIED in lattice
    for each cell d in newUntriedList do
      set d to FREE
    end for
  end for
end if

```

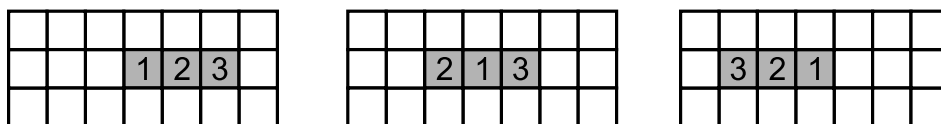


Figure 4.2: Failure of uniqueness under translation.

As stated so far, this approach will produce many copies of each fixed polyomino. There are two sources of redundancy to be removed.

The first is *translational invariance*; that is, we must avoid producing two copies of the same shape that differ only in their position in the lattice. For example, if we number the cells by the order in which they became occupied, we could produce this 3-omino in at least the three ways shown in Figure 4.2.

We avoid this by marking all cells on rows below the origin (the location of the starting 1-omino) and all cells directly to the left of the origin as BLOCKED, excluding them from ever becoming OCCUPIED. This forces any polyomino into a canonical position on the lattice, with the leftmost occupied cell on the bottom row always located at the origin as shown in Figure 4.3.

The second source of redundancy is producing the same shape by adding cells in two

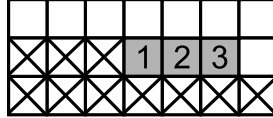


Figure 4.3: Addition of BLOCKED cells.

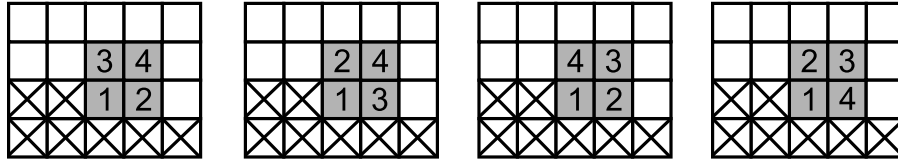


Figure 4.4: Multiple orders of generation.

or more different orders. For example, the square 4-omino can be produced in the various possible orders in Figure 4.4.

If we consider the structure of recursive calls during the algorithm as a tree with each node having a polyomino of size equal to its distance from the root plus one, our goal is to ensure that no two nodes are associated with the same polyomino. If we can ensure that for any node, the subtrees rooted at any two of the node's children have no polyomino in common, then applying that property recursively will produce the desired result.

This can be accomplished by a simple modification. After finishing the recursive call with cell c OCCUPIED, set c to the REACHABLE state, which will prevent it from becoming OCCUPIED on any of the remaining recursive calls from the current level. Once the untried list is exhausted, undo the change to REACHABLE. Then if the untried list consists of c_1, c_2, \dots, c_n , the i^{th} recursive subtree created when c_i is set to OCCUPIED will differ from the contents of all other subtrees at this level because:

- for all $1 \leq j < i$, everything in the j^{th} subtree has c_j OCCUPIED, but no descendant of the i^{th} recursive call will ever have c_j OCCUPIED
- for all $i < j \leq n$, every descendant of the i^{th} recursive call has c_i OCCUPIED, but nothing in the j^{th} subtree will ever have c_i OCCUPIED

Figure 4.5 illustrates the recursive tree showing generation of all fixed 3-ominoes and

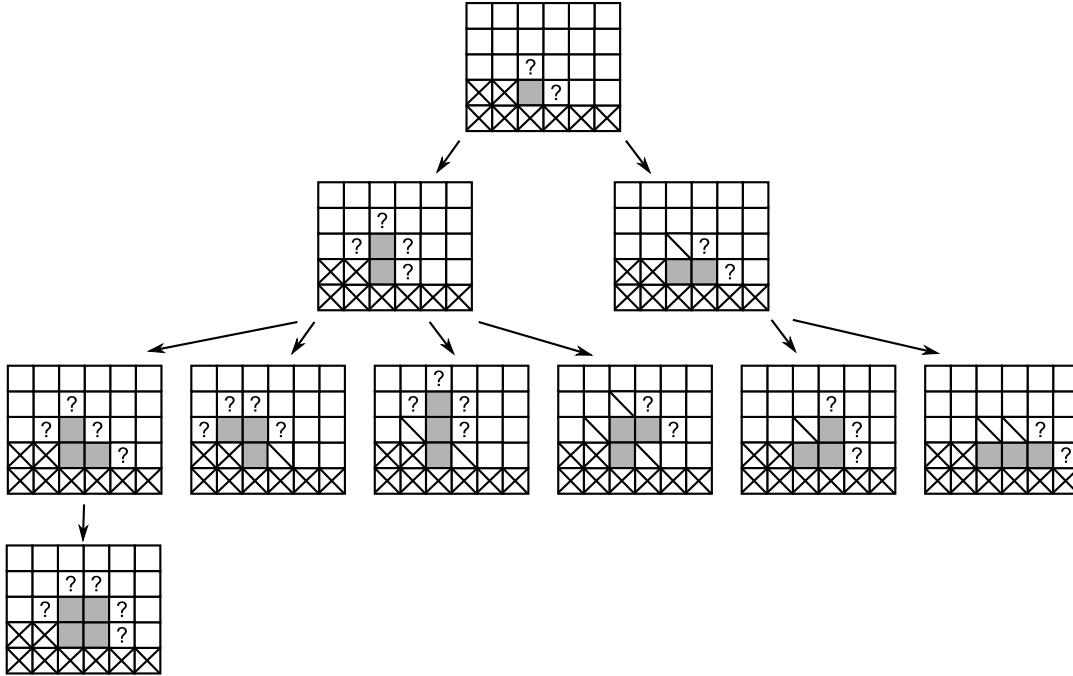


Figure 4.5: Finished algorithm execution up to 3-ominoes, and the 2x2 4-omino.

the only path in the tree leading to the 2×2 square polyomino. Recursive calls from each node are ordered in sequence from left to right.

Pseudocode for the completed algorithm is shown in Algorithm 2.

4.2.1 Implementation

The dominating feature of Redelmeier’s algorithm is the large number of recursive calls. As the amount of work done at each node is relatively small, the overhead of making the calls is significant. Deep-copying the lattice data structures at every call would be a huge expense, hence the extra manipulations to put the lattice back to its previous state before returning. Taking advantage of the fixed recursive stack depth to convert the algorithm to an iterative implementation with static data structures, despite some extra bookkeeping, sped up the execution time of the enumeration process by approximately a factor of two. The untried list has some interesting properties that allow the implementation to pass modified versions of the list without copying. We can conservatively allocate a contiguous array large enough to hold every cell in the lattice, which will always be sufficient to hold

Algorithm 2 *redelmeier(lattice, untriedList, depth)*

```
if depth == number of cells in completed polyomino then
  output completed polyomino
else
  for i = 1 to length of untriedList do
    c = untriedList[i]
    set c to OCCUPIED in lattice
    newUntriedList = EMPTY
    for each FREE cell d adjacent to cell c do
      set d to UNTRIED
      add d to newUntriedList
    end for
    childUntriedList = untriedList[i + 1 . . . end]  $\cup$  newUntriedList
    redelmeier( lattice, childUntriedList, depth + 1 )
    set c to REACHABLE in lattice
    for each cell d in newUntriedList do
      set d to FREE
    end for
  end for
for each cell c in untriedList do
  set c to UNTRIED
end for
end if
```

the entire untried list. At any level, the untried list being passed to a child consists of part of the tail end of the current untried list plus zero or more elements appended onto that list. By maintaining the list purely as a start and end index into this large array, we can append to the list by writing past the end index and pass a subarray by manipulating the indices passed to the child call.

4.3 Parallelization

Redelmeier's algorithm can be easily parallelized with negligible overhead by observing that if we choose a level in the tree, any subtree rooted at that level is completely independent of any other subtree rooted at that level. Therefore, we can produce any number of independent subjobs simply by picking a level with a large enough number of nodes and farming out one or several of those nodes to each job. The only coordination required is adding up the totals at the end. Since the enumeration follows a fixed order the subjobs

can simply be given a list of numbers (k_1, k_2, \dots, k_n) , run their own enumeration of the entire tree above the chosen level, and only explore the descendants of the $k_1^{st}, k_2^{nd}, \dots, k_n^{th}$ nodes encountered in sequence at that level.

4.4 Conversion to Boundary Word Representation

Given a shape on the lattice, we convert it to a boundary word by starting at the origin cell, which is guaranteed to be on the boundary as it borders on a BLOCKED cell, and walk around the edge building the string one character at a time. This representation will not capture any holes in the interior of the shape.

4.5 Canonicalization

At this stage we have the boundary word representation of a fixed polyform, possibly with internal holes being ignored. As the enumeration will produce one copy of a shape for each of its asymmetric rotated or reflected orientations, we need to discard all but one copy to consider each free polyform only once. In some applications it might be acceptable to keep the extra copies and divide out the redundancy in the results, but since we expect the tiling tests to be the bottleneck it is preferable to only test a shape once. The simplest solution is to define a canonical form for boundary words, and ensure that exactly one copy will be identified as canonical.

The first step is to canonicalize an individual boundary word, since it has no fixed starting point. This is a classic problem in formal language theory, and both the usual solution—select the starting point that produces a string lexicographically less than or equal to any other starting point—and algorithms for computing it are well known [5]. My implementation only uses the brute force quadratic time approach to find the lexicographically least starting point, as the input lengths involved are relatively short.

Next, we can observe that if we apply every possible rotation and every possible reflection and canonicalize each resulting boundary word, we produce every possible fixed

version of the shape. If the shape has symmetries, some of these will be identical, but then they will only be produced once, since Redelmeier’s algorithm ensures that each fixed polyform is only produced once. Therefore, we can select the lexicographically least aspect as the canonical form. To test whether the current shape is in canonical form, we simply apply rotations and reflections and look for a resulting boundary word in canonical form that is lexicographically strictly less than the current shape’s boundary word.

This approach will not quite produce the correct list of free polyforms, because symmetries involving internal holes will be missed. Once the shapes with holes are identified and discarded, the resulting list of free polyforms without holes will be correct.

4.6 Hole Detection

By our definition of tiling, no shape with an interior hole (i.e., whose boundary is not a single closed curve) can ever tile the plane. Therefore, we need to detect and discard any polyform containing a hole. This is accomplished by a straightforward flood fill on the lattice starting from outside the shape and filling any cell that is not OCCUPIED. The implementation of Redelmeier’s algorithm maintains a bounding rectangle for the current shape to limit the region of the lattice that needs to be flood filled. If the number of cells marked by the flood fill is a , the number of cells within the bounding rectangle is b , and the size of the polyform is c cells, then the shape contains a hole if and only if $a < b - c$. Because the bounding rectangle is changing during recursive generation and holes that exist partway through the recursion tree might be filled in some of their descendants, it is most practical just to do this step at the leaves (finished shapes).

4.7 Enumeration of Polysnakes

Producing polysnakes requires a significantly different approach than producing polyforms. The idea of recursively generating polysnakes from shorter completed polysnakes is not useful, as a completed polysnake already forms a closed loop. The boundary word

inherently describes a shape without internal holes as long as it does not intersect itself, so the hole detection step is replaced by the process of avoiding self-intersection. As the corners of a polysnake do not necessarily lie on a discrete grid, some floating-point math will be necessary to determine their relative locations.

The basic approach to enumerating polysnakes with a given perimeter is to begin at the origin in the plane and recursively concatenate each possible character to the end of the current string. To determine if a character can be legally added to the current string, we must check that the new edge does not touch or intersect any existing edge or vertex (note that the negation of the preceding character is always invalid as a next character), and that the final edge returns to the origin. We can also quickly check that the distance from the resulting new vertex to the origin is not greater than the number of unit-length edges still to be added, to avoid partial strings that cannot return to the origin. The resulting boundary must be wound clockwise; counterclockwise boundaries are detected by a signed area check and discarded.

We define a canonical form for polysnakes in exactly the same way as we did for hole-free polyforms in Section 4.5, with the additional complication that this enumeration algorithm will produce k copies of a fixed polysnake of length k —one for each starting point. With a small modification the enumeration can be made to only produce one copy of each fixed polysnake; specifically, the lexicographically least starting point. Each recursive call takes a parameter $q \geq 0$, indicating the length of the longest suffix of the current word that is equal to a prefix of the current word. The next character cannot be lexicographically less than the $q + 1^{\text{st}}$ character, or the new suffix would be lexicographically less than the entire word. The value of q passed to descendent calls is $q + 1$ if the new character was equal to the $q + 1^{\text{st}}$ character, and 0 otherwise.

With only one copy of each fixed polysnake being generated, we can apply the same canonicity test as we did for polyforms to discard all but one copy of each free polysnake: generate all rotated and reflected aspects of the current shape, and discard the current shape if any of the least cyclic shifts of other aspects are lexicographically less.

Chapter 5

Implementation: Tiling Tests

Given a boundary word as output from the enumeration described in the previous chapter, my program proceeds to determine some of its tiling properties. The first step is to apply the criteria from Section 2.1.4 to determine whether the shape tiles isohedrally. To produce results that can be directly compared to past work, I separately report whether the shape tiles by translation (Criterion 4), tiles by the Conway Criterion (Criterion 1), or tiles by one of the other seven criteria.

If the shape is not an isohedral tiler then it must either not tile the plane, or be an anisohedral tiler, or be aperiodic. To determine which of these is the case, I begin exploring the ways in which the shape can be surrounded by copies of itself to a depth of one layer, then two layers, and so on. The data from these “surrounds” is used to produce all connected hole-free patches of m tiles that could potentially participate in a tiling, first for $m = 2$, then $m = 3$, and so on. If the shape is k -anisohedral and not unbalanced, then at least one patch of k tiles will pass the isohedral tiling test (used here as a subroutine) and no patch of m tiles will pass the test for any $m < k$. If the shape is unbalanced, this approach will incorrectly report it as having a higher isohedral number than it actually has. If at some point it is determined that the shape cannot be surrounded to some number of layers, it is reported as a non-tiler.

In the unlikely event that the program stumbles across an aperiodic shape, it will

not find any patch of tiles that passes the isohedral tiling test and not find any limit on the depth to which the shape can be surrounded by copies of itself. Since the program operates in finite time and space, it will stop at some built-in limit and report the shape's properties as "unknown" for investigation by hand.

5.1 Isohedral Tiling

My initial implementation of tests to determine if a boundary word represents an isohedral tiler began with the discovery that the case of testing polyomino boundary words for tiling by translation has a number of existing published algorithms. The study of this case began with a paper by Beauquier and Nivat in 1991 [1], which explicitly articulated and proved the case of Criterion 4 from Section 2.1.4 on polyomino boundary words. In 2003, Gambini and Vuillon [8] presented an algorithm to test this condition in $O(n^2)$ time, where n is the boundary length, improving on the naive $O(n^4)$ approach. This was improved by Brlek and Provençal in 2006 [6] to an $O(n)$ time algorithm by a more complex approach.

Rhoads [24] has also implemented a boundary-based test for the Conway Criterion (Criterion 1), but it appears my program is the first time the other seven criteria have been explicitly implemented. Previous work has relied on building up to a patch that tiles by translation, and then analyzing the resulting symmetries to determine the isohedral number.

As a first step, I implemented the quadratic-time algorithm of Gambini and Vuillon for Criterion 4. It is not obvious that the linear-time algorithm would actually be faster on the relatively small input lengths under consideration, or that any speedup would be worth the difficulty of implementing a more sophisticated algorithm. In any case, this algorithm was eventually replaced in favour of a different approach as described later in this section.

The design of Gambini and Vuillon's algorithm provided the template for testing the

other eight criteria, so it is useful to give a high-level overview. We can observe that if the input passes Criterion 4, every character in the string is part of some translational pair. Therefore, we can start with the first character in the input string (since the string is circular, this is effectively arbitrary) and look for all occurrences of its negation. For each such occurrence, we extend the translational pair in each direction until it is maximal. Note that since the correspondance under translation is negation *and string reversal*, the two components extend in opposite directions. Geometrically, this corresponds to aligning the two chosen edges and determining how much of the boundary lines up on each side.

Any maximal translational pair found in this manner fixes two components of a potential decomposition, leaving two substrings to be tested to see if they fit the remaining pieces of the criterion. In this case neither substring can be empty, because otherwise the boundary word would contain a substring $a\bar{a}$ for some character a , which is impossible if the boundary word describes a simple polygon. If we call these two substrings X and Y , we need to test for the cases where $X = \bar{Y}$ or $X = A \cdot B$ and $Y = \bar{A} \cdot \bar{B}$ for some strings A and B . We can note as a first check that in both cases $|X| = |Y|$, and quickly reject any strings with $|X| \neq |Y|$. Then we can test for $X = \bar{Y}$ in the straightforward way, and test the second case using the observation that there exist A and B with $X = A \cdot B$ and $Y = \bar{A} \cdot \bar{B}$ if and only if $\bar{X} = \bar{B} \cdot \bar{A}$ is a substring of $Y \cdot Y = \bar{A} \cdot \bar{B} \cdot \bar{A} \cdot \bar{B}$, a simple pattern matching problem.

To summarize the approach, we select a part of the decomposition that must exist and identify each possible pair of substrings that fit that part. For each possible case making up the remaining parts of the decomposition, we check if the remaining substrings fit those parts. That is, this is a purely top-down approach. For many of the criteria, the possibility of any piece of the decomposition being empty creates two “major” cases (e.g., a translation pair exists, or a translation pair does not exist) that are separate at the top level, and several “minor” cases once the first pieces of the decomposition are fixed. Aside from the complexity of implementing and verifying all of the various cases,

this approach was conceptually straightforward.

However, when I began using this test as a subroutine in the identification of k -isohedral tilings by checking many different patches of tiles for isohedral tiling it became apparent that it was becoming a bottleneck. The immediate observation was that the lowest-level primitives, such as “is this substring a palindrome” or “are there a pair of glide reflected segments extending from these two positions” could be called repeatedly on the same positions in the input, duplicating huge amounts of work. This situation lends itself naturally to a dynamic programming approach.

As there are only a linear number of possible maximal palindrome substrings (n odd palindrome centers and n even palindrome centers), a quadratic number of possible translational or glide reflected pairs, and a linear number of rotated pairs (since they must share a common endpoint), it is possible with a relatively modest amount of work to build a library of every piece that might participate in any decomposition, indexed by their various starting points. Although some entries in the library might never be used, doing this work up front reduces the later top-down steps to a series of array lookups and eliminates the duplicated effort completely. The top-down search for a decomposition by each criterion is conceptually the same as before, but now it consists of iterating through the library or querying for pieces with specific starting points instead of actually inspecting the string.

A further optimization (albeit a minor one, as the expected case is for most input to fail the test) is to arrange the criteria so that parts of the library will not be built until criteria that do not require those parts have already been checked. The order is as follows:

- build library of translation pairs
- test Criterion 4
- build library of palindromes
- test Criterion 1
- build library of rotated segments, as appropriate for the current boundary language
- test Criteria 7, 8, 9
- build library of glide reflected segments
- test Criteria 2, 3, 5, 6

Empirically, this revised approach sped up the isohedral test by a factor of about three.

5.2 Anisohedral and Non-tiling

5.2.1 Basic Approach

Having eliminated the possibility that a given shape tiles isohedrally, we are left with the task of either proving that it does not tile, or determining that it is k -anisohedral for some $k \geq 2$, or reporting that the shape exceeds the limits of implementation's ability to determine tiling status. My implementation finds the size of the minimal isohedral repeating unit as defined in Section 2.1.1 instead of the actual isohedral number. For brevity of explanation, I will write as though I were properly testing shapes for their isohedral number, with the understanding that a few unbalanced shapes will be reported incorrectly.

A systematic proof technique to prove non-tiling is to start with a single tile and explore all the ways in which it can be surrounded by copies of itself. If it cannot be surrounded, then it does not tile the plane. If it can be surrounded in one or more ways, then take each such surround and explore all the ways in which it can be surrounded by another layer of copies of the original tile. If no such second-layer surrounds exist, then the tile does not tile the plane. This can be repeated indefinitely, to the limits of the time and space available to the program. The maximum number of surrounds is related to but not the same as the Heesch number, since I consider a shape to be surrounded once every edge is covered, while the definition of corona used in Heesch numbers requires that vertices also be covered.

Any shape that does not tile the plane must eventually be unable to continue adding surrounds, or we could invoke the Extension Theorem [11, Theorem 3.8.1], which states that if a set of tiles can cover arbitrarily large circular discs then it tiles the plane. Since the highest known Heesch number is currently 5, it is unlikely that this process will

continue into dozens of surround layers for a non-tiler. Of course, a shape that tiles the plane can continue this process forever. Therefore, we need to be simultaneously working on a proof of anisohedral tiling, which is simply exploring progressively larger patches of tiles until we find a patch that is itself an isohedral tiler. Fortunately, these two procedures work hand-in-hand: if the shape admits an anisohedral tiling some series of surrounds will eventually include any patch of that tiling as a subset, at least one of which will be an isohedral repeating unit. So by exploring surrounds we are also generating all patches of tiles that need to be tested. Recall that a “patch” is the union of a finite collection of tiles assembled without gaps or overlap. Of course, surrounds grow by an entire layer of tiles and we want to test progressively larger patches in increments of one tile, so surrounds must be disassembled into various subsets of their component parts for testing. Since we explore all possible surrounds we can be assured that if an isohedral repeating unit exists, the single tile at the center of the surrounds is in the middle of the patch. That is, in the worst case we need only $\text{floor}(k/2)$ surrounds to explore all patches of size k .

To summarize, the initial concept of the non-tiling and anisohedral test is as follows:

```

generate all 1-surrounds
if no 1-surrounds exist then
    return Heesch number 0
end if
use 1-surrounds to test all patches of 2 tiles for isohedral tiling
if isohedral patch found then
    return 2-anisohedral
end if
use 1-surrounds to test all patches of 3 tiles for isohedral tiling
if isohedral patch found then
    return 3-anisohedral
end if
extend 1-surrounds to generate all 2-surrounds
if no 2-surrounds exist then
    return can't be surrounded 2 times
end if
use 2-surrounds to test all patches of 4 tiles for isohedral tiling
if isohedral patch found then
    return 4-anisohedral
end if
use 2-surrounds to test all patches of 5 tiles for isohedral tiling

```

```

if isohedral patch found then
  return 5-anisohedral
end if
extend 2-surrounds to generate all 3-surrounds
if no 3-surrounds exist then
  return can't be surrounded 3 times
end if
etc.

```

If the shape is aperiodic, this algorithm will run forever as the shape can be surrounded infinitely many times but no patch of tiles will ever be isohedral. In practice, this simply means that once built-in limits are reached, the implementation will stop and report the shape for further examination by hand.

Although this approach is correct for all non-tilers and anisohedral tilers, it suffers from fatal efficiency problems in practice. It can be optimized to some extent by noting that empirically there are few shapes with isohedral numbers above 3 but enormously many non-tilers. Therefore it is better to build 2, 3, and even 4-surrounds before starting to test patches of size 4 and above, on the assumption that non-tiling is vastly more probable than tiling. However, the central problem is that some shapes admit a combinatorial explosion of surrounds—so many that even building 2-surrounds can be impractical. Perhaps the most startling example is a 6-hex (Figure 5.1), which admits nearly 1.5 *million* distinct 1-surrounds. The combinatorial explosion is made possible by the fact that the shape can be placed with either end of the “hook” touching almost any spot on the boundary in either of two reflected aspects, and it only takes 10 locations with 4 options each that don’t interfere with the other locations to reach 2^{20} . It can be surrounded at least twice, and the evidence using later optimizations shows that it is in fact non-tiling, but this approach is unable to determine the maximum number of surrounds in any reasonable amount of time.

Before going into the optimizations that escape this situation, and other issues that arise along the way, I will present some implementation details of this basic approach.

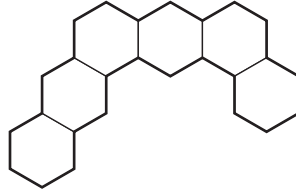


Figure 5.1: 6-hex problem case.

5.2.2 Boundary Merge Implementation

The main primitive operation in building surrounds is taking two boundary words with a specified edge in each, and either returning the boundary of the shape produced by placing the shapes beside each other with a position and orientation that aligns the specified pair of edges, or reporting that the operation would cause an overlap or an internal hole.

Algorithm 3 `merge(boundary1, offset1, boundary2, offset2)`

compute the rotation that aligns the specified edges, apply it to one of the boundary words
 extend the match in both directions along the boundary to encompass edges that coincide with each other
 check that the next step past the coincident boundary edges does not step into the interior of the other shape
 removing the coincident pieces, splice together the two boundary words
 check that the result is a simple polygon
return resulting boundary word

The simple polygon check will detect any internal holes or overlaps in the resulting boundary word. For the polyform case, this can be done quickly by clearing a quadratic-size bitmap (which can be performed quickly on modern hardware) and performing a linear walk following each step of the boundary word and marking locations visited on the lattice. Any non-simple polyform boundary must re-visit some lattice cell. For the polysnake case, I take advantage of the observation that the new boundary word is the concatenation of pieces of two boundary words that were each known to be simple polygons. Therefore, any part of the result that intersects itself must be an intersection between a vertex or edge from one input word and a vertex or edge from the other input word. This greatly limits number of pairs that must be considered in a naive quadratic-time simple polygon test, speeding it up to the point where I chose to forgo the overhead

of the data structures for a proper $O(n \log n)$ time simple polygon test.

5.2.3 2-tile Patch Construction

The first step in building surrounds is to compile a list of all valid ways a second tile can be placed touching a first fixed-orientation tile. Since any boundary offset in the second tile can potentially be paired with any boundary offset in the first tile in either an original or reflected aspect, this information can be stored in two matrices. The second matrix, which tracks placements of the reflected aspect of the tile, can be omitted if the tile has a symmetry of reflection.

However, simply trying each possible pairing and storing the result in the corresponding matrix entry has a fair amount of duplication of effort. If a particular pairing causes five pairs of edges to coincide at the meeting point, there will be five matrix entries that represent exactly the same resulting shape. So instead, I store a linked list of all 2-patches that represent the alignment of distinct pairs of boundary substrings, and allow multiple matrix entries to reference the same patch. The concept is illustrated in Figure 5.2.

The process of building all 2-patches is to find a matrix entry that has not been checked, run the merge subroutine, and update the matrix entry and potentially some neighbours with the result. If there is an edge on the fixed first tile that cannot be paired with any second tile edge in any orientation (for example, the interior of the concavity in Figure 2.20), then we know immediately that no surrounds will be found.

5.2.4 Surround Depth-First Search Implementation

With the library of 2-patches in hand, we can build all possible first-layer surrounds of a fixed starting tile. The basic approach is a depth-first search starting from all 2-patches that cover an arbitrarily chosen starting point on the first tile, progressively trying each option to cover the next remaining uncovered edge of the first tile going clockwise around the boundary. Each added tile is attached to the growing patch with the merge operation, which also verifies that the new tile does not create any overlap or holes. When a tile is

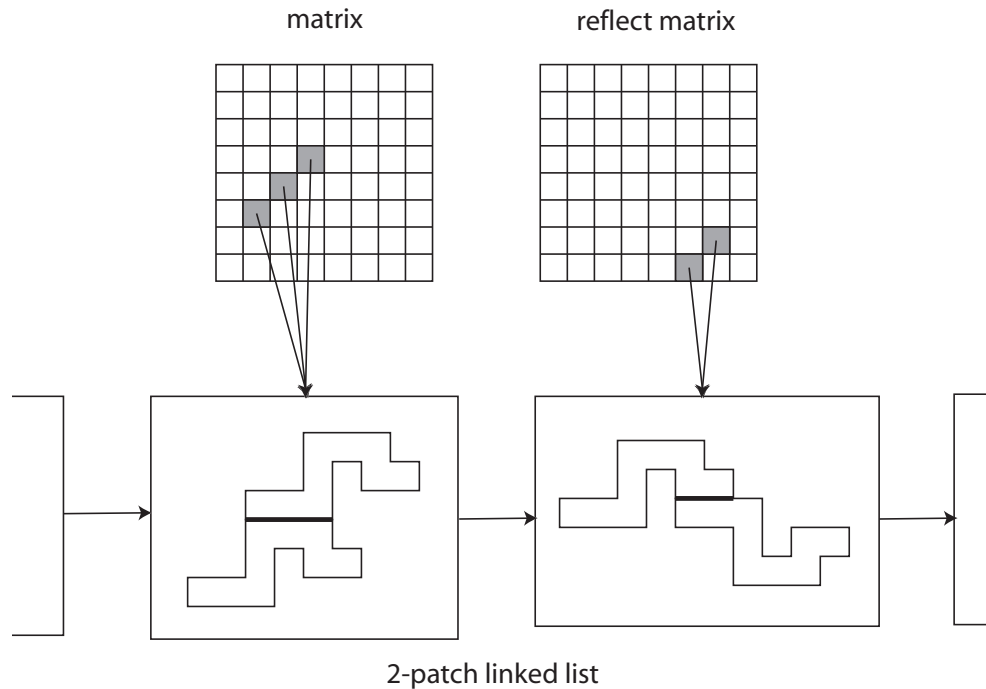


Figure 5.2: 2-patch data structures.

placed that finishes covering the edges of the first tile, the completed surround is copied to a list of surrounds for later use.

This process is generalized to be recursive, with the fixed first tile shape replaced by the outer boundary of a k -layer surround to be extended by another layer of tiles into a $k + 1$ -layer surround.

5.2.5 Optimizations

I am indebted to Joseph Myers for some of the optimizations that avoid the exponential explosion of surrounds.

The first heuristic to apply is that many tiles cannot be surrounded even once, because some region of the boundary prevents it. The process of building 2-patches will catch edges that cannot be covered at all, but another case that appears to be common is where every individual edge can be covered but only by placing a second tile in ways that will

prevent some neighbouring edge from ever being covered. To catch the vast majority of instances of this case, I build a list for every 2-patch p of the 2-patches that can be merged with p to cover the next edge clockwise along the base tile, called *successors*. Then any 2-patch that has no successors cannot participate in a surround, so it can be marked as invalid. Likewise, any 2-patch that has no predecessor (checked by a reverse lookup through lists of successors) is marked invalid. This process can be iterated as long as patches are being marked invalid, as a newly invalid patch might be the last available successor for another patch.

The next observation is that even after the first heuristic, some 2-patches participate in many different surrounds and some participate in none. It is useful to identify the latter kind before generating the exhaustively complete list of surrounds, to avoid considering such “dead end” 2-patches. To this end, I run the depth-first search starting from any valid 2-patch not known to be part of a surround and stop when one surround is found. Every 2-patch that participates in that surround is flagged as participating in a surround. If no surround is found, the starting 2-patch is marked as invalid to remove it from future consideration.

At this point, my algorithm has produced a powerful piece of data: a list of the relative positions in which any two adjacent tiles in any tiling cannot be placed, because that relative configuration does not occur in any 1-surround. This list can be leveraged when building surrounds to do more than just limit the possible ways the base tile can relate to tiles in the 1-surround: it can be used to detect if any two touching tiles in the 1-surround are in a configuration that will not be extended, and therefore represent a dead end. To implement this optimization, I keep an *atlas* for every patch of what edge of the underlying tile each edge on the boundary corresponds to, and whether it comes from a direct or reflected aspect of the tile. When performing a merge operation, every pair of edges being matched in the merged portion of the boundaries have their atlas entries looked up in the list of valid relative configurations. If the lookup shows that the configuration does not occur in any 1-surround, then the merge fails, avoiding the dead

end.

With these optimizations in place, I go ahead and build all of the 1-surrounds that pass the various tests. When extending 1-surrounds into 2-surrounds, all of the previous optimizations can be generalized and re-applied. As a further extension, I keep a reference count for each 2-patch of the number of 1-surrounds in which it participates. If a 1-surround cannot be extended into a 2-surround, I decrement the reference counts of the 2-patches that make up the 1-surround. If a reference count hits zero, it represents a proof that the 2-patch cannot be extended all the way into a 2-surround and should therefore be discarded from consideration. Whenever a patch is discarded in this way, I throw out all existing surrounds and restart from the calculation of successors. This is a risky step, as it is based on the assumption that eliminating a 2-patch that survived so long will have a significant impact on the surrounds that have already been built. So far, this assumption has proved to work reasonably well in practice.

In particular, the problematic 6-hex mentioned above in Figure 5.1 is very quickly identified as a non-tiler. Part of the inner side of the “hook” can only be covered in a relatively small number of ways, which in turn constrain how the ends of the hook can be covered, which in turn eliminate most of the combinatorial explosion of possibilities for covering the outer side of the hook.

At this point, the process of optimizing the surround-building code is simply to find a shape that takes a long time to test, investigate where the exponential explosion of possible surrounds happens, and identify an optimization that correctly determines the outcome without exploring all of the possibilities. As my implementation is still much slower than that of Myers [21], this process could continue for many more iterations.

Chapter 6

Results

The completed implementation consists of about 3500 lines of C code. During development and testing, any discrepancies with Joseph Myers' results [21] were carefully investigated and in every case the previous work was found to be correct.

Because I count free polyforms by a canonicity check that only considers the outer boundary and not any internal holes, my count of free polyforms with holes will be incorrect in some cases. Since I am concerned only with the tiling properties of polyforms without holes, this does not alter the main results.

The unbalanced 8-hex (see Section 2.1.1) is listed here as 3-anisohedral due to the limitations of the test for a minimal isohedral repeating unit. It is actually 2-anisohedral.

The following tables summarize the results of my program's output, organized for consistency with the presentation of Myers' results. The table columns are as follows:

Area/Perimeter is the area for polyforms, or the perimeter for polysnakes.

Free is the number of free shapes of each size (defined in Section 2.2).

Holes is the number of free shapes that have holes, and so do not tile the plane.

Trans is the number of free shapes that tile the plane by translation (Criterion 4).

Conway is the number of free shapes that do not tile by translation but do tile by the Conway Criterion (Criterion 1).

Iso is the number of free shapes that do not tile by translation or the Conway Criterion but do tile by another isohedral criterion.

Aniso is the number of free shapes that are k -isohedral for some $k \geq 2$. The details for each isohedral number are in a separate table below.

Non-tiling is the number of free shapes successfully identified as non-tiling.

Unknown counts shapes that could not be successfully handled by my implementation.

Time gives the elapsed wall clock time for computing the results in the row, in hour : minutes : seconds format.

The primary result of interest for polyominoes, polyhexes, and polyiamonds is simply that my results agree with Joseph Myers' results, except for the unbalanced 8-hex and the meaning of the count of free polyforms as previously mentioned. This provides an independent confirmation of some existing results. The time taken to compute each row in the table is somewhat variable, as my program is still subject to certain "problem cases" that my existing optimizations do not resolve.

The results for polysnakes are new results. From an enumeration perspective, it is notable that there are no odd-length 4-snakes. This is not unexpected. Considering a 4-snake as a walk from the origin, every step changes the current x and y coordinate by an absolute value of 0, 1, or $\sqrt{2}/2$. Since $\sqrt{2}/2$ and 1 are incommensurate, returning to the origin requires that changes of 1 be paired with changes of -1 , and changes of $\sqrt{2}/2$ with changes of $-\sqrt{2}/2$. Since 5-snakes can describe a pentagon, they have an immediate building block to create odd-length perimeters.

A similar observation can be applied to the zeroes for odd-length 5-snakes tiling by translation. The test for tiling by translation factors the boundary word into pairs of segments that are each others' negations, which is not possible for odd-length words. I

do not have a similar explanation for why no odd-length 5-snakes tile by the Conway Criterion. The fraction of non-tiling shapes among polysnakes appears to be roughly the same as that of polyforms. One 5-snake still remains in the “unknown” column, as it admits a combinatorial explosion of possible surrounds that my program is unable to avoid exploring.

Given the speculation that the relationship between 5-snakes and the Penrose rhombs suggests that 5-snakes might have interesting properties, it is a surprise that the only anisohedral 5-snakes found were 2-anisohedral, while 4-snakes quickly produced 3- and 4-anisohedral examples. The two 4-anisohedral 4-snakes are shown in Figures 6.1 and 6.2.

Table 6.1: Tiling results for polyominoes.

Area	Free	Holes	Trans	Conway	Iso	Aniso	Non-tiling	Unknown	Time
1	1	0	1	0	0	0	0	0	< 00:00:01
2	1	0	1	0	0	0	0	0	< 00:00:01
3	2	0	2	0	0	0	0	0	< 00:00:01
4	5	0	5	0	0	0	0	0	< 00:00:01
5	12	0	9	3	0	0	0	0	< 00:00:01
6	35	0	24	11	0	0	0	0	< 00:00:01
7	108	1	41	60	3	0	3	0	< 00:00:01
8	369	6	121	199	22	1	20	0	< 00:00:01
9	1285	37	213	748	80	9	198	0	00:00:22
10	4655	195	522	2181	323	44	1390	0	00:00:11
11	17074	980	783	5391	338	108	9474	0	00:01:26
12	63600	4663	2712	17193	3322	222	35488	0	00:03:29
13	238604	21487	3179	31881	3178	431	178448	0	00:18:17

Table 6.2: Isohedral numbers for polyominoes.

Area	2-aniso	3-aniso	4-aniso	5-aniso	6-aniso
8	1	0	0	0	0
9	8	0	1	0	0
10	41	3	0	0	0
11	89	18	1	0	0
12	214	6	2	0	0
13	406	24	0	1	0

Table 6.3: Tiling results for polyhexes.

Area	Free	Holes	Trans	Conway	Iso	Aniso	Non-tiling	Unknown	Time
1	1	0	1	0	0	0	0	0	< 00:00:01
2	1	0	1	0	0	0	0	0	< 00:00:01
3	3	0	3	0	0	0	0	0	< 00:00:01
4	7	0	6	1	0	0	0	0	< 00:00:01
5	22	1	12	9	1	0	0	0	< 00:00:01
6	82	1	36	39	1	1	4	0	00:00:06
7	333	2	60	197	33	4	37	0	00:03:47
8	1448	13	209	721	88	36	381	0	00:02:56
9	6573	68	387	2717	611	71	2717	2	00:16:21

Table 6.4: Isohedral numbers for polyhexes.

Area	2-aniso	3-aniso	4-aniso	5-aniso	6-aniso
6	1	0	0	0	0
7	3	1	0	0	0
8	26	8	2	0	0
9	66	4	1	0	0

Table 6.5: Tiling results for polyiamonds.

Area	Free	Holes	Trans	Conway	Iso	Aniso	Non-tiling	Unknown	Time
1	1	0	0	1	0	0	0	0	< 00:00:01
2	1	0	1	0	0	0	0	0	< 00:00:01
3	1	0	0	1	0	0	0	0	< 00:00:01
4	3	0	2	1	0	0	0	0	< 00:00:01
5	4	0	0	4	0	0	0	0	< 00:00:01
6	12	0	8	4	0	0	0	0	< 00:00:01
7	24	0	0	21	2	0	1	0	< 00:00:01
8	66	0	24	32	10	0	0	0	< 00:00:01
9	160	1	0	111	22	6	20	0	00:00:02
10	448	4	62	200	54	25	103	0	00:00:04

Table 6.6: Isohedral numbers for polyiamonds.

Area	2-aniso	3-aniso	4-aniso	5-aniso	6-aniso
9	5	0	1	0	0
10	23	1	1	0	0

Table 6.7: Tiling results for 4-snakes.

Perimeter	Free	Trans	Conway	Iso	Aniso	Non-tiling	Unknown	Time
4	2	2	0	0	0	0	0	< 00:00:01
5	0	0	0	0	0	0	0	< 00:00:01
6	6	6	0	0	0	0	0	< 00:00:01
7	0	0	0	0	0	0	0	< 00:00:01
8	59	38	7	4	0	10	0	00:01:54
9	0	0	0	0	0	0	0	< 00:00:01
10	695	227	119	103	1	245	0	00:03:34
11	0	0	0	0	0	0	0	00:00:02
12	12198	1563	1488	994	49	8104	0	00:08:40

Table 6.8: Isohedral numbers for 4-snakes.

Perimeter	2-aniso	3-aniso	4-aniso	5-aniso	6-aniso
10	1	0	0	0	0
11	0	0	0	0	0
12	46	1	2	0	0

Table 6.9: Tiling results for 5-snakes.

Perimeter	Free	Trans	Conway	Iso	Aniso	Non-tiling	Unknown	Time
4	2	2	0	0	0	0	0	< 00:00:01
5	2	0	0	1	0	1	0	< 00:00:01
6	10	10	0	0	0	0	0	< 00:00:01
7	15	0	0	9	0	5	1	00:00:01
8	124	73	13	12	0	26	0	00:00:20
9	352	0	0	64	10	278	0	00:04:29
10	2393	586	267	296	4	1240	0	00:03:07
11	9948	0	0	701	15	9232	0	01:06:24

Table 6.10: Isohedral numbers for 5-snakes.

Perimeter	2-aniso	3-aniso	4-aniso	5-aniso	6-aniso
9	10	0	0	0	0
10	4	0	0	0	0
11	15	0	0	0	0

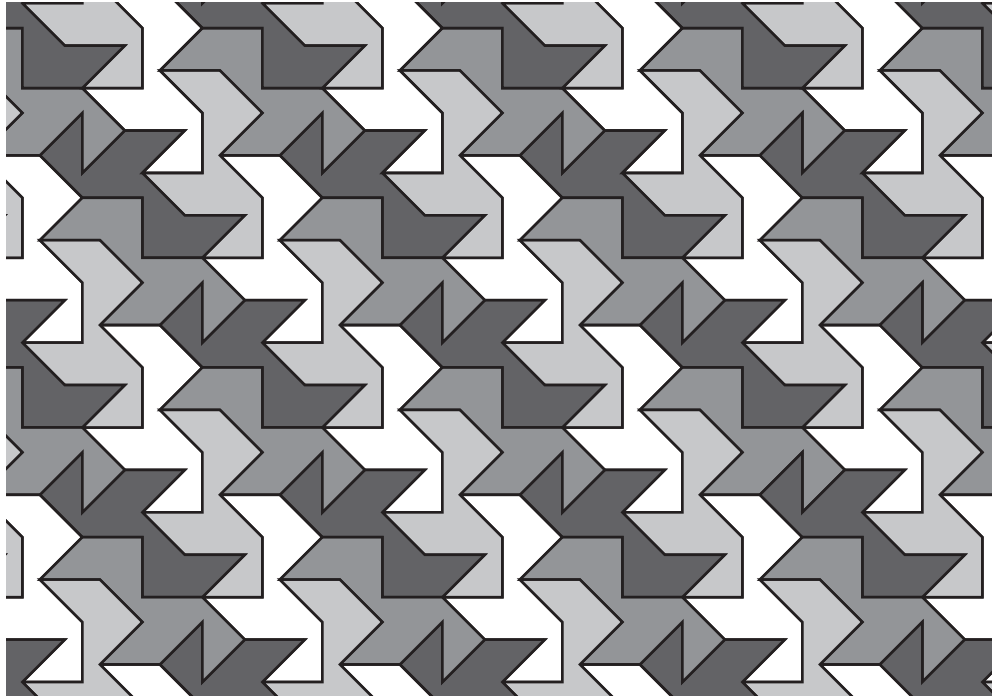


Figure 6.1: The 4-anisohedral 4-snake represented by 013124725065, shown in a 4-isohedral tiling with transitivity classes indicated by shading.

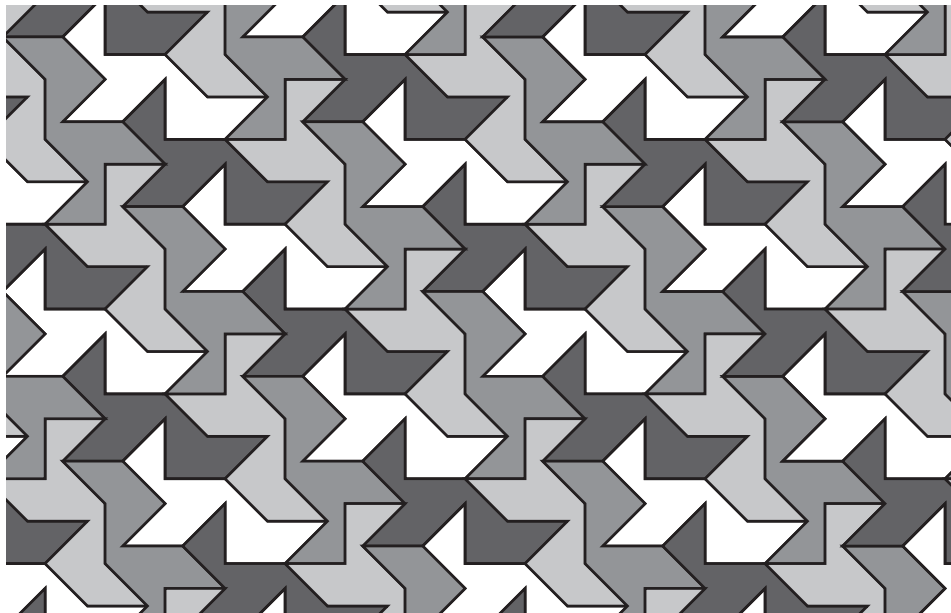


Figure 6.2: The 4-anisohedral 4-snake represented by 010346347572, shown in a 4-isohedral tiling with transitivity classes indicated by shading.

Chapter 7

Conclusions and Future Work

My research has accomplished the following objectives:

- Demonstrated that a purely boundary-based approach can be implemented to determine isohedral numbers or prove non-tiling.
- Identified and corrected deficiencies in Heesch's isohedral tiling criteria.
- Independently confirmed Myers' previous experimental results up to 13-ominoes, 9-hexes, and 10-diamonds.
- Extended the search for shapes with interesting tiling properties to a new, previously unexplored class of shapes.
- Discovered two new 4-anisohedral shapes.

Since the time required to run the tests on a class of shapes scales exponentially with the growth of the perimeter or area parameter, there is always the option of extending the results further by dividing the work among more processors or simply waiting longer. For polysnakes, there is also the possibility of investigating 6-snakes, 7-snakes, etc. There is no evidence to suggest in advance that those new classes of shapes will or will not be interesting.

A different future direction arising from my implementation of the 9 isohedral tiling criteria is to ask whether there is a set of necessary and sufficient boundary criteria for 2-anisohedral tiling, and then 3-anisohedral tiling, and so on. Since the boundary of a shape encodes all of the information that determines its tiling properties, in principle such criteria should exist. If the explicit description of such criteria is possible (and practical), the process of generalizing them to k -anisohedral shapes for progressively larger k could finally settle the question of whether there is a bound on isohedral numbers.

The surround-building framework also provides a base that could be leveraged to investigate Heesch numbers. Since the most recent progress in finding shapes with high Heesch number was also made through a computer search, there is an opportunity to apply different approaches and new optimizations to continue that search.

With the ever-increasing complexity of the shapes and tilings being investigated, an experimental approach to tiling theory provides a useful supply of data and examples to fuel the flashes of inspiration that may someday allow the many unsolved problems to be resolved.

List of References

- [1] Danièle Beauquier and M. Nivat. On translating one polyomino to tile the plane. *Discrete Comput. Geom.*, 6:575–592, 1991.
- [2] Robert Berger. The undecidability of the domino problem. *Memoirs Amer. Math Soc.*, 66, 1966.
- [3] John Berglund. An unbalanced tile. <http://www.angelfire.com/mn3/anisohedral/unbalanced.html>.
- [4] John Berglund. Is there a k -anisohedral tile for $k \geq 5$? *American Mathematical Monthly*, 100(6):585–588, 1993.
- [5] Kellogg S. Booth. Lexicographically least circular substrings. *Inf. Process. Lett.*, 10(4/5):240–242, 1980.
- [6] Srećko Brlek and Xavier Provençal. On the problem of deciding if a polyomino tiles the plane by translation. In *Proceedings of the Prague Stringology Conference '06*, pages 65–76, Czech Technical University in Prague, Czech Republic, 2006.
- [7] Karel Culik. An aperiodic set of 13 Wang tiles. *Discrete Mathematics*, 160:245–251, 1996.
- [8] Ian Gambini and Laurent Vuillon. An algorithm for deciding if a polyomino tiles the plane by translations. Technical report, LAMA, 2003.
- [9] Martin Gardner. More about tiling the plane: The possibilities of polyominoes, polyiamonds, and polyhexes. *Scientific American*, August 1975:112–115.

- [10] Solomon Golomb. *Polyominoes*. Scribner, New York, 1965.
- [11] Branko Grünbaum and G. C. Shephard. *Tilings and Patterns*. W. H. Freeman and Company, New York, 1987.
- [12] Heinrich Heesch. Aufbau der ebene aus kongruenten bereichen. *Nachr. Ges. Wiss.*, New Ser. 1:115–117, 1935.
- [13] Heinrich Heesch. *Reguläres Parkettierungsproblem*. Westdeucher Verlag, Cologne and Opladen, 1968.
- [14] Heinrich Heesch and O. Kienzle. *Flächenschluss. System der formen lückenlos aneinanderschliessender flachteile*. Springer, Berlin, 1963.
- [15] Iwan Jensen. Enumerations of lattice animals and trees. *ArXiv Condensed Matter e-prints*, July 2000.
- [16] Neal Madras and Gordon Slade. *The Self-Avoiding Walk*. Birkhäuser Boston, 1996.
- [17] Casey Mann. Heesch’s tiling problem. *Amer. Math. Monthly*, 111(6):509–517, 2004.
- [18] Stephan Mertens. Lattice animals: A fast enumeration algorithm and new perimeter polynomials. *J. Stat. Phys.*, 58(5/6):1095–1108, 1990.
- [19] Stephan Mertens and M. E. Lautenbacher. Counting lattice animals: A parallel attack. *J. Stat. Phys.*, 66(1/2):669–678, 1992.
- [20] Joseph Myers. Personal communication. <http://tiling.uttyler.edu/read/messages?id=215>.
- [21] Joseph Myers. Polyomino, polyhex, and polyiamond tiling. <http://www.srcf.ucam.org/jsm28/tiling/>.
- [22] Roger Penrose. The role of aesthetics in pure and applied mathematical research. *Bull. Inst. Math. Appl.*, 10:266–271, 1974.
- [23] D. Hugh Redelmeier. Counting polyominoes: Yet another attack. *Discrete Math.*, 36:191–203, 1981.

- [24] Glenn C. Rhoads. Planar tilings by polyominoes, polyhexes, and polyiamonds. *J. Comput. Appl. Math.*, 174(2):329–353, 2005.
- [25] Raphael M. Robinson. Undecidability and nonperiodicity of tilings of the plane. *Inventiones Math.*, 12:177–909, 1971.
- [26] Doris Schattschneider. Will it tile? Try the Conway Criterion! *Mathematics Magazine*, 53:224–233, 1980.
- [27] Doris Schattschneider. *M.C. Escher: Visions of Symmetry*. Harry N. Abrams, 2004.
- [28] Rolf Stein. A new pentagon tiler. *Mathematics Magazine*, 58:308, 1985.